These service guides reflect the current version of the Fastly web interface. Service guides for the next web interface can be accessed at docs-next.fastly.com (https://docs-next.fastly.com). For more information see our announcement (/changes/2016/07/20/special).

Generated: Sat, 13 Aug 2016 16:18:45 +0000

Fastly Help Guides

• Guides (/guides/) > Account management (/guides/accounts) > User access and control (/guides/user-access-and-control/)

§ Adding and deleting user accounts (/guides/user-access-and-control/adding-and-deleting-user-accounts)

Fastly allows you to add users to your account and assign them different roles (/guides/user-access-and-control/user-roles-and-how-to-change-them), delete user accounts when you no longer want them to have access, or add existing users to existing accounts.

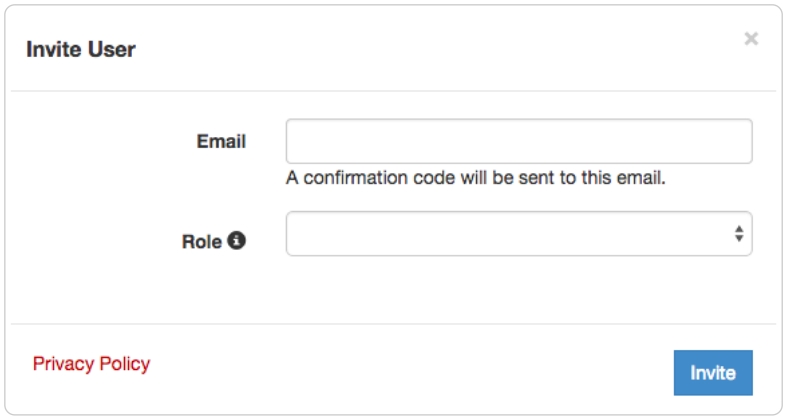
IMPORTANT: You must be assigned the role of super user (/guides/user-access-and-control/user-roles-and-how-to-change-them) in order to add or delete user accounts.

Adding new account users

TIP: Adding a new user to make them the billing contact for your account? Follow our billing contact instructions (/guides/account-types-and-billing/who-receives-your-bill#changing-who-receives-your-bill) instead.

To add a user to your account, send them an invitation to join following the steps below:

1. Log in to the Fastly application.
2. Click the **account** tab to access your Account Settings.
3. In the **User Invitations** area, click the **Invite** button. The **Invite User** window appears.



4. In the **Email** field, type the email address of the user you'd like to invite.
5. From the **Role** menu, select the role to assign (/guides/user-access-and-control/user-roles-and-how-to-change-them) once the user accepts the invitation.
6. Click the **Invite** button to send an invitation to the email you specified.

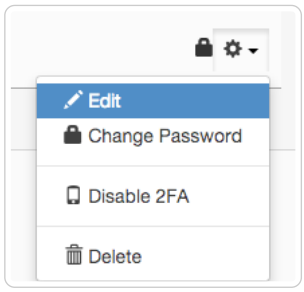
Deleting existing account users

TIP: Deleting the owner of the account? Be sure to transfer ownership (/guides/user-access-and-control/user-roles-and-how-to-change-them#account-ownership-and-how-to-transfer-it) first.

To delete a user from your account, follow the steps below:

1. Log in to the Fastly application.
2. Click on the **account** tab. The Account Settings controls appear.
3. Scroll down to the **Users** area and search for the name of the user you want to delete.

4. Click the gear icon to the right of the user's name and select **Delete** from the menu that appears.



A confirmation window appears.

5. Click **Confirm** on the confirmation window.

Adding existing account users

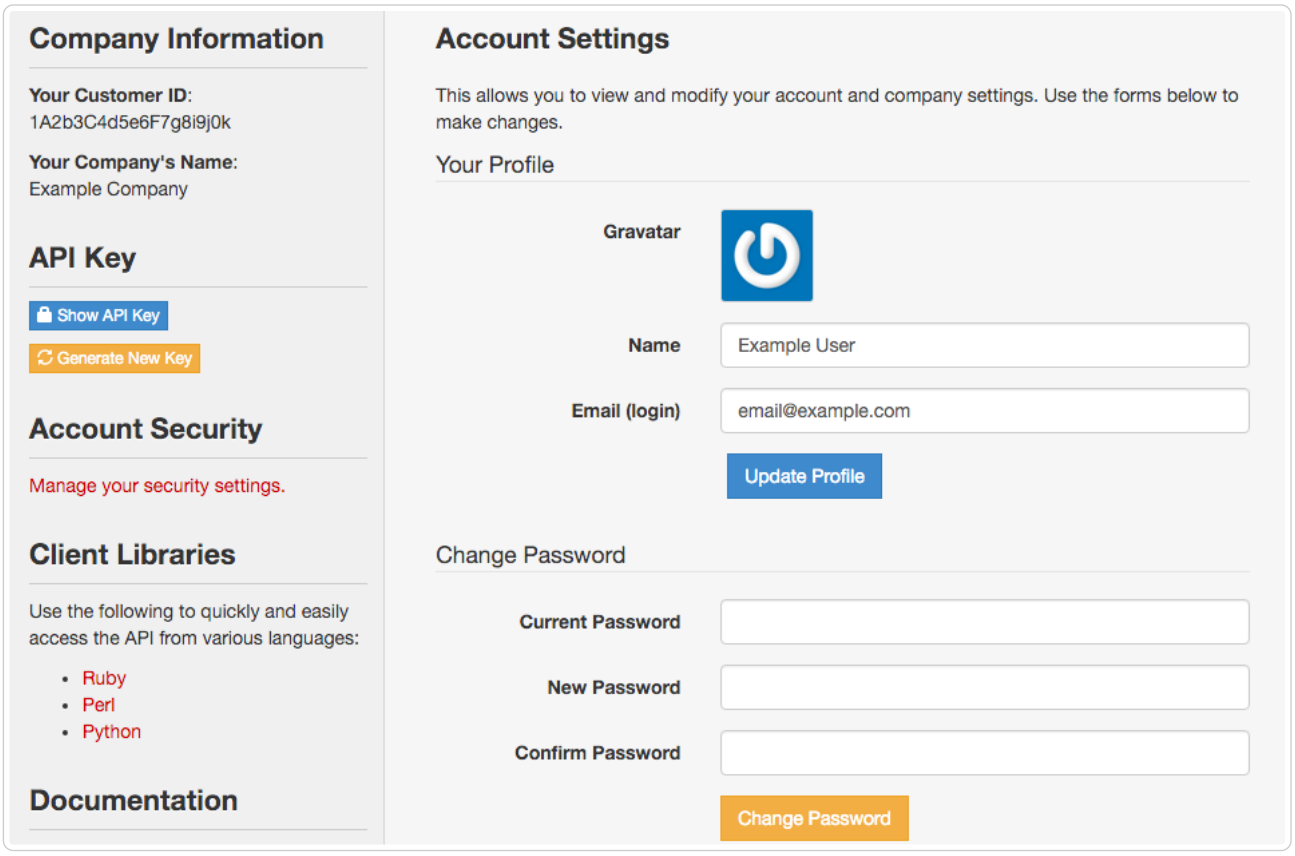
There might be times when you want to invite a user who already has a Fastly account to your account. To add an existing user, follow the steps below:

1. Have the existing user you're inviting cancel their account (/guides/account-types-and-billing/accounts-and-pricing-plans#canceling-your-account) so you can invite them to yours. We tie a user's email address to the creation of an account. Canceling that account allows the email address to be reused.
2. Log in to the Fastly application.
3. Re-invite the user to your account.

NOTE: Account cancelation might not be an option in some situations. Contact support@fastly.com (mailto:support@fastly.com) to discuss how accounts can be combined.

§ Email and password changes (/guides/user-access-and-control/email-and-password-changes)

Log in to the Fastly application and click the **account** tab to access the Account Settings (/guides/about-fastly-services/about-the-web-interface-controls#about-the-account-settings-controls) information.



Changing your name or email

To change the name or email associated with your account, update the information that appears in the **Name** and **Email (login)** fields in the **Your Profile** section. When changing your email, you'll need to confirm your password in the window that appears. To finalize your new name or email, click **Update Profile** to save the changes.

Changing your password

To change the password currently associated with your account, type your current password in the **Current Password** field, then type your new password twice, once in the **New Password** field and once in the **Confirm Password** field. Save the changes by clicking **Change Password**.

Password requirements

When choosing a password keep in mind that it must:

- be at least 6 characters long
- contain at least one letter and one number

In addition, passwords cannot solely contain:

- sequences of letters or numbers (e.g., `12345678`, `abcdefg`)
- repeated characters (e.g., `222222`, `aaaaaa`)
- adjacent key placements on a standard keyboard (e.g., `QWERTY`)

The system will prevent you from choosing a password that:

- matches commonly used passwords (e.g., `password123`, `changeme`)
- uses dictionary words (e.g. `correcthorsebatterystaple`)
- matches your user name or your email address

§ Merging accounts (/guides/user-access-and-control/merging-accounts)

If several employees at your company independently create testing accounts when learning about Fastly services (/guides/about-fastly-services/), you can have those testing accounts merged into a single account by emailing the Customer Support team at support@fastly.com (mailto:support@fastly.com) with the following information:

- the Customer IDs (/guides/account-management-and-security/finding-and-managing-your-account-info) of the accounts to be merged
- which account should be considered the primary account (any other accounts will be merged into the primary)

After you contact us, we'll reach out to verify the ownership of each account. If we can confirm ownership, we'll initiate a merge.

§ User roles and how to change them (/guides/user-access-and-control/user-roles-and-how-to-change-them)

Your Fastly account can be managed by multiple users. Users can be added to your account by by invitation (/guides/user-access-and-control/adding-and-deleting-user-accounts), with their roles assigned and changed by **superusers** at any time via the web interface or via the API (/api/).

Assignable user roles

Fastly allows you to assign one of four different roles to each user who has access to your account:

	User	Billing	Engineer	Superuser
Analytics & Stats				
Select a service	X	X	X	X
Select a data center	X	X	X	X
Configure				
Create and delete services			X	X
Configure services			X	X
Compare service versions			X	X
Deactivate services			X	X
Purge			X	X
View and download generated VCL			X	X
Customize VCL			X	X
Account & Organization				
Update personal profile settings	X	X	X	X
Update company settings				X
Issue user invitations				X
Assign user roles				X
Issue password resets				X
Delete user accounts				X
Enable and disable personal 2FA	X	X	X	X
Enable and disable company-wide 2FA				X
View and generate API keys			X	X
Billing				
View invoices		X		X
View billing history		X		X
Pay bills		X		X
Update credit card info		X		X
Change account type		X		X

By default, the system assigns the role of **superuser** to the person who signs up for your organization's account.

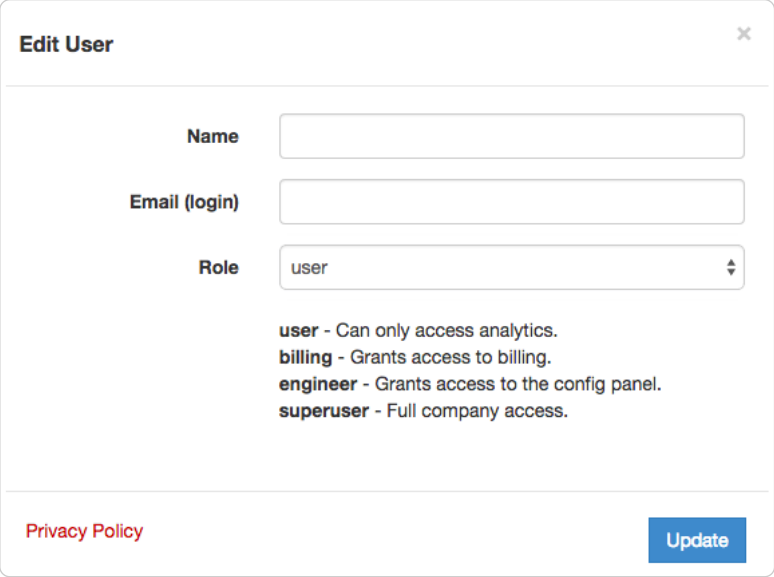
NOTE: The role you assign to users does not affect their ability to submit requests to Fastly Customer Support (/guides/customer-support/submitting-support-requests).

Changing roles for existing users

Follow these instructions to change the roles for existing users:

1. Log in to the Fastly application.
2. Click the **account** tab. The Account Settings page appears.

3. In the **User** area, click the gear icon next to a user and then select **Edit**. The Edit User window appears for the selected user.



4. From the **Role** menu, select a role for the user.
5. Click **Update**. The user's role will be changed.

Account ownership and how to transfer it

By default, we assign account "ownership" to the first user who signs up for an account for your organization. Accounts can only be canceled (/guides/account-types-and-billing/accounts-and-pricing-plans#canceling-your-account) by owners. As the first user of an account, we automatically assign the owner the superuser role, though that role can be changed by another superuser once additional users are added.

An account owner serves as the primary point of contact for billing purposes. Invoices are sent to them, but if a specific billing contact (/guides/account-types-and-billing/who-receives-your-bill) has been defined, invoices go to that contact instead.

To transfer account ownership to another user, please contact support@fastly.com (mailto:support@fastly.com) for assistance.

• Guides (/guides/) > Account management (/guides/accounts) > Account management and security (/guides/account-management-and-security/)

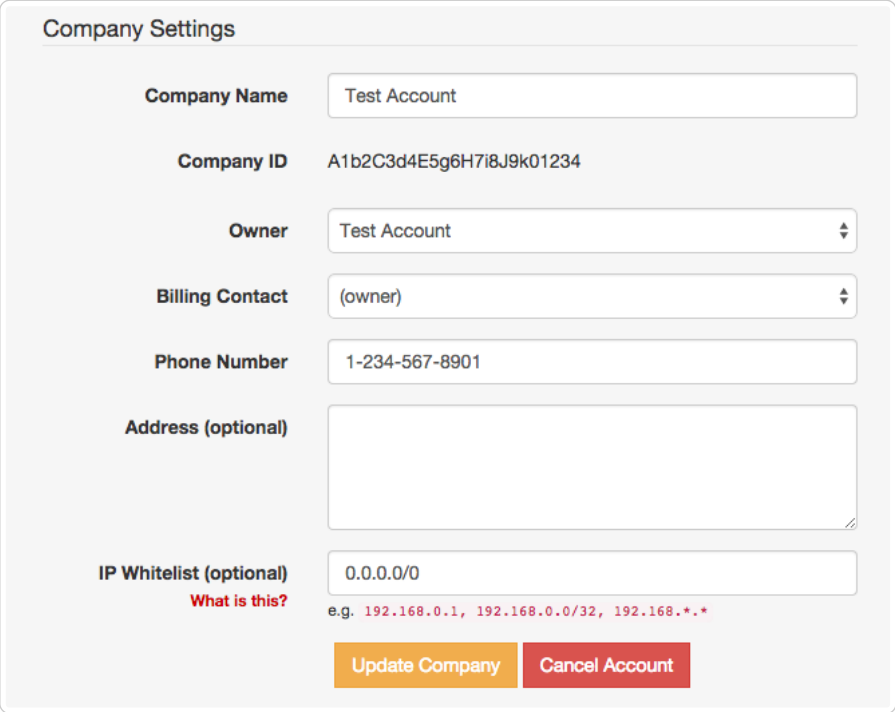
§ Changing your account's company name (/guides/account-management-and-security/changing-your-accounts-company-name)

Fastly allows you to change your account's company name at any time after it's been created.

IMPORTANT: You must be the account owner (/guides/user-access-and-control/user-roles-and-how-to-change-them#account-ownership-and-how-to-transfer-it) or be assigned the role of superuser (/guides/user-access-and-control/user-roles-and-how-to-change-them) to change your account name.

To change the company name, follow the steps below.

1. Log into the Fastly application (https://app.fastly.com)
2. Click the **account** tab. The Account Settings appear.
3. In the **Company Name** field of **Company Settings** area towards the middle of the page, replace the current company name.



4. Click the **Update Company** button and refresh your screen. The new company name appears in the Company Information area next to the Account Settings controls.

§ Enabling an IP whitelist for account logins (/guides/account-management-and-security/enabling-an-ip-whitelist-for-account-logins)

Fastly allows you to define the range of IP addresses authorized to access your Fastly account. This optional IP whitelisting functionality is not enabled by default.

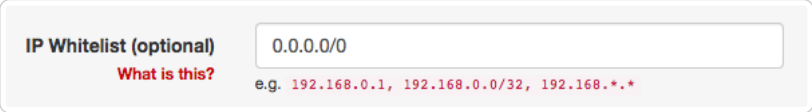
WARNING: ***If you decide to use optional IP whitelisting, your account owner must have a valid telephone number on file.*** During setup, Fastly checks your current IP address against the list you provide to ensure you don't lock yourself out of your account. If your IP addresses change at a later date (for example, because you move offices) and you forget to update your whitelist configuration, you will be locked out of your account. You will need to contact support@fastly.com (mailto:support@fastly.com) to request that a Customer Support representative contact your account's owner via telephone during Fastly's regular business hours. To protect your account's security, ***we will not unlock your account (/guides/customer-support/account-lockouts) based on an email request alone***.

Enabling an IP whitelist

To restrict access to your Fastly account based on a specific list or range of IP addresses, follow these steps.

1. Log in to the Fastly application.
2. Click the **account** tab to access the **Account Settings** information.

3. In the **IP Whitelist** field of the **Company Settings** area, replace `0.0.0.0/0` (the default IP range indicating no whitelisting) with the IP addresses to which your account access should be restricted.



   In the IP Whitelist field you can include single or multiple IP addresses or IP ranges (separated by commas) as follows:

   - a single IPv4 address (e.g., replace the default with `192.168.0.1`)
   - an IPv4 CIDR range (e.g., replace the default with `192.168.0.0/32`)
   - an IPv4 Wildcard range (e.g., replace the default with `192.168.0.*`, `192.168.*.1`, `192.168.*.*`)

4. Click the **Update Company** button.

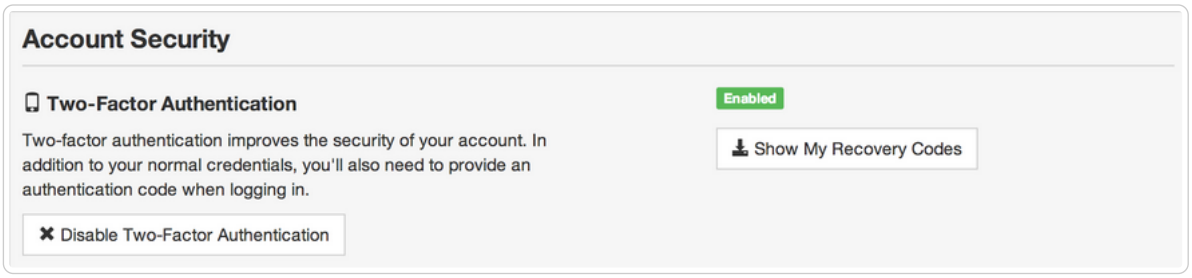
Disabling an IP whitelist

To disable IP whitelisting on your Fastly account, follow these steps.

1. Log in to the Fastly application.
2. Click the **account** tab to access the **Account Settings** information.
3. In the **IP Whitelist** field of the **Company Settings** area, type `0.0.0.0/0` (the default IP range indicating no whitelisting).
4. Click the **Update Company** button.

§ Enabling and disabling two-factor authentication (/guides/account-management-and-security/enabling-and-disabling-two-factor-authentication)

Fastly supports two-factor authentication, a two-step verification system, for logging in to the application. In a two-factor authentication security process, users provide two means of identifying themselves to the system, typically by providing the system with something they know (for example, their login ID and password combination) and something they have (such as an authentication code). Organizations can enable company-wide two-factor authentication to require all users within the organization to use two-factor authentication.



Before you begin

You'll need to enter an authentication code regularly. Once two-factor authentication has been enabled, an authentication code will be requested upon login at least every 14 days for each computer and browser you use to access the Fastly application.

A mobile device is required. Using this security feature with a Fastly account requires a mobile device capable of scanning a barcode or QR code using a downloadable authenticator application. We recommend the following:

- For Android, iOS, and Blackberry: Google Authenticator (https://support.google.com/accounts/answer/1066447?hl=en)
- For Android and iOS: Duo Mobile (https://guide.duo.com/third-party-accounts)
- For Android: AWS Virtual MFA (http://www.amazon.com/Amazon-com-AWS-Virtual-MFA/dp/B0061MU68M)
- For Windows Phone: Authenticator (https://www.microsoft.com/en-us/store/apps/authenticator/9nblggh09llf)

There are special requirements for using this feature with the API. If you enable two-factor authentication via the user interface, you will no longer be able to use a username and password when using the API. You must use the organization's API key for authentication.

Managing two-factor authentication as a user

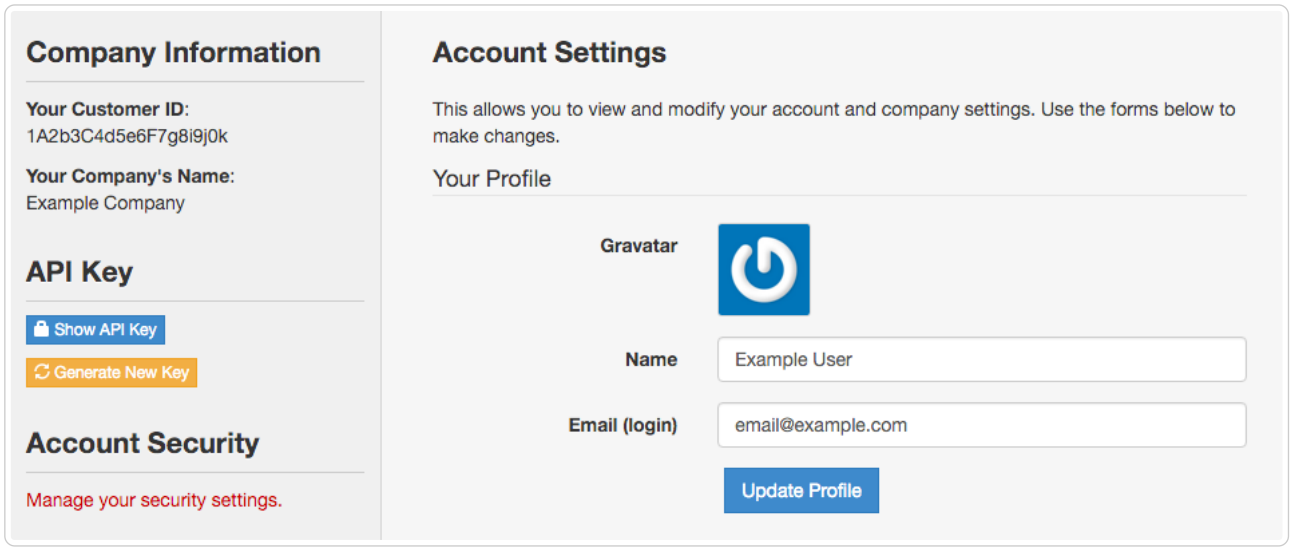
Depending on whether or not your organization has enabled company-wide two-factor authentication, you may be able to enable and disable two-factor authentication for your personal account. We also have instructions for recovering access to your account if you lose your mobile device.

Enabling two-factor authentication

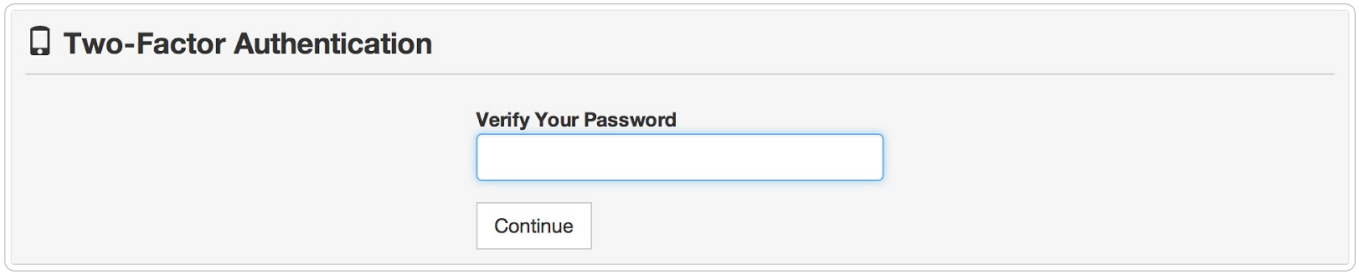
To enable two-factor authentication for your user account, follow the steps below.

IMPORTANT: If your organization has enabled company-wide two-factor authentication, you will be required to set up two-factor authentication when you log in to the Fastly application. Skip to step six for instructions.

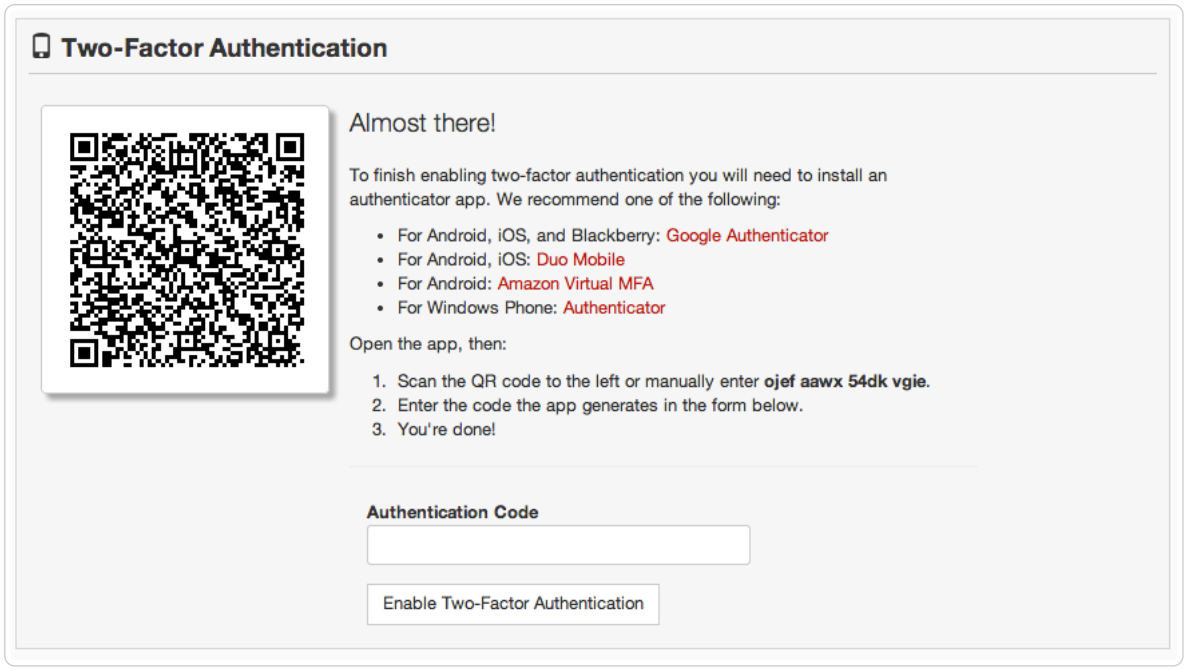
1. Log in to the Fastly application.
2. Click the **account** tab to access the account settings.



3. Click **Manage your security settings** from the **Account Security** area on the left. The Account Security settings appear.
4. Click **Set Up Two-Factor Authentication**. The password verification screen appears.



5. In the **Verify Your Password** field type your Fastly password and then click **Continue**. The authentication QR code appears.

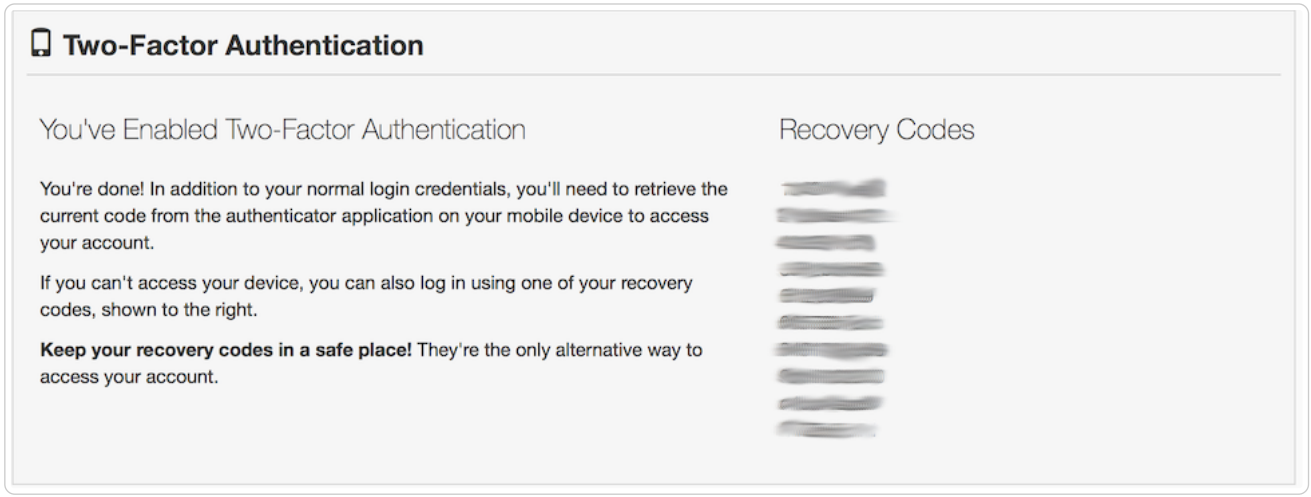


IMPORTANT: The QR code above is an example. Scan the one that appears in the Fastly application, not in this guide.

6. Launch the authenticator application installed on your mobile device and scan the displayed QR code or manually enter the key displayed in the setup window. A time-based authentication code appears on your mobile device. Depending on your device, however, a browser link may first appear. You need to click this link to save it. When you do, the words `Secret saved` appear briefly.
7. In the **Authentication Code** field, type the time-based authentication code displayed on your mobile device.

"ANDROID: A common time syncing issue may cause your authenticator codes to fail. You can correct this using Google's instructions (https://support.google.com/accounts/answer/185834?hl=en#sync) for your authenticator app.

8. Click **Enable Two-Factor Authentication**. The confirmation screen appears along with your recovery codes.



IMPORTANT: If you're ever unable to access your mobile device, the displayed recovery codes can be used to log in when your account has two-factor authentication enabled. Each of these recovery codes can only be used once, but you can regenerate a new set of 10 at any time (any unused codes at that time will be invalidated). Store your recovery codes in a safe place.

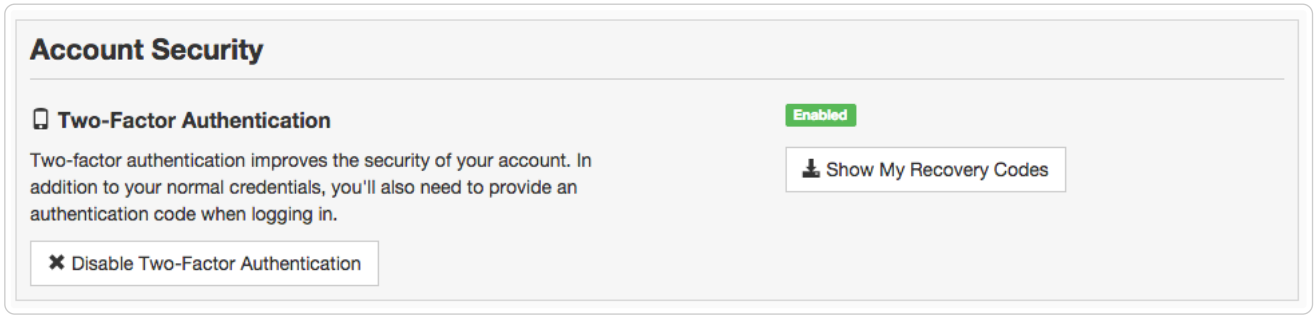
After you enable two-factor authentication, logging in to your Fastly account will require your email address and password, and then an authentication code generated by the authenticator application you've installed on your mobile device. By default, the system requires you to authenticate your login using an authentication code at least every two weeks for each computer and browser you use to access the Fastly application.

Disabling two-factor authentication

Once two-factor authentication is enabled for your account, you can disable it at any time by following the steps below.

IMPORTANT: If your organization has enabled company-wide two-factor authentication, you cannot disable two-factor authentication for your account.

1. Log in to the Fastly application.
2. Click the **account** tab to access the account settings.
3. In the **Account Security** area of the settings sidebar, click **Manage your security settings**. The two-factor authentication controls appear.



4. Click **Disable Two-Factor Authentication**. The verification screen appears.
5. In the **Authentication Code** field, type the time-based authentication code displayed in the authenticator application on your mobile device, then click **Disable**.

NOTE: If you have lost your mobile device, you can enter a recovery code in the **Authentication Code** field. For more information, see the section on what to do if you lose your mobile device.

What to do if you lose your mobile device

If you lose your mobile device after enabling two-factor authentication, use a recovery code to log in to your Fastly account. You can continue to use recovery codes to log in until you get your mobile device back. Recovery codes can only be used once, however, so remember to regenerate a new list of codes to avoid running out before you recover your mobile device.

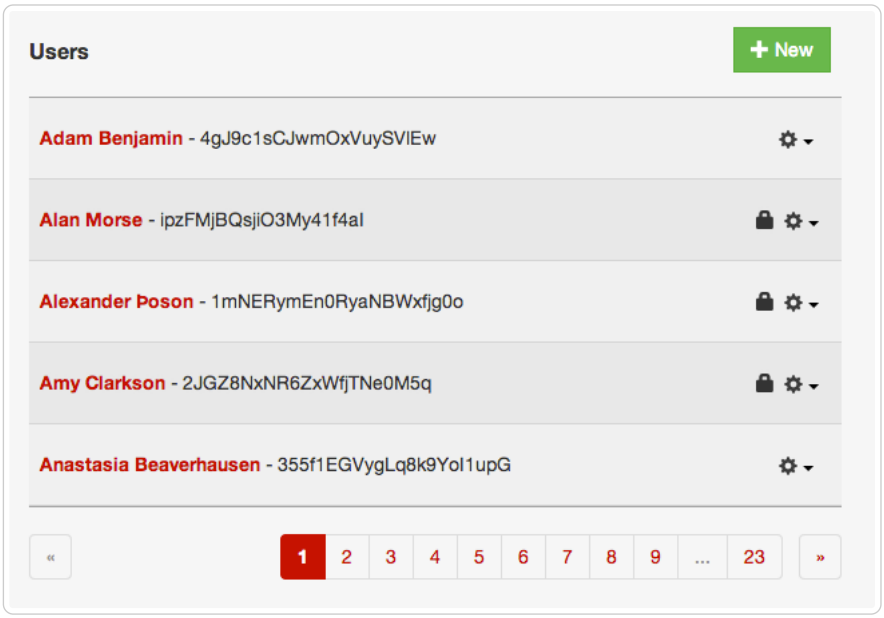
If you do not believe you will be able to recover your lost mobile device and you still have at least two recovery codes left, you can log in with one recovery code and disable two-factor authentication with a second code. Once two-factor authentication is disabled, you can re-enable it with a new mobile device at a later time and regenerate a new set of codes.

If your organization has enabled company-wide two-factor authentication, you can contact a superuser (/guides/user-access-and-control/user-roles-and-how-to-change-them) for your organization and ask them to reset your two-factor authentication.

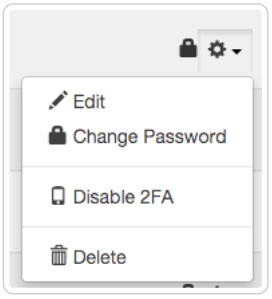
Locked out of your account? See our article on what you can do about it (/guides/customer-support/account-lockouts).

Managing two-factor authentication as a superuser

If you are assigned the superuser role (/guides/user-access-and-control/user-roles-and-how-to-change-them) for your organization, you can view who has two-factor authentication enabled on the account tab in the Users area of the Account settings. Users with this feature enabled have padlocks displayed next to their names.



To disable two-factor authentication for any user within your organization, select **Disable 2FA** from the menu that appears when you click the gear icon next to that user's name.



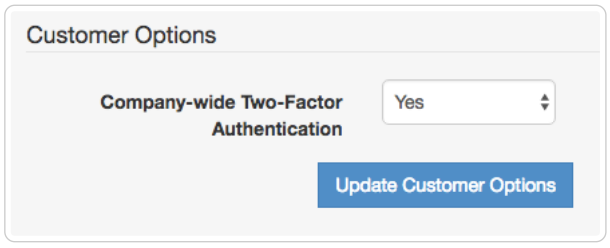
Managing two-factor authentication as a company

Organizations can enable two-factor authentication for all of their users. When the company-wide two-factor authentication feature is enabled, all users within the organization are required to use two-factor authentication to log in to the Fastly application, and they cannot disable two-factor authentication for their accounts.

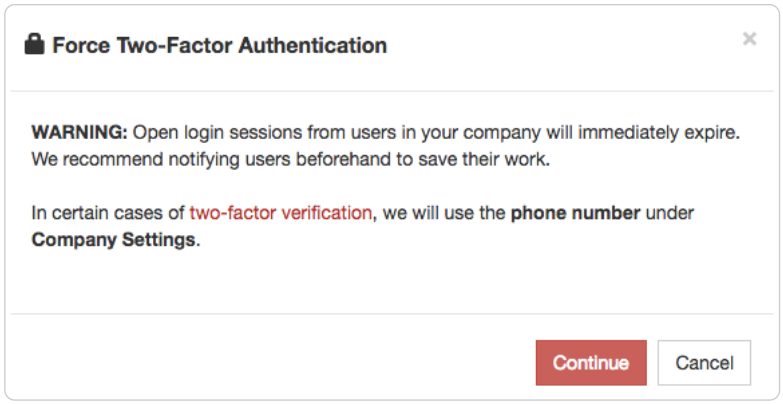
Enabling company-wide two-factor authentication

Users assigned the superuser role (/guides/user-access-and-control/user-roles-and-how-to-change-them) can enable this feature on the Account page. To enable company-wide two-factor authentication for all users within your organization, follow the steps below.

1. Log in to the Fastly application.
2. Click the **account** tab to access the account settings.
3. In the **Customer Options** area, select **Yes** from the **Company-wide Two-Factor Authentication** menu.



4. Click **Update Customer Options**. A warning message appears.



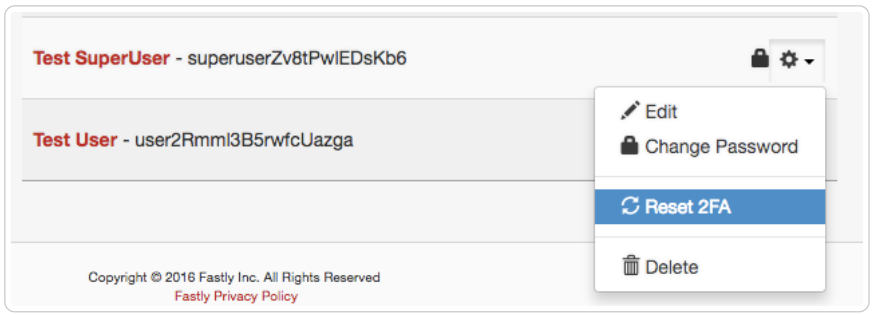
5. Click **Continue**. You will be logged out of the Fastly application. This completes the setup process for company-wide two-factor authentication.

Users who have not already enabled two-factor authentication for their accounts will be prompted to do so the next time they log in to the Fastly application.

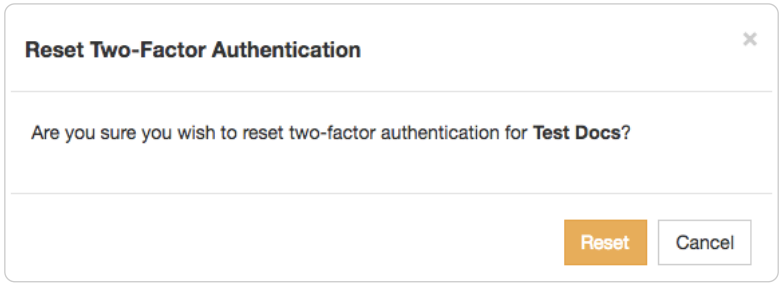
Resetting a user's two-factor authentication

If company-wide two-factor authentication is enabled, and a user within the organization gets locked out of their account or needs to enable a new device, a superuser (/guides/user-access-and-control/user-roles-and-how-to-change-them) can reset the user's two-factor authentication. To reset a user's two-factor authentication, follow the steps below.

1. Log in to the Fastly application.
2. Click the **account** tab to access the account settings.
3. In the **Users** area, click the gear icon next to a user and then select **Reset 2FA**.



A warning message appears.



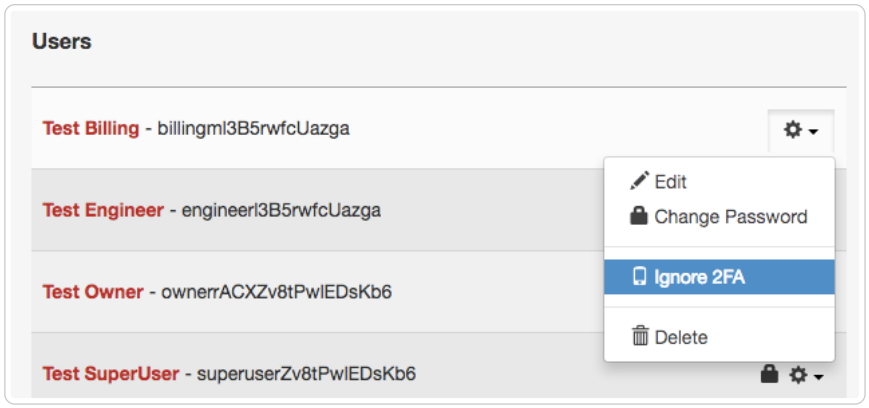
4. Click **Reset**.

The user will need to set up two-factor authentication for their account the next time they log in.

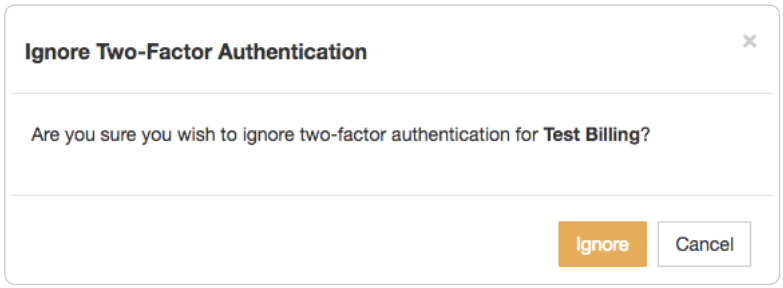
Disabling two-factor authentication for a single user's account

If company-wide two-factor authentication is enabled, a superuser (/guides/user-access-and-control/user-roles-and-how-to-change-them) can disable two-factor authentication for a single user's account. This is typically done for user accounts being used for scripts and session authentication. To disable two-factor authentication for a single user's account, follow the steps below.

1. Log in to the Fastly application.
2. Click the **account** tab to access the account settings.
3. In the **Users** area, click the gear icon next to a user and then select **Ignore 2FA**.



A warning message appears.



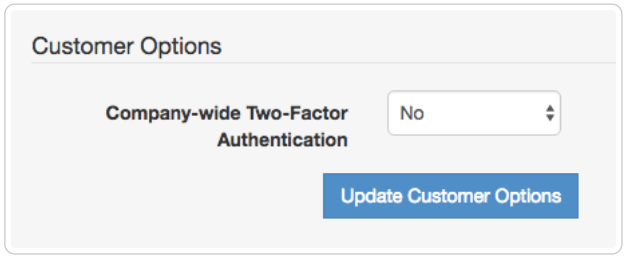
4. Click **Ignore**.

The user account you selected will no longer be required to use two-factor authentication.

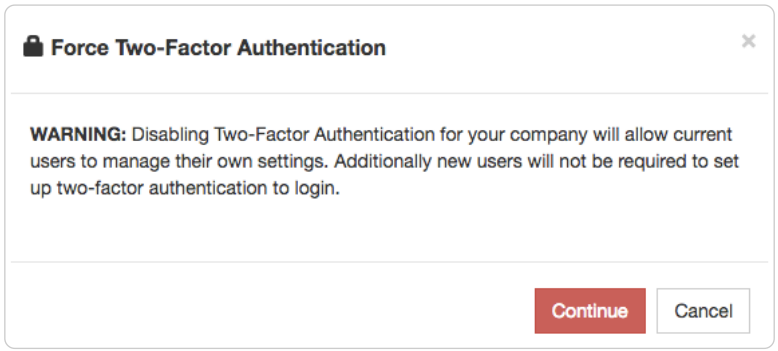
Disabling company-wide two-factor authentication

A superuser (/guides/user-access-and-control/user-roles-and-how-to-change-them) can disable company-wide two-factor authentication. Once this feature is disabled, existing users within the organization will be able to manage their own two-factor authentication settings, and new users will not be required to set up two-factor authentication to log in to the Fastly application. To disable company-wide two-factor authentication, follow the steps below:

1. Log in to the Fastly application.
2. Click the **account** tab to access the account settings.
3. In the **Customer Options** area, select **No** from the **Company-wide Two-Factor Authentication** menu.



4. Click **Update Customer Options**. A warning message appears.



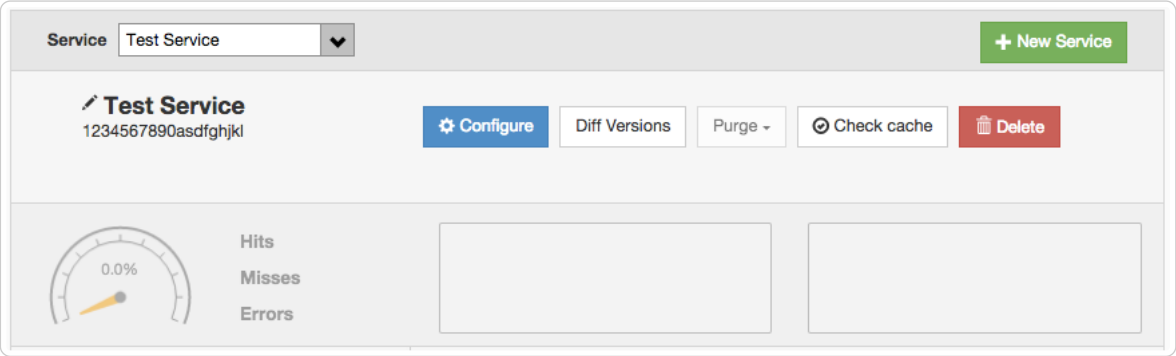
5. Click **Continue**.

§ Finding and managing your account info (/guides/account-management-and-security/finding-and-managing-your-account-info)

Account information, including your service ID, your customer ID (also called your company ID), and your API key, can be accessed directly from the Fastly application (https://app.fastly.com/).

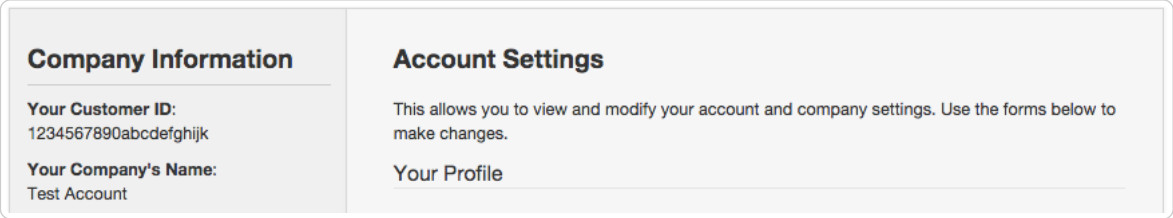
Finding your service ID

Your **Service ID** appears as an alphanumeric string immediately below the name of your service on the **configure** tab:



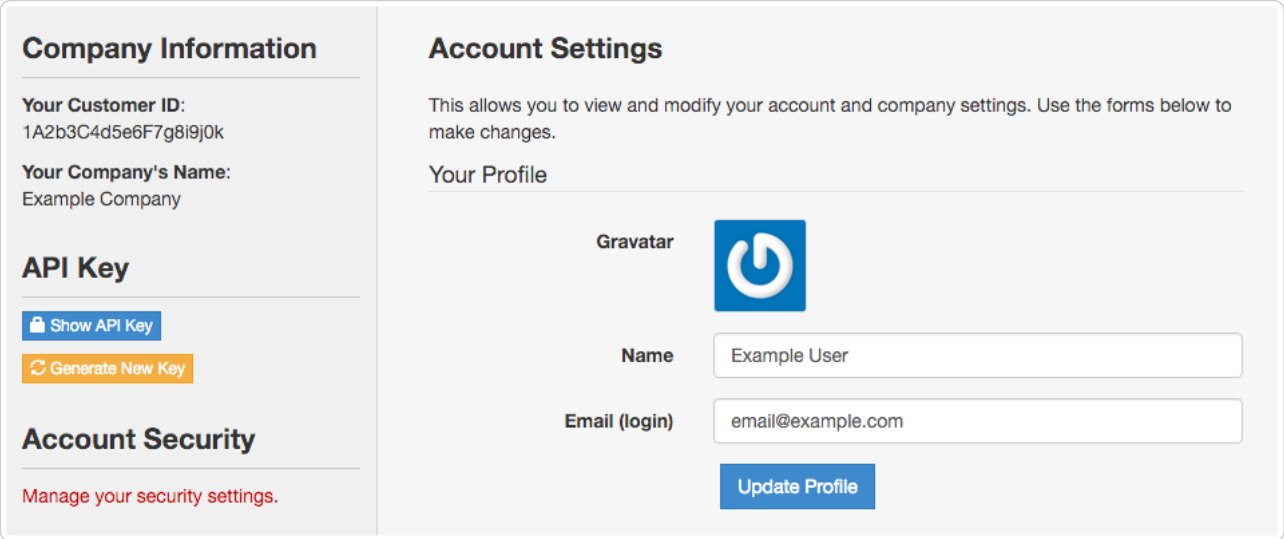
Finding your customer ID

Your **Customer ID** (also called your Company ID) is located on the **account** tab to the left of the **Account Settings** area:



Finding and regenerating your API key

Your **API Key** is located on the **account** tab to the left of the **Account Settings** area:



To display your API key, click the **Show API Key** button, type your password in the confirmation window, and click the **Confirm** button.

To regenerate your API key, click the **Generate New Key** button, type your password in the confirmation window, and click the **Confirm** button.

IMPORTANT: Only account users assigned the engineer or superuser roles (/guides/user-access-and-control/user-roles-and-how-to-change-them) can view and change the API key.

• Guides (/guides/) > Account management (/guides/accounts) > Account types and billing (/guides/account-types-and-billing/)

§ Accounts and pricing plans (/guides/account-types-and-billing/accounts-and-pricing-plans)

Types of accounts and plans

Fastly offers a variety of accounts and pricing plans.

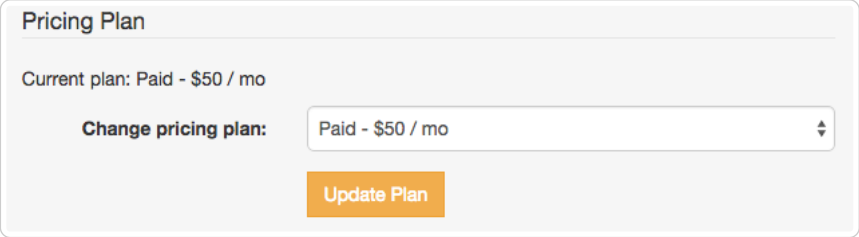
Free developer trials

We offer a development trial that allows you to test our services free of charge. We allow you to test up to $50 of traffic for free to ensure everything fits your requirements. Simply sign up for a trial (https://www.fastly.com/signup) and begin testing.

We don't limit functionality during your trial. Once your testing is complete and you're ready to start pushing production traffic our way, you can switch your account to a Paid account.

Paid accounts without contracts

After your trial period ends, you can use Fastly's services on a month-to-month basis without having to sign a contract. Simply switch to a paid account by selecting "Paid" from the pricing plan menu on the billing tab (/guides/about-fastly-services/about-the-web-interface-controls#about-the-billing-and-invoice-controls) (under the Pricing Plan area of the Settings section) while logged into your account.



When you do switch to a paid account, be sure to enter your current billing address (/guides/account-types-and-billing/paying-your-bill#changing-your-billing-address) and your credit card information since these are both required.

Once you switch to a paid plan, the development trial option will disappear and we'll bill you automatically (/guides/account-types-and-billing/how-we-calculate-your-bill) at the end of every month using the credit card information (/guides/account-types-and-billing/paying-your-bill#changing-your-credit-card-%20information) you have on file with us. You can cancel this account at any time.

Contractual commitments

If you plan to push at least 2TB of data per month and require one of our TLS service options (/guides/securing-communications/ordering-a-paid-tls-option), or if you plan to push a minimum of 4TB of data per month, it might be worthwhile to consider a contract with Fastly. Please contact us at sales@fastly.com (mailto:sales@fastly.com) for more information. We also offer solutions (https://www.fastly.com/services/web-mobile-performance) targeted to the needs of specific industries (for example, our Ecommerce package (https://www.fastly.com/industries/ecommerce)).

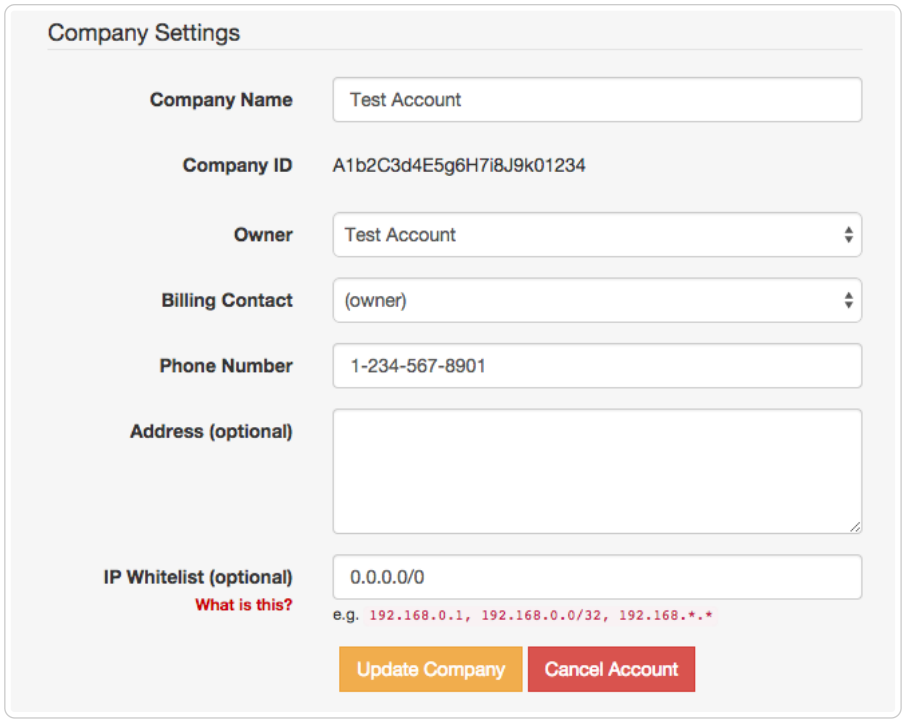
Free open source developer accounts

We're active open source contributors and support the community (https://www.fastly.com/open-source) whenever possible. If you're an open source developer, your bill is on us. Contact us at community@fastly.com (mailto:community@fastly.com) to get started.

Canceling your account

To cancel your account, have your account owner (/guides/user-access-and-control/user-roles-and-how-to-change-them#account-ownership-and-how-to-transfer-it) follow the steps below:

1. Log into the Fastly application and click the **configure** tab. The Control Panel appears.
2. Delete all services (/guides/basic-setup/working-with-services#deleting-a-service-version-or-an-entire-service) associated with your account.
3. Click the **account** tab. The Account Settings appear.
4. In the **Company Settings** area, click the **Cancel Account** button.



A Cancel Account confirmation window appears.

5. In the **Your Password** field of the confirmation window, type the password associated with your account and click **Cancel Account**.

After your account is canceled, you'll be billed for any outstanding charges (/guides/account-types-and-billing/how-we-calculate-your-bill) accrued through the day you canceled. For questions about your final billing statement, contact our billing team (mailto:billing@fastly.com) for assistance. If you decide at a later date to reactivate your account, contact Customer Support (mailto:support@fastly.com) and request reactivation.

§ How we calculate your bill (/guides/account-types-and-billing/how-we-calculate-your-bill)

What we do and don't charge you for

We bill you monthly according to that month's use of Fastly's services. Your monthly charges represent the combined total of your bandwidth and request usage, using the rates listed on our pricing page (https://www.fastly.com/pricing). The minimum charge you pay each month is $50.

We only charge for egress traffic, or traffic that moves directly from Fastly to end users. This excludes traffic traveling to and from the origin. We base this charge on the response size in bytes (the header size plus the body size) and how often a response occurs.

Two specific settings related to responses may affect the total charges on your bill. Enabling gzipping (/guides/basic-configuration/enabling-automatic-gzipping) can reduce the size of responses which reduces the bandwidth you use and thus can reduce your total bill. Enabling shielding (/guides/performance-tuning/shielding) may initially result in greater bandwidth use because requests may need to travel between POPs. The reduced load on your origin servers, however, frequently offsets this increased cost and the potential increase in your bill's total.

Charges for TLS service options (/guides/securing-communications/ordering-a-paid-tls-option) you've chosen are applied in addition to the bandwidth and request usage we charge for normal content delivery and streaming. We don't bill you for the amount of space your content takes up on our cache servers.

We don't charge you for Standard Support (/guides/customer-support/support-description-and-sla) service.

About the monthly minimum charges

We bill a minimum of $50 per month so we can fully support all of our customers. This is the minimum price you'll pay in any month once you've completed your testing trials (/guides/account-types-and-billing/accounts-and-pricing-plans).

For example, say that you're done testing Fastly's services and you've begun to push production-level traffic through Fastly. If most of your site's traffic for the current month is in North America and Europe and your site uses 10GB of traffic over 10 million requests, the combined bandwidth and request charges would be $8.70 for the month. Because this amount falls below the $50 monthly minimum, we would charge you $50 for that month, not $8.70.

Asia-Pacific, Australia, New Zealand and Latin America prices listed are slightly higher. If most of your site's traffic were in these areas instead of North America and Europe, then at the above traffic levels your bandwidth and request usage charges would still fall below the monthly minimum and we would charge you $50 for that month.

NOTE: If you're using Fastly for content delivery via Heroku's cloud development services, please see Fastly's Heroku add-ons pricing plan (https://elements.heroku.com/addons/fastly) for additional details.

When we charge you for Fastly services

Fastly bills in arrears, not in advance, meaning that we bill you for services after you've used them, not before. For example, if you sign up for and start using Fastly services in January, the bill you receive in February reflects January's charges and services, your March bill reflects February's charges services, and so forth.

How account cancelation affects your bill

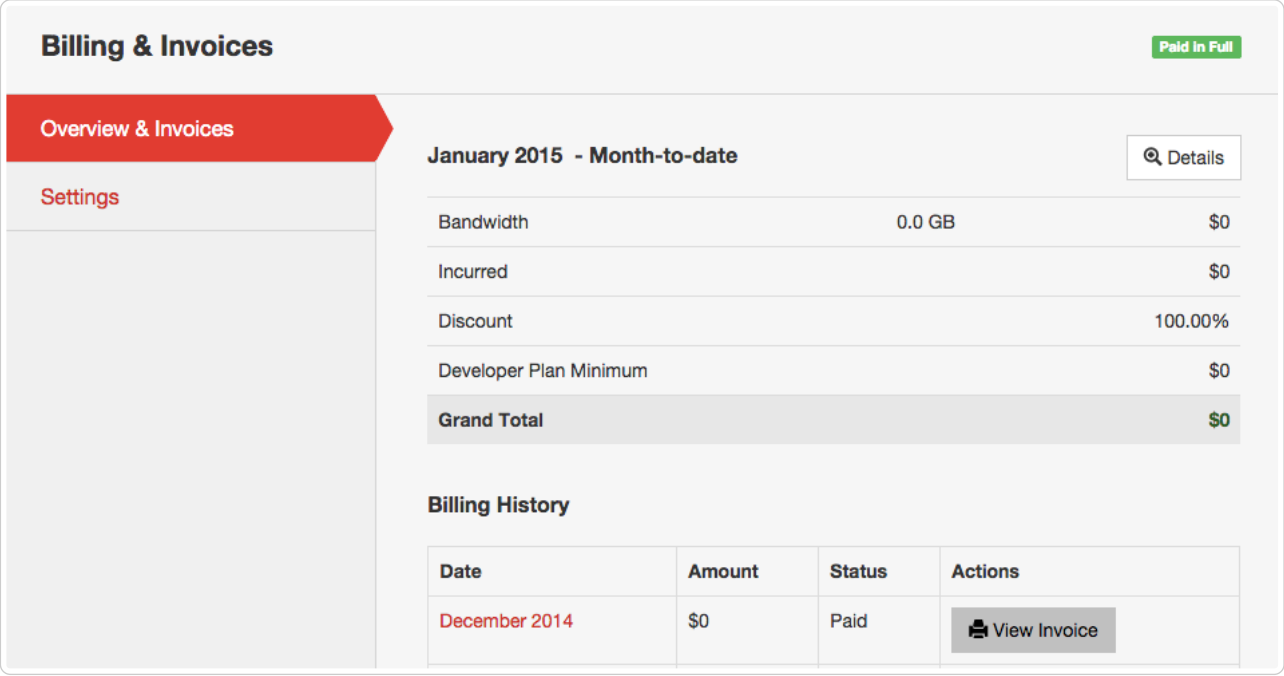
If you ever cancel your account (/guides/account-types-and-billing/accounts-and-pricing-plans#canceling-your-account), you'll be billed for any outstanding charges accrued through the day you canceled, or at least the monthly minimum, whichever amount is greater.

Reviewing the charges to your account

If you've been assigned a superuser or billing role (/guides/user-access-and-control/user-roles-and-how-to-change-them), you can review your account use and the associated charges via the billing tab (https://app.fastly.com/#billing) in the Fastly application.

Account use summaries

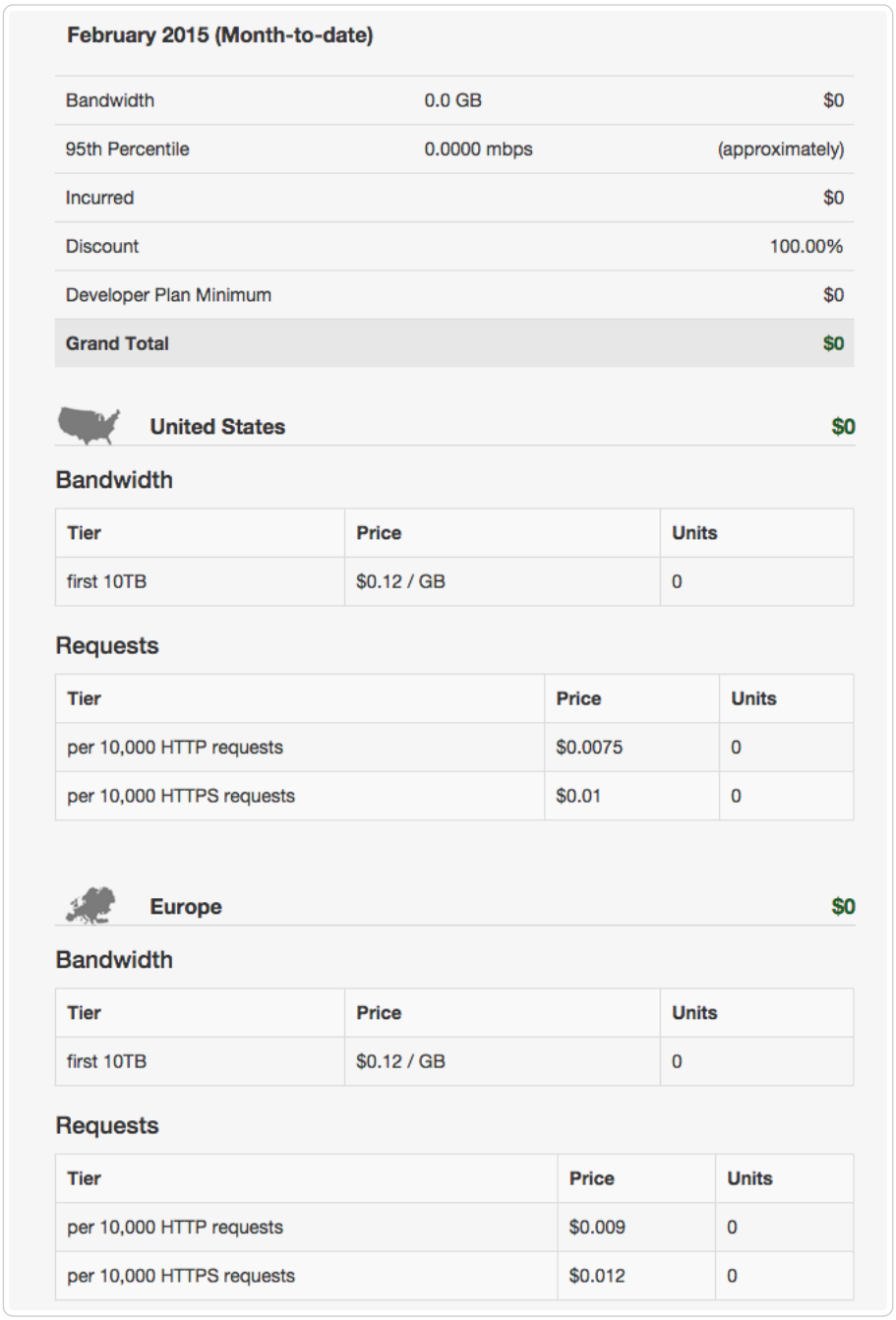
Billing and invoice information that appears by default when you click the billing tab. The Overview & Invoices summary lists the current month-to-date bandwidth you've used and the associated charges, followed by the charges you've incurred for requests for the current month. The bottom of the summary displays the grand total dollar amount owed for the month-to-date.



You can review similar details (including invoices) for previous months by clicking on the link for any month listed in the Date column of the Billing History section.

Regional account use details

To view specific regional use details for your account, click the Details button at the top-right of the Overview & Invoices summary for the current month-to-date. The Details page that appears includes the current month's tiered bandwidth and request use for each geographic region of the world.



February 2015 (Month-to-date)

Bandwidth	0.0 GB	$0
95th Percentile	0.0000 mbps	(approximately)
Incurred		$0
Discount		100.00%
Developer Plan Minimum		$0
Grand Total		**$0**

United States **$0**

Bandwidth

Tier	Price	Units
first 10TB	$0.12 / GB	0

Requests

Tier	Price	Units
per 10,000 HTTP requests	$0.0075	0
per 10,000 HTTPS requests	$0.01	0

Europe **$0**

Bandwidth

Tier	Price	Units
first 10TB	$0.12 / GB	0

Requests

Tier	Price	Units
per 10,000 HTTP requests	$0.009	0
per 10,000 HTTPS requests	$0.012	0

You can review similar regional, tiered bandwidth and request use details for previous months by clicking on the link for any month under the Date column of the Billing History section below the current month's billing summary.

Printing account use details

You can print account use details for any month directly from your browser window by viewing the specific summary or regional details and then selecting the Print option from your browser's File menu.

---

§ Paying your bill (/guides/account-types-and-billing/paying-your-bill)

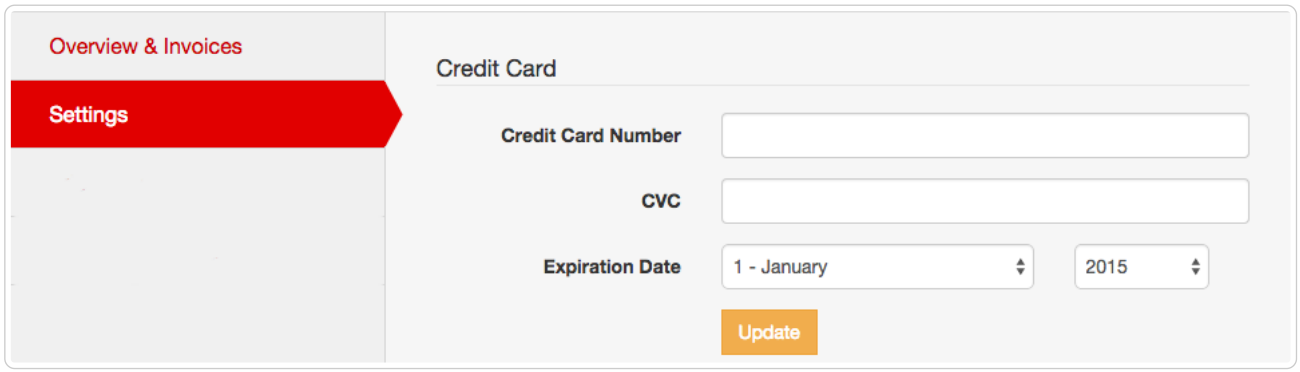
At the end of each month, your billing contact (/guides/account-types-and-billing/who-receives-your-bill) will be sent an email summarizing your current usage levels (/guides/account-types-and-billing/how-we-calculate-your-bill) and the charges your account incurred for the month. The email contains a link to the related invoice for your account.

You'll need both a valid credit card and current billing address when you switch to a paid, month-to-month account (/guides/account-types-and-billing/accounts-and-pricing-plans#paid-accounts-without-contracts). Once your invoice gets generated, your credit card is automatically charged.

Changing your credit card information

To change your credit card information, follow the steps below:

1. Log in to the Fastly application.
2. Click the **billing** tab. The Billings & Invoices controls appear.
3. Click the **Settings** pane from the list on the left.
4. In the **Credit Card** area of the **Settings** pane, make any necessary changes to the credit card number, card verification code, and expiration date for the credit card to be used for automatic billing.



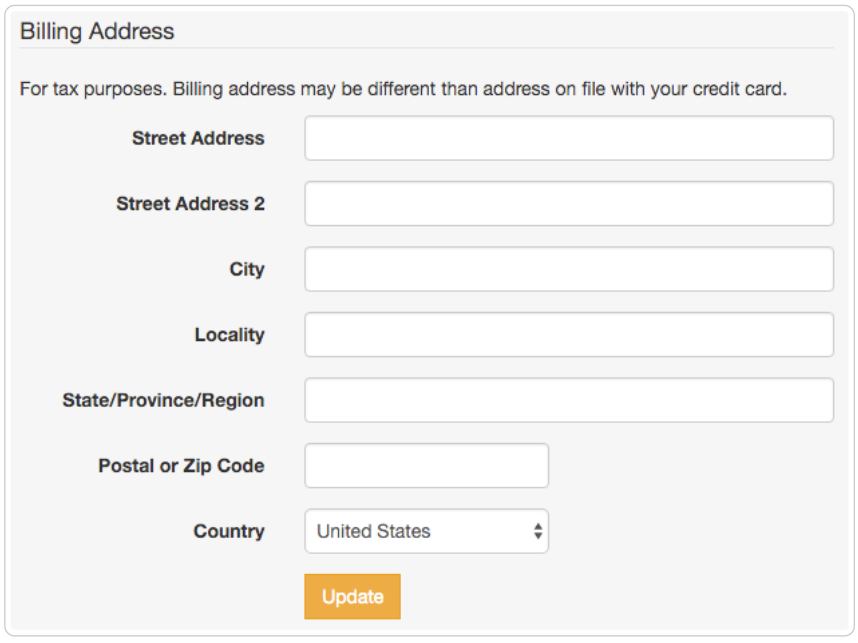
5. Click the **Update** button to save your credit card information.

★ **TIP:** Fastly never sees your credit card number. All transactions are handled by our fully PCI compliant payment gateway (https://stripe.com/docs/security), Stripe.

Changing your billing address

To change your billing address, follow the steps below:

1. Log in to the Fastly application.
2. Click the **billing** tab. The Billings & Invoices controls appear.
3. Click the **Settings** pane from the list on the left.
4. In the **Billing Address** area of the **Settings** pane, type the billing address.



5. Click the **Update** button to save the billing address.

§ Who receives your bill (/guides/account-types-and-billing/who-

receives-your-bill)

By default, your account owner is considered your billing contact and will receive your bill for Fastly services. You can change your billing contact at any time if you've been assigned the superuser role (/guides/user-access-and-control/user-roles-and-how-to-change-them) for an account. If you ever delete your billing contact, billing will automatically revert to the account owner.

IMPORTANT: Invoices are only sent to the email addresses of the Account Owner or the Billing Contact. Invoices are not sent every user assigned a billing role (/guides/user-access-and-control/user-roles-and-how-to-change-them).

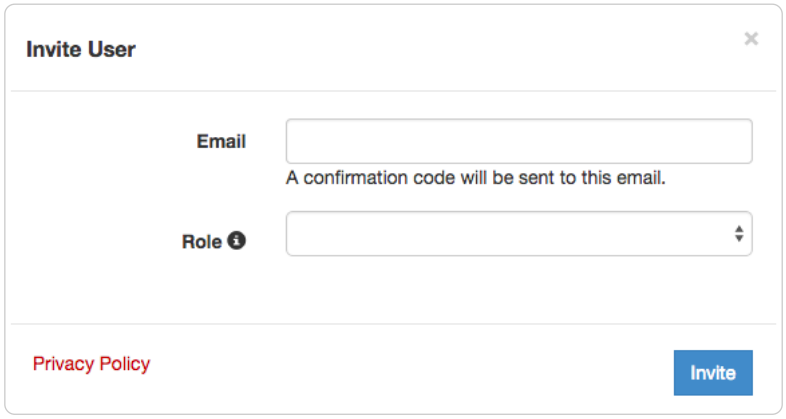
Changing who receives your bill

Follow the steps below to have your billing invoice sent to a person other than the owner of your account.

For new users

To send the billing invoice to a user who has not yet created an account, follow these steps.

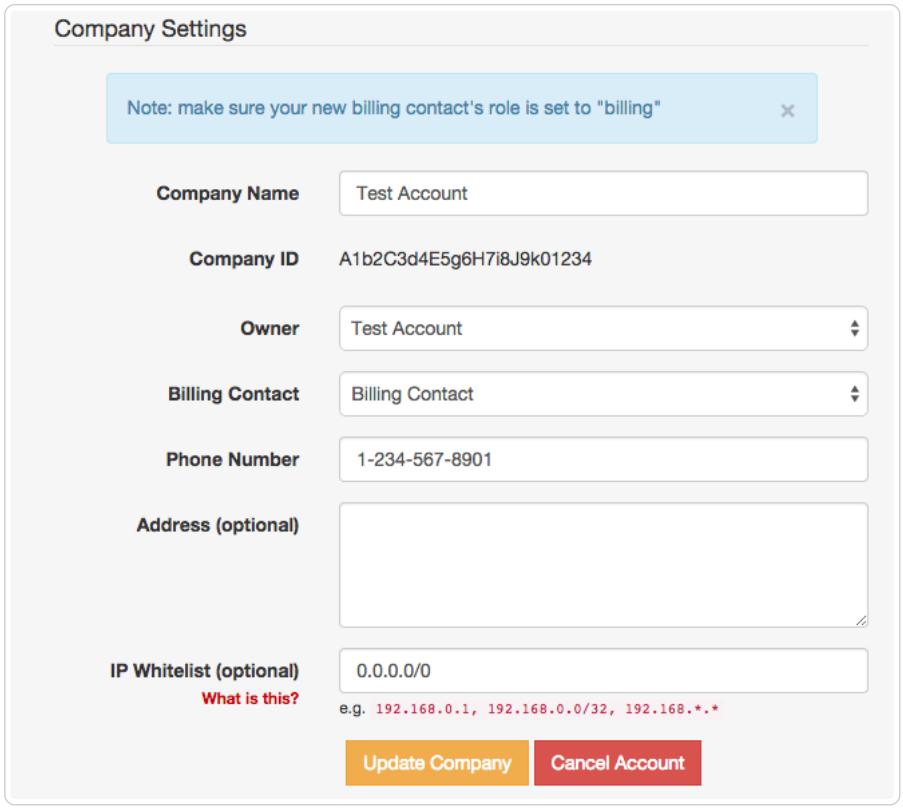
1. Log in to the Fastly application (https://app.fastly.com/).
2. Click the **account** tab to access your Account Settings.
3. In the **User Invitations** area, click the **Invite** button. The Invite User window appears.



4. In the **Email** field, type the email address of the user you'd like to invite to become a billing contact.

   ★ **TIP:** To send invoices to multiple people, we recommend setting up a group email address and setting that email address as your Billing Contact user.

5. From the **Role** menu, select the role to assign (/guides/user-access-and-control/user-roles-and-how-to-change-them) once the user accepts the invitation.
6. Click the **Invite** button to send an invitation to the email you specified.
7. Once the user has accepted the invitation, return to the **Account** tab in the application.
8. In the **Company Settings** area, select the user's name from the **Billing Contact** menu, and then click **Update Company** to set the billing contact.



For existing users

To send the billing invoice to a user who already has an account, follow these steps.

1. Log in to the Fastly application (https://app.fastly.com/).
2. Click the **account** tab to access your Account Settings.
3. In the **Company Settings** area, select the user's name from the **Billing Contact** menu, and then click **Update Company** to set the billing contact.
4. Make sure the user name you select has the correct role (/guides/user-access-and-control/user-roles-and-how-to-change-them) assigned to view and manage billing information.

• Guides (/guides/) > Account management (/guides/accounts) > Customer support (/guides/customer-support/)

§ Account lockouts (/guides/customer-support/account-lockouts)

Why is my account locked?

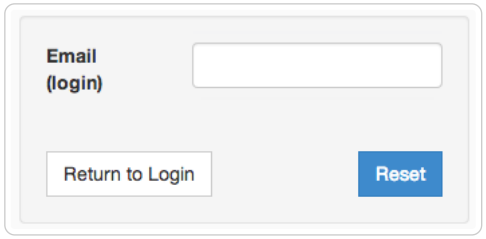
For security reasons, Fastly limits the number of times someone can try logging in to an account. We don't want to give people unlimited attempts at guessing your password, so we stop them from trying after a limited number of failed attempts to sign in. You can change your password (/guides/user-access-and-control/email-and-password-changes#changing-your-password) at any time when you're logged in to your account.

I am not using two-factor authentication. How can I access my account?

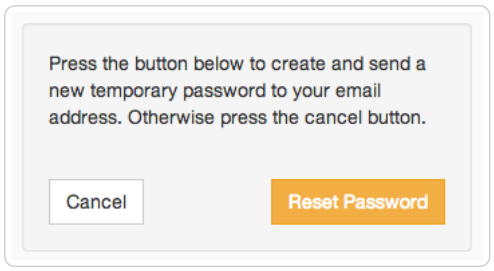
Once locked, you will not be able to sign in to your account, even with the correct password. To unlock your account because you exceeded the number of guesses you were allowed:

1. Point any standard web browser (/guides/debugging/browser-recommendations-when-using-the-fastly-application) to the Fastly application login page (https://app.fastly.com/#login). The login controls appear.

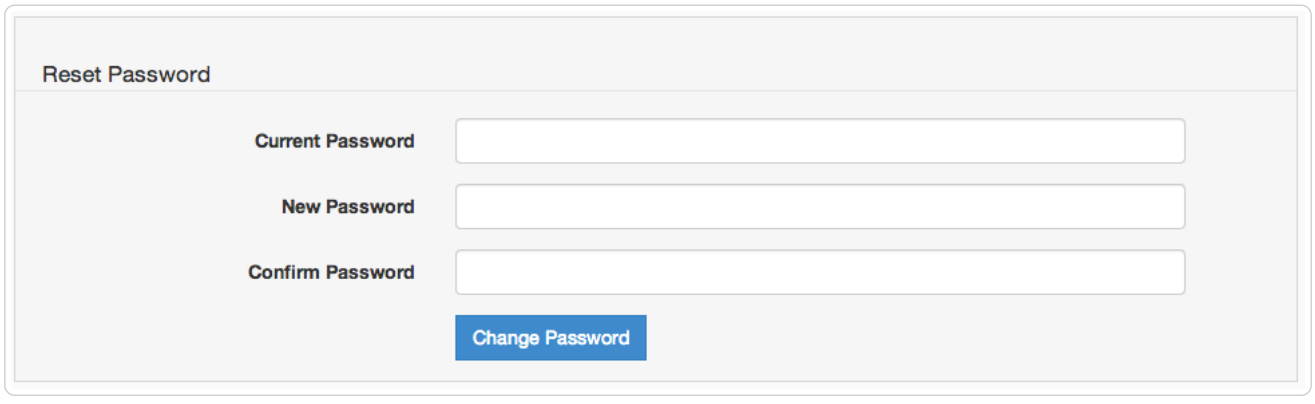
2. Click the **Forgot Password** button underneath the login controls. The Reset Password Controls appear.



3. In the **Email (login)** field, type the email address you normally use to log in to your Fastly account, and then click the **Reset** button. Password reset instructions will be emailed to you.
4. Click on the password reset link in the emailed instructions that the system sends you. The Reset Your Password page appears.



5. Click the **Reset Password** button. The system sends you a temporary password to the email address you supplied.
6. Using the temporary password you receive, log in to your account. The controls to create a new password appear.



7. Fill out the **Reset Password** controls as follows:
   - In the **Current Password** field, type the temporary password that the system emailed to you when you requested a password reset.
   - In the **New Password** field, type a new password to replace the temporary password you were sent.
   - In the **Confirm Password** field, type the new password a second time to confirm it.
8. Click the **Change Password** button. The system changes your password and logs you into your account.

I am using two-factor authentication. How can I access my account?

I don't have my mobile device.

If you do not have access to your mobile device, you can complete the login process using one of your recovery codes. These were the recovery codes you saved in a secured location outside of your Fastly account when two-factor authentication was first enabled. You can continue to use your recovery codes until your device is once again accessible. Recovery codes can only be used once, however, so remember to regenerate a new set to avoid running out before you recover your mobile device.

If you don't believe you will be able to recover your lost mobile device and you still have at least two recovery codes left, you can log in with one recovery code and disable two-factor authentication with a second code. Once two-factor authentication is disabled, you can re-enable it with a new mobile device at a later time and regenerate a new set of codes.

I don't have my mobile device and I don't have my recovery codes.

If you don't have your mobile device and didn't save any recovery codes, have another user at your company with the superuser role (/guides/user-access-and-control/user-roles-and-how-to-change-them) contact Customer Support at support@fastly.com (mailto:support@fastly.com). Have them inform Customer Support which user needs assistance with their login. After Customer Support verifies that the request is from a superuser, we will provide them with your recovery code. The superuser will then send you this information and reset your password so that you can access your account.

I don't have my phone, I didn't save my recovery codes, and I am the only superuser for the account.

Please contact Customer Support at support@fastly.com (mailto:support@fastly.com). We will verify that you are associated with the company by phone. We will use the contact information located on the company website or under the Fastly account tab. Upon verification, we will send you a recovery code and reset your password.

Was my account compromised?

If a user's account appears to be hacked or phished, we may proactively reset the passwords for the affected accounts to in order to revoke access to the hacker. In these cases, we send an email to the account's real owner (you) with additional information on how to reset the password. If you received one of these emails, please follow the instructions in the email.

If you think your account has been hacked or phished, please contact Customer Support at support@fastly.com (mailto:support@fastly.com) immediately.

How is a locked account different from a blocked account?

Fastly allows you to restrict who can access your Fastly account based on the IP address of the person attempting to log in. This means that even with the correct login name and password, access to your Fastly account may be blocked if the IP doesn't match your company's list of allowed addresses.

If your company enables this optional IP whitelisting (/guides/account-management-and-security/enabling-an-ip-whitelist-for-account-logins), they must keep the list of restricted IP addresses up to date. Only users with the role of superuser (/guides/user-access-and-control/user-roles-and-how-to-change-them) can make changes to the IP whitelist settings (your account owner is always a superuser), and your account owner must have a valid telephone number on file to do so.

If your IP addresses change after whitelisting is enabled and you forget to update your whitelist configuration, you will be locked out of your account. You will need to contact support@fastly.com (mailto:support@fastly.com) to request that a Customer Support representative contact your account's owner via telephone during Fastly's regular business hours. To protect your account's security, we will not unlock your account based on an email request alone.

---

§ Common service and domain errors (/guides/customer-support/common-service-and-domain-errors)

Exceeding max number of domains

We currently limit the maximum number of services and domains you can configure with the Fastly application. Once you reach that limit, error messages may appear that look something like this:

```
{
  "msg": "An error occurred while connecting to the fastly API, please try your request again.",
  "detail": "Exceeding max number of domains: 10"
}
```

If you're receiving a limit message and need to create more services or domains, please contact support@fastly.com (mailto:support@fastly.com) so we can help you. Fastly support engineers can not only increase the number of services that you can use, they can suggest other ways to design what you are trying to achieve.

§ Compliance Services (/guides/customer-support/compliance-services)

Subscribers who purchase Compliance Services will:

- have access to a library of third-party audit reports and certification attestations (most recent 12 months).
- have access to executive summary reports for penetration tests and network scans (most recent 12 months).
- have access to a library of security-related policies and procedures.
- have access to a library of executive summaries of annual risk assessments (most recent 12 months).
- have access to a library of historical Fastly Service Advisory (FSA) documents (most recent 12 months).
- be able to perform unlimited audits of Fastly's security (/guides/compliance/security-program) and technology compliance (/guides/compliance/technology-compliance) programs, subject to Subscriber's purchase of Professional Services (/guides/customer-support/professional-services). Audits require advance notice of at least 10 business days and shall be performed by Subscriber (or a mutually acceptable third party) according to standard audit practices.
- have the ability to be added as an Additional Insured on Fastly's General Commercial Liability Insurance for an additional fee.

Subscribers who wish to purchase Compliance Services must also purchase Gold or Platinum Support (/guides/customer-support/support-description-and-sla).

§ DDoS Protection and Mitigation Service and SLA (/guides/customer-support/ddos-protection-and-mitigation-service-and-sla)

Fastly offers DDoS Protection and Mitigation Service to customers with a sustained DDoS threat risk or with short term and seasonal events to protect. While the DDoS Protection and Mitigation Service cannot prevent or eliminate attacks or guarantee the uptime of your origin servers, it offers the following resources to assist you with mitigating the service and financial impacts of DDoS and related attacks.

Fastly's DDoS Protection and Mitigation Service includes:

- Immediate onboarding - We will work directly with you to immediately transition you to Fastly's CDN service if you're not already a customer.
- Emergency configuration and deployment support - We will actively work with you to configure your service map and provide an initial filter policy to immediately block an attack.
- Ongoing attack mitigation support - We will work directly with you to write custom VCL filters to deal with changing attacks or new attacks. We'll also isolate malicious traffic on your behalf.
- Incident response plan - We will create a plan that identifies how communication and escalation will occur between you and your staff and Fastly if an attack occurs. The plan will also describe mitigation and defense details such as any DDoS filters that we can insert into VCL prior to or during an attack.

Using our knowledge of attacks against our network and our customers, we analyze all DDoS Attack vectors using VCL statements, network filters, bulk traffic filtering through regional sinks, or a combination of these techniques.

The following table summarizes what is provided under our DDoS Protection and Mitigation Service:

Support offering	**Details**
Online self-service help	Unlimited access.
Availability for general inquiries	24/7.
Availability for incident reports	24/7.
Initial response times	Attack notification response within 15 minutes. Service onboarding beginning within 60 minutes of threat notification.

Overage Insurance	Included.
Access to Fastly IP Space	Included.
Web and email support	Available.
Phone and chat support	Toll-free telephone available 24/7/365. Dedicated chat channel available during Fastly Business Hours.
Emergency escalation	Available via email and phone support.

Technical support

The following section applies to all Subscribers of the DDoS Protection and Mitigation Service.

Definitions

- "Business Hours" are 8AM-6PM during a Business Day in California, New York, London, or Tokyo.
- "Business Days" are Monday through Friday, excluding any day that is simultaneously a US, UK, and Japanese national or banking holiday.
- A "DDoS Attack" is a Denial of Service (DoS) event (including Distributed Denial of Service (DDoS) or Distributed Reflection Amplification Denial of Service (DRDoS) attacks) that includes both an increase of unwanted traffic beyond two (2) times the average traffic of any Fastly Service for the preceding two (2) month period and a simultaneous increase in error responses from origin sites configured for any Fastly service. Fastly captures and analyzes suspected or actual DDoS Attack traffic to improve and protect its services.
- A "Fastly IP Space" is a published API endpoint (https://api.fastly.com/public-ip-list) that allows Subscribers to download an updated list of all Fastly IPs globally and can be used to filter traffic and control communication between Fastly's caches and a Subscriber's origin. Fastly provides the Fastly IP Space to Subscribers in order to ensure known communication between the Fastly cache nodes and a Subscriber's origin data center.
- "Fastly Control" means elements entirely under Fastly's control and not a consequence of (a) a Subscriber's hardware or software failures, (b) a Subscriber's or end user's connectivity issues, (c) Subscriber operator errors, (d) Subscriber traffic amounts that exceed a Subscriber's Permitted Utilization as defined in the Service Availability SLA (/guides/customer-support/service-availability-sla), (e) corrupted Subscriber content, (f) acts of god (any) or war, or earthquakes, or terrorist actions.

Subscriber responsibilities

As a Subscriber, you:

- must identify and maintain two points of contact to be used during an attack to communicate status, issues, and coordinate with Fastly to successfully protect services.
- must use common best practices for DDoS Attack defense including:
  - using updated white and black lists in the Fastly IP Space at the origin data center to protect against attack traffic bypassing Fastly's infrastructure.
  - limiting or eliminating your origin IP addresses from Domain Name System (DNS) records to avoid these addresses being used as attack targets.
- are responsible for using and configuring services according to the documentation available at https://docs.fastly.com (/guides/).

Support requests

Subscribers may make support requests by submitting a support ticket which will trigger a system-generated acknowledgement within minutes containing the ticket number and a direct link to the ticket.

DDoS Attack reports should include at least:

- a determination of the severity of the attack.
- the size of the attack threatened or previously observed.
- the type and vector of attack traffic seen or threatened.
- any duration of previous attacks and vector behavior including major source IP addresses.
- attack history for the last 24 months.
- threat specifics including all details of any attacks that the protected services or sites have experienced in the past.

Communications and channels of support

Support tickets

Create support tickets by sending an email to support@fastly.com (mailto:support@fastly.com), calling our dedicated phone line, or opening a web support ticket by selecting the Control Panel "SUPPORT" tab in the Fastly application. Filed tickets trigger Fastly's promised response time.

Tickets for communication between Fastly support engineers and a Subscriber's personnel are tracked using a ticketing application, which maintains a time-stamped transcript of communications, and sends emails to Subscriber and Fastly staff as tickets are updated.

Phone support

Subscribers to the DDoS Protection and Mitigation Service receive a dedicated phone number to contact Fastly support engineers. Fastly personnel can also establish audio and video conferencing (free app or browser plug-in required) for real-time voice and video communications.

Chat

To facilitate real-time communication, Subscribers to the DDoS Protection and Mitigation Service receive a dedicated chat channel for real-time communications during Business Hours or as needed by Fastly personnel. Though subject to change, Fastly's current chat provider is Slack (www.slack.com (https://slack.com/)).

Attack traffic

Response time

Fastly commits to responding to DDoS Attack notifications from Subscribers within 15 minutes of notice and, as applicable, will begin on-boarding Subscribers to the DDoS Protection and Mitigation Service within 60 minutes of a DDoS Attack notification.

Related Invoice Credits

Fastly will waive all bandwidth and request charges associated with DDoS Attack traffic and will provide Invoice Credits or adjustments for the same.

Attack traffic credit terms

Subscribers must submit claims for waiver of DDoS Attack-related charges to billing@fastly.com (mailto:billing@fastly.com) within 30 days of the DDoS Attack.

DDoS Mitigation response SLA

If, during a DDoS Attack on a Subscriber with DDoS Protection and Mitigation Service, there is a material delay in response time and the cause of the delay is within Fastly's control, a one-time credit of $500 per incident will be credited to that Subscriber's account.

SLA credit terms

- Requests for Invoice Credits must be made within 30 days of the DDoS Attack that triggered the service credit.
- All requests for Invoice Credits must be made to billing@fastly.com (mailto:billing@fastly.com).
- In no event shall Invoice Credits exceed the fee for the DDoS Protection and Mitigation Service payable by a Subscriber for the month in which the Invoice Credits accrued.
- A pending Invoice Credit does not release a Subscriber from the Subscriber's obligation to pay Fastly's submitted invoices in full when due.
- Invoice Credits will be applied to the invoice within the month the credits were incurred.

Termination for SLA

For a Subscriber of the DDoS Protection and Mitigation Service with a Termed Contract, if in any three-month period where three (3) or more support response time objectives are not met and the failure to meet the objectives materially adversely impacted the Subscriber, the Subscriber will have 30 days to terminate the DDoS Protection and Mitigation Service subscription following the third response failure. Subscribers must notify Fastly of their intention to terminate the DDoS Protection and Mitigation Service subscription within 30 days of the triggering event.

---

§ Legacy Premium Support and SLA (/guides/customer-support/legacy-premium-support-and-sla)

NOTE: Fastly maintains support for its original Premium Support plan. For more information about our current Gold and Platinum support plans (/guides/customer-support/support-description-and-sla) or for information about our Professional Services packages (/guides/customer-support/professional-services), contact sales@fastly.com (mailto:sales@fastly.com).

Legacy Premium Support description and SLA

Support availability and response times vary depending on the type of account you have and the level of support you have purchased. The following table summarizes those offerings:

Offering	Unpaid Account	Month-to-Month Account	Termed Contact	Premium Support
Online Forums	Yes	Yes	Yes	Yes
Email Support Response Time Commitment	Best Effort	Best Effort	Next Business Day	Severity 1 Incidents: 15 minutes*. All Others: Next Business Day
Severe Incident Response Email Address	No	No	No	Yes
Support SLA	None	None	Termination Option	Invoice Credits + Termination Option

Technical support

The following section applies to all Subscribers.

Definitions

- **"Business Hours"** are 8AM-6PM Monday through Friday, Pacific Time.
- **"Business Days"** are Monday through Friday excluding US and UK national and banking holidays.
- An **"Incident"** is an occurrence during which an end user's use of Subscriber's services is adversely impacted.
- A **"Severity 1 Incident"** is an Incident resulting in a major service outage requiring Subscriber to redirect all traffic from Fastly to another CDN.
- A **"Severity 2 Incident"** is an Incident resulting in minor or intermittent outage not requiring Subscriber to redirect traffic to another CDN.
- **"Fastly Control"** means elements entirely under Fastly's control and not a consequence of (a) Subscriber hardware or software failures, (b) Subscriber or end user connectivity issues, (c) Subscriber operator errors, (d) Subscriber traffic amounts that exceed Subscriber's Permitted Utilization as defined in the Terms and Conditions, (e) corrupted Subscriber content, (f) acts of god (any) or war, or earthquakes, or terrorist actions.

Subscriber responsibilities

Subscriber is responsible for using and configuring services according to the Documentation available at https://docs.fastly.com (/).

Support requests

Subscribers submit support requests by sending email to support@fastly.com (mailto:support@fastly.com), or by selecting the Control Panel "SUPPORT" tab. Subscribers receive a system-generated response within minutes containing the ticket number and a direct link to the ticket.

Incident reports should include at the least the following:

- Services not responding to end user requests.
- Services incorrectly sending end users error condition messages.
- Services sending incorrect or partial content to end-users.

Incident reports should include all relevant information, such as:

- Subscriber's determination of the Severity Level of the Incident,
- Subscriber hardware failures,
- Subscriber operator errors,
- Services configuration errors made by Subscriber employees,
- Potential Excess Utilization (as defined in the Terms of Use or master services agreement),
- Corrupted Subscriber content,
- DDOS attacks, and
- Relevant *force majeure* acts such as extreme weather, earthquakes, strikes or terrorist actions.

Communications

Communications between Fastly support engineers and Subscriber staff are conducted using the ticketing application, which maintains a time-stamped transcript of all communications, and sends emails to Subscriber and Fastly staff as tickets are updated.

Response time

Fastly shall use best efforts to respond in a timely fashion.

Termed contracts

The following applies to any Subscriber that has a contract with a term and a minimum commitment.

Response times

Fastly commits to acknowledging receipt of a support ticket within the next business day following submission of a support request.

Termination

In any three-month period where three (3) or more support Response Time objectives are not met and the failure to meet the objectives materially adversely impacted Subscriber, Subscriber shall have thirty (30) days to terminate their subscription agreement following the third failure.

Premium Support

The following applies to Subscribers who have purchased Premium Support.

Incident reporting

Severity 1 Incidents: Fastly will provide Subscriber an Incident Support Email address for Subscriber to report Incidents. Subscriber should report Incidents promptly using the Incident Support email.

Severity 2 Incidents: Subscriber should report Severity 2 Incidents by submitting a Support Request.

Response times

Fastly will respond to the report of an Incident by troubleshooting the cause(s) of the Incident and resolve them if caused by factors within Fastly's control, or provide information to those who can resolve the factors if the factors are within others' control, as follows:

For a Severity 1 Incident:

- Fastly support staff will acknowledge receipt of the email within 15 minutes.
- Fastly will start actively troubleshooting within 30 minutes of receipt of the email.
- Fastly will perform its tasks on a 24/7 basis.
- Fastly and Subscriber will immediately communicate upon learning new information that may be useful in troubleshooting the Incident, and status updates between Fastly and Subscriber staff will take place no less frequently than every 30 minutes for the first two hours, and no less frequently than every hour thereafter.
- Fastly staff will work until (a) the Incident is resolved or (b) the Incident is believed to be outside of Fastly's control.

For a Severity 2 Incident:

- During Business Hours, Fastly support staff will acknowledge receipt of the email within two hours or within two hours of the start of the next business day if the Incident does not come in during a Business Day.
- Fastly engineers will begin actively troubleshooting within one business day, will work on the Incident during Business Hours, and will provide status updates to Subscriber daily on each subsequent Business Day.

Charges for Incident Response

For Severity 1 Incidents caused by factors within Subscriber's control, a flat fee of $1500 will be assessed, and any time spent beyond three hours will be invoiced at Subscriber's undiscounted Professional Services rates.

For Severity 2 Incidents caused by factors within Subscriber's control, Subscriber will be invoiced at Subscriber's undiscounted Professional Services Rates.

For all Incidents:

- If the Incident-causing factors are within Fastly's control, there will be no hourly charges for Fastly engineering staff time.
- If the factors are within Subscriber's control, Subscriber agrees to pay Fastly its hourly charges for Fastly engineering staff time. If it appears likely the factors are within Subscriber's, Subscriber may tell Fastly staff to stop working on troubleshooting the Incident (thereby stopping the hourly charges from being incurred). Subscriber agrees to tell Fastly to stop working on an Incident via an email sent to Fastly's Incident Support email address. The timestamp on the email will be the time charges cease to be incurred.

Support Invoice Credits

In the event a Severity 1 Incident occurs, Subscriber has purchased Premium Support, the cause of the Incident is within Fastly's control, and any of the communication or response timeframes are materially not met, a one-time credit of $500 per Incident will be credited to Subscriber's account.

Credit Terms:

- Requests for Invoice Credits must be made within 30 days of the Incident which triggered the service credit.
- In no event shall Invoice Credits exceed the invoice value of the month in which they are accrued.
- A pending credit does not release Subscriber from its obligation to pay Fastly's submitted invoices in full when due.
- Credits will be applied to the invoice two months following the month an invoice credit was incurred.

Legacy Service availability SLA

Support availability and response times vary depending on the type of account (/guides/account-types-and-billing/accounts-and-pricing-plans) you have and the level of support (/guides/customer-support/support-description-and-sla) you have purchased.

Agreement Type	Unpaid Account	Month-to-Month Account	Termed Contract	Premium Support
Service Level Agreement	None	None	Termination Option	Invoice Credits + Termination Option

Definitions

"Degraded Performance" for the Services means the Services are experiencing Error Conditions that are (1) caused by issues under Fastly Control, (2) observable or reproducible by Subscriber or Fastly, (3) requiring Subscriber to redirect traffic off the Services. Degraded Performance does not include any reduction on availability of the Application User Interface or API due to planned maintenance.

"Error Condition" means the Services are (1) not responding to end user requests, (2) incorrectly sending end users error condition messages or (3) sending incorrect partial content to end users and these conditions are observable or reproducible by Subscriber or Fastly.

"Fastly Control" means elements entirely under Fastly's control and not a consequence of (a) Subscriber hardware or software failures, (b) Subscriber or end user connectivity issues, (c) Subscriber operator errors, (d) Subscriber traffic amounts that exceed Subscriber's Permitted Utilization, (e) corrupted Subscriber content, (f) acts of god (any) or war, or earthquakes, or terrorist actions.

Termination

Any Subscriber that has a contract with a term and a minimum commitment shall have thirty (30) days to terminate their subscription agreement if the Services experience Degraded Performance (a) for longer than 7.2 hours in any one month, or (b) for longer than 43.8 minutes each month in any three contiguous months. Subscriber shall have thirty (30) days to terminate their contract following the third failure.

Availability of invoice credits

Subscribers who purchase Premium Support shall be entitled to Invoice Credits according to the following table.

Availability Percent	Period of Degraded Performance	Monthly Credit Percent
Below 100% - 99.99%	Up to 4.32 minutes	1%
99.99% – 99.9%	Up to 43.8 minutes	5%
99.89% – 99.0%	Up to 7.2 hours	10%
98.99% - 98.0%	Up to 14.4 hours	25%
Below 97.99%	Greater than 864 minutes	50%

Invoice Credits for unavailability will accrue on a monthly basis. The Credit Amount for a month is equal to the monthly usage charge multiplied by Monthly Credit Percent.

Credit terms

- Requests for Invoice Credits for Degraded Performance must be made within 30 days of the period of Degraded Performance.
- The maximum amount of any credit is the Invoice Amount for the month the Degraded Performance occurred.
- A pending credit does not release Subscriber from its obligation to pay Fastly's submitted invoices in full when due.
- Credits will be applied to the Invoice two months following the month an invoice credit was incurred.

---

§ Product and feature life cycles (/guides/customer-support/product-and-feature-life-cycles)

Fastly releases or retires its products and features as detailed below.

Product and feature releases

We release our products and features in the following stages.

Beta

Beta products are initial releases of potential future products or features. We provide customers who participate in our Beta program the opportunity to test, validate, and provide feedback on future functionality. Feedback gathered during this phase helps us to determine which features and functionality provide the most value to our customers and helps us focus our efforts accordingly.

These guidelines apply to Fastly's Beta program:

- Customers can choose or elect to participate in a Beta program.
- Fastly does not make any promises on the features, functionality, or performance of our Beta products.
- We reserve the right to change the scope of or discontinue a Beta product or feature at any point in time.
- We do not charge our subscribers for using our Beta products or features.
- Beta products or features are not included in any existing support contracts or obligations.
- Fastly does not provide Beta customers with discounts on future purchases of any products or services.
- Fastly strongly advises against using production traffic for Beta products due to their dynamic nature.

Beta services are covered by Section 7 of our Terms of Service (https://www.fastly.com/terms).

Limited Availability

Limited Availability products are ready to be released to the world, pending some fine tuning. Limited Availability allows us to test out a product or service with a limited number of customers, so we can closely monitor it and make any necessary adjustments before rolling it out more broadly. Our goal is to make it easy for customers to set up our products with their services and take advantage of the features that come along with them.

These guidelines apply to Fastly's Limited Availability program:

- Fastly may charge its Limited Availability customers and pricing may vary depending on features.
- Fastly does not make any promises on the features, functionality, or performance of our Limited Availability products.
- Fastly does not provide its Limited Availability customers with discounts on future purchases of any products or services.
- Fastly does provide limited product and customer engineering support and documentation for Limited Availability products.

General Availability

General Availability products released by Fastly are available for everyone's use. Fastly manages these products in accordance with Fastly's terms and conditions (https://www.fastly.com/terms).

Product or feature retirement

The decision to retire or deprecate Limited Availability or General Availability features always follows a rigorous process including understanding the demand, use, impact of feature retirement and, most importantly, customers' feedback. Our goal is to always invest resources in areas that will add the most value for customers. When low-value functionality or less successful features compete for resources or create confusion, we may decide that retirement or deprecation is the best solution. In the most difficult of scenarios, feature retirement may cause temporary challenges for some customers. Focusing on the highest priorities of the greatest number of customers, however, allows us to continue to deliver a superior solution with the most benefit.

Fastly is committed to transparency in everything we do, particularly when that activity has implications on the functionality of our features or platform. In the interest of building trust and clarifying change, we have established a number of guidelines around communication of feature retirement, end-of-life, and deprecation.

When a decision to retire a feature is reached Fastly will strive to provide:

- **Advance notice:** We will provide notification proportional to feature criticality. For minor changes with improved functionality we will notify customers no less than three (3) months prior to deprecation, and for major changes we will notify no less than six (6) months in advance.
- **Alternative functionality:** We will include guidance and direction on new features in our services which replace retired or deprecated functionality. New features and functionality will always be provided in advance to ensure customers have time to understand and transition to new functionality prior to the retirement of previous functionality. In some cases this may be with partners or other approved third-party services.
- **Continuous support:** Fastly commits to providing continuous support for all features until the retirement date.
- **Considerate scheduling:** When planning significant changes, including feature or product retirement, we will align retirement as close to major updates or releases as possible to limit the scope of impact on your services.

In some extreme cases Fastly may need to accelerate the retirement of functionality timeline:

- Essential changes that are necessary or appropriate to protect the integrity of our service may occasionally be required. In these cases, it is important that those changes occur as quickly as possible. We will communicate with customers transparently with as much advance notice as possible in these situations.
- Integrated third-party software or services (/guides/third-party-technology/) may need to be retired due to the third-party decision to change or retire their solution. In these situations, the pace of the retirement will be out of our control, although we remain committed to transparency and will strive to provide as much notice as possible.

For more information, contact customer support (mailto:support@fastly.com) or your account team.

§ Professional Services (/guides/customer-support/professional-services)

Fastly offers a range of Professional Services to help you begin using Fastly services. Choose between Implementation Services, Management Services, or Consulting Engagement Services, depending on your needs. For more information about any of our Professional Services packages, contact sales@fastly.com (mailto:sales@fastly.com).

Implementation Services

How it works

Fastly Professional Services staff will personally guide you through the following stages:

- **Planning:** Professional Services staff help you with requirements gathering, solution design, documentation and resource allocation.
- **Implementation:** Professional Services staff help you with configuration of Fastly services and custom VCL development. They provide best-practice consulting for configuration of your origins.
- **Testing:** Professional Services staff help you validate configurations and set up testing.
- **Go-Live:** Professional Services staff monitor and address issues during final production testing and deployment.

Implementation, Testing, and Go-Live may involve some iterative cycles depending on the complexity of your configuration.

Implementation options

Some common implementation options we offer include:

- Initial setup and configuration
- End-to-end encryption setup
- Fine-tuning cache times
- Custom header logic
- Dynamic content delivery optimization
- Multi-tiered caching setup
- Lightweight web page hosting
- Custom purging and event-driven content management
- Geographic or localization detection
- Edge logic and device detection
- Stale content configuration and origin outage handling
- Edge authentication and authorization
- ESI (edge side includes)
- Streaming and video packaging
- Site performance analysis
- Managed vendor migration

Fastly offers two Service Implementation packages:

- **Standard Implementation -** Basic implementation for Fastly customers who have simple content configurations.
- **Enterprise Implementation -** Advanced implementation for Fastly customers who have complex, custom configurations.

Management Services

For customers who require ongoing configuration and technical assistance, Fastly offers Management Services that provide professional services to you and your staff on an as-needed basis. These hours may be used to supplement your existing Support Plan or Implementation Services.

Some common activities you may need assistance with:

- Site performance analysis
- Varnish and VCL training
- Service configuration
- End-to-end encryption setup

- Cache time fine-tuning
- Custom header logic creation
- Dynamic content delivery optimization
- Multi-tiered caching setup
- Lightweight web page hosting
- Custom purging and event-driven content management
- Geographic or localization detection
- Edge logic and device detection
- Stale content configuration and origin outage handling
- Edge authentication
- ESI (edge side includes) configuration
- Streaming and video packaging

Consulting Engagement Services

For customers who lack in-house expertise or dedicated resources, Fastly's Support Engineers are available to provide a range of more technical professional services, including:

- Technical advisory services
- Translating configurations to VCL
- Optimization of website performance
- On-site Varnish and VCL training
- Non-Fastly related performance tuning
- Adapting Fastly features to a particular customer use case

§ Service availability SLA (/guides/customer-support/service-availability-sla)

Support availability and response times vary depending on the type of account (/guides/account-types-and-billing/accounts-and-pricing-plans) you have and the level of support (/guides/customer-support/support-description-and-sla) you have purchased.

Agreement Type	Unpaid Account	Month-to-Month Account	Termed Contract	Gold & Platinum Support
Service Level Agreement	None	None	Termination Option	Invoice Credits + Termination Option

Definitions

"Degraded Performance" means the Services are experiencing Error Conditions that are (1) caused by issues under Fastly Control, (2) observable or reproducible by Subscriber or Fastly, (3) requiring Subscriber to redirect traffic off the Services. Degraded Performance does not include any reduction on availability of the Application User Interface or API due to maintenance.

"Error Condition" means the Services are (1) not responding to end user requests, (2) incorrectly sending end users error condition messages or (3) sending incorrect partial content to end users and these conditions are observable or reproducible by Subscriber or Fastly.

"Fastly Control" means elements entirely under Fastly's control and not a consequence of (a) Subscriber hardware or software failures, (b) Subscriber or end user connectivity issues, (c) Subscriber operator errors, (d) a Utilization spike (see below), (e) corrupted Subscriber content, (f) acts of god (any) or war, or earthquakes, or terrorist actions.

Termination

Any Subscriber that has a contract with a term and a minimum commitment shall have thirty (30) days to terminate their subscription agreement following (1) a period of Degraded Performance longer than 7.2 hours in any one month, or (b) three contiguous months that have periods of Degraded performance longer than 43.8 minutes each.

Availability invoice credits

Subscribers who purchase Gold or Platinum Support shall be entitled to Invoice Credits according to the following table.

Availability Percent	Period of Degraded Performance	Monthly Credit Percent

Below 100% - 99.99%	Up to 4.32 minutes	1%
99.99% – 99.9%	Up to 43.8 minutes	5%
99.89% – 99.0%	Up to 7.2 hours	10%
98.99% - 98.0%	Up to 14.4 hours	25%
Below 97.99%	Greater than 864 minutes	50%

Invoice Credits for unavailability will accrue on a monthly basis. The Credit Amount for a month is equal to the monthly usage charge multiplied by Monthly Credit Percent.

Credit terms

- Requests for Invoice Credits for Degraded Performance must be made within 30 days of the period of Degraded Performance.
- The maximum amount of any credit is the Invoice Amount for the month the Degraded Performance occurred.
- A pending credit does not release Subscriber from its obligation to pay Fastly's submitted invoices in full when due.
- Credits will be applied to the Invoice two months following the month an invoice credit was incurred.

Utilization Spikes

Subscriber's bandwidth utilization, measured in megabits per second, will be sampled every five (5) minutes on a region-by-region basis each month (the "**Samples**"). Subscriber's "**Average Utilization**" for a region in a month will be the average of the Samples. Subscriber's "**Peak Utilization**" for a region in a month will be calculated by the 95th percentile method, according to which the Samples will then be ordered from highest to lowest, and the highest five percent (5%) of Samples will be discarded and the remaining highest Sample will be Subscriber's Peak Utilization for the region in that month. Subscriber's "**Permitted Utilization**" in a month for a region will be five (5) times Subscriber's Average Utilization in that month for that region. A "**Utilization Spike**" will occur if Subscriber's Peak Utilization exceeds its Permitted Utilization in a region. Utilization Spikes may interfere with or disrupt the integrity or performance of the Services. Subscribers should contact Support in advance of any planned utilization spike and respond immediately to any communications from Fastly regarding an actual or suspected Utilization Spike.

§ Submitting support requests (/guides/customer-support/submitting-support-requests)

Fastly offers standard technical support for all accounts through its support@fastly.com (mailto:support@fastly.com) email address. However, support availability and response times vary depending on the type of account you have and the level of support you have purchased. Our support description and SLA (/guides/customer-support/support-description-and-sla) information discusses these details further.

For all levels of technical support, Fastly's customer support ticketing system automatically generates and assigns each request a unique service ticket number, which is then sent within minutes to the customer at the email address used when submitting the ticket. This automated response also contains a direct link to the service ticket within Fastly's customer support system. Fastly then responds to these requests as appropriate for the level of support purchased by the customer.

Accounts with Gold or Platinum Support

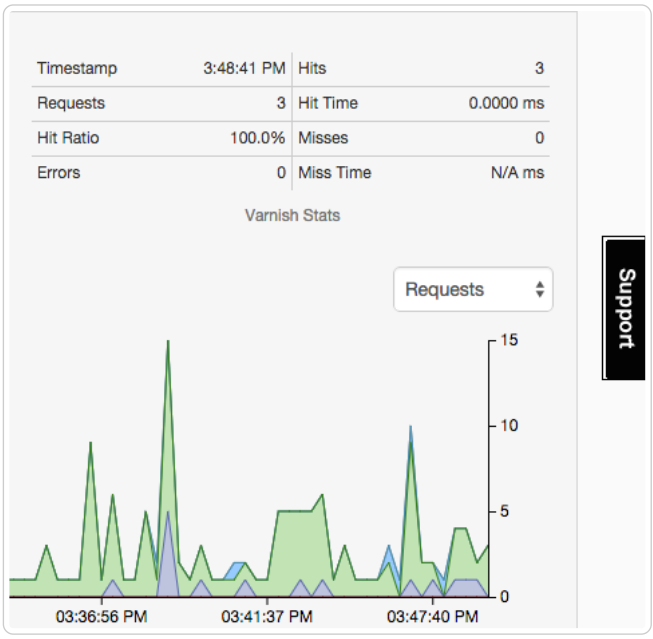
To submit emergency support tickets, customers must pay for a Gold or Platinum Support plan that includes support for severe incidents. Customers with Gold or Platinum Support plans can submit requests for support any time:

- a service that has been working stops working.
- a production deployment experiences a drastic change in performance that either is customer- or end-user-affecting.

Customers with Gold or Platinum Support accounts submit emergency tickets to an emergency support email address (instead of via the Support tab on the Fastly application interface or via the standard Customer Support email address (mailto:support@fastly.com)). Customers with Platinum Support accounts can also submit tickets via a toll-free telephone number.

Accounts without Gold or Platinum Support

Customers who do not pay for Gold or Platinum Support (/guides/customer-support/support-description-and-sla) levels can submit support requests either via email to support@fastly.com (mailto:support@fastly.com) or directly in the Fastly application interface, via the Support tab at the right side of every page in the interface, any time they require assistance with self-provisioned tasks (/guides/about-fastly-services/self-provisioned-fastly-services). Fastly then responds to these requests as appropriate for the level of support purchased by the customer.



§ Support description and SLA (/guides/customer-support/support-description-and-sla)

Support availability and response times vary depending on the type of account you have and the level of support you have purchased. The following table summarizes those offerings:

Support Offering	**Standard Support**	**Gold Support**	**Platinum Support**
Online Self-Service Help	Unlimited access.	Unlimited access.	Unlimited access.
Availability for General Inquiries	Business hours.	Business hours.	24/7/365.
Availability for Incident Reports	Business hours, including weekends & holidays.	24/7/365.	24/7/365.
Initial Response Times	By the next business day.	Severity 1 Incidents within 2 hours. Severity 2 Incidents within same day. All other Incidents by the next business day.	Severity 1 Incidents within 15 minutes. Severity 2 Incidents within 2 hours. All other Incidents by the next business day.
Web and email support	Available.	Available, with priority over Standard Support.	Available, with priority over Standard and Gold Support.
Phone and chat support	Not available.	Not available.	Toll-free telephone available 24/7/365. Dedicated chat channel available during Fastly business hours.
Emergency Escalation	Not available.	Not available.	Available via email and phone.
Designated Customer Support Engineer	Not available.	Not available.	Available for large accounts on case-by-case basis.
Termination Option	Not available for unpaid and month-to-month customers. Only included for termed contracts.	Available with invoice credits.	Available with invoice credits.

Technical support

The following section applies to all subscribers.

Definitions

- **"Business Hours"** are 8AM-6PM during a Business Day in California, New York, London, or Tokyo.
- **"Business Days"** are Monday through Friday, excluding any day that is simultaneously a US, UK, and Japanese national or banking holiday.
- An **"Incident"** is an occurrence during which end users' use of Subscriber's services is adversely impacted.
- A **"Severity 1 Incident"** is an incident resulting in a major service outage requiring Subscriber to redirect all traffic from Fastly to another CDN.
- A **"Severity 2 Incident"** is an incident resulting in minor or intermittent outage not requiring Subscriber to redirect traffic to another CDN.
- **"Fastly Control"** means elements entirely under Fastly's control and not a consequence of (a) a Subscriber's hardware or software failures, (b) a Subscriber's or end user's connectivity issues, (c) Subscriber operator errors, (d) Subscriber traffic amounts that exceed a Subscriber's Permitted Utilization as defined in the Terms and Conditions, (e) corrupted Subscriber content, (f) acts of god (any) or war, or earthquakes, or terrorist actions.

Subscriber responsibilities

Subscriber is responsible using and configuring services according to the Documentation available at https://docs.fastly.com (/).

Support requests

Subscribers submit support requests by sending email to support@fastly.com (mailto:support@fastly.com), or by selecting the Control Panel "SUPPORT" tab. Subscribers receive a system-generated response within minutes containing the ticket number and a direct link to the ticket.

Incident reports should include at the least the following:

- Services are not responding to end user requests.
- Services incorrectly send end users error condition messages.
- Services send incorrect or partial content to end users.

Incident reports should include all relevant information such as:

- Subscriber's determination of the Severity Level of the incident,
- Subscriber hardware failures,
- Subscriber operator errors,
- Services configuration errors made by Subscriber employees,
- A potential Utilization Spike (see the Service Availability SLA (/guides/customer-support/service-availability-sla)),
- Corrupted Subscriber content, DDOS attacks, and Relevant force majeure acts such as extreme weather, earthquakes, strikes or terrorist actions.

Communications

Tickets

Communications between Fastly support engineers and Subscriber personnel are conducted using the ticketing application, which maintains a time-stamped transcript of communications, and sends emails to Subscriber and Fastly staff as tickets are updated.

Chat

Subscribers to Platinum Support receive a dedicated chat channel for real-time communications during Business Hours. Though subject to change, Fastly's current chat provider is Slack (www.slack.com (https://slack.com/)).

Phone support

Subscribers to Platinum Support receive a dedicated phone number to contact Fastly support engineers. Fastly personnel can also establish audio and video conferencing (free app or browser plug-in required) for real-time voice and video communications.

Response time

Fastly shall use best efforts to respond in a timely fashion.

Termed contracts

The following applies to any subscriber that has a contract with a term and a minimum commitment.

Response times

Fastly commits to acknowledging receipt of a support ticket within the next Business Day following submission of a support request by a Subscriber with a Termed Contract.

Termination

In any three-month period where three (3) or more support Response Time objectives are not met and the failure to meet the objectives materially adversely impacted Subscriber, Subscribers with a Termed Contract, Gold Support or Platinum Support shall have thirty (30) days to terminate their subscription agreement following the third failure.

Incident response times

Incident reporting

Severity 1 Incidents: Fastly will provide Subscriber an Incident Support Email address for Subscriber to report Incidents. Subscriber should report Incidents promptly using the Incident Support email.

Severity 2 Incidents: Subscriber should report Severity 2 Incidents by submitting a Support Request.

Platinum Support

Fastly will respond to the report of an Incident by troubleshooting the cause(s) of the Incident and resolve them if caused by factors within Fastly's control, or provide information to those who can resolve the factors if the factors are within others' control, as follows:

For a Severity 1 Incident:

- Fastly support staff will acknowledge receipt of the email within 15 minutes.
- Fastly will start actively troubleshooting within 30 minutes of receipt of the email.
- Fastly will perform its tasks on a 24/7 basis.
- Fastly and Subscriber will immediately communicate upon learning new information that may be useful in troubleshooting the incident, and status updates between Fastly and Subscriber staff will take place no less frequently than every 30 minutes for the first two hours, and no less frequently than every hour thereafter.
- Fastly staff will work until (a) the incident is resolved or (b) the incident is believed to be outside of Fastly's control.

For a Severity 2 Incident:

- Fastly support staff will acknowledge receipt of the email within two hours.
- Fastly engineers will begin actively troubleshooting within the same day, will work on the Incident during the same day, and will provide status updates to Subscriber daily on each subsequent day.

Gold Support

Fastly will respond to the report of an Incident by troubleshooting the causes of the Incident and resolve them if caused by factors within Fastly's control, or provide information to those who can resolve the factors if the factors are within others' control, as follows:

For a Severity 1 Incident:

- Fastly support staff will acknowledge receipt of the email within two hours.
- Fastly engineers will begin actively troubleshooting within the same day, will work on the Incident during the same day, and will provide status updates to Subscriber daily on each subsequent day.
- Fastly staff will work until (a) the incident is resolved or (b) the incident is believed to be outside of Fastly's control.

For a Severity 2 Incident:

- Fastly support staff will acknowledge receipt of the email within the same day.
- Fastly engineers will begin actively troubleshooting within the same day, will work on the Incident during the same day or next day, and will provide status updates to Subscriber daily on each subsequent day.

For Severity 1 Incidents caused by factors within Subscriber's control, a flat fee of $1500 will be assessed, and any time spent beyond three hours will be invoiced at Subscriber's undiscounted Professional Services rates.

For Severity 2 Incidents caused by factors within Subscriber's control, Subscriber will be invoiced at Subscriber's undiscounted Professional Services Rates.

For all incidents:

- If the Incident-causing factors are within Fastly's control, there will be no hourly charges for Fastly engineering staff time.
- If the factors are within Subscriber's control, Subscriber agrees to pay Fastly its hourly charges for Fastly engineering staff time. If it appears

likely the factors are within Subscriber's control, Subscriber may tell Fastly staff to stop working on troubleshooting the Incident (thereby stopping the hourly charges from being incurred). Subscriber agrees to tell Fastly to stop working on an Incident via an email sent to Fastly's Incident Support email address. The timestamp on the email will be the time charges cease to be incurred.

Support invoice credits

In the event a Severity 1 Incident occurs, Subscriber has purchased Gold or Platinum Support, the cause of the Incident is within Fastly's control, and any of the communication or response timeframes are materially not met, a one-time credit of $500 per incident will be credited to Subscriber's account.

Credit Terms:

- Requests for Invoice Credits must be made within 30 days of the incident which triggered the service credit.
- In no event shall Invoice Credits exceed the invoice value of the month in which they are accrued.
- A pending credit does not release Subscriber from its obligation to pay Fastly's submitted invoices in full when due.
- Credits will be applied to the invoice two months following the month an invoice credit was incurred.

NOTE: Fastly maintains support for its original Premium Support plan (/guides/customer-support/legacy-premium-support-and-sla). To convert your account to the current Gold and Platinum Support plans, contact sales@fastly.com (mailto:sales@fastly.com).

• Guides (/guides/) > Compliance and law (/guides/compliance-law) > About our terms (/guides/about-our-terms/)

• Guides (/guides/) > Compliance and law (/guides/compliance-law) > Compliance (/guides/compliance/)

§ HIPAA and caching PHI (/guides/compliance/hipaa-and-caching-phi)

You can configure the Fastly CDN service to cache and transmit protected health information (PHI) in keeping with Health Information Portability and Accountability Act (HIPAA) security requirements. Use the following features to ensure secure handling of cache data that contains PHI:

- Configure frontend (/guides/securing-communications/ordering-a-paid-tls-option) and backend (/guides/securing-communications/connecting-to-origins-over-tls) TLS to encrypt transmitted data from your origin to your end users.
- Add the `beresp.hipaa` variable (/guides/vcl/miscellaneous-VCL-extensions) to objects containing PHI to keep that data out of non-volatile disk storage at the edge.

Contact sales@fastly.com (mailto:sales@fastly.com) for more information on how to enable the `beresp.hipaa` feature for your account. For accounts that have this feature enabled, Fastly will enter into a HIPAA business associate agreement (BAA) as an addendum to our terms of service (https://www.fastly.com/terms).

NOTE: Fastly's security and technology compliance program includes safeguards for the entire Fastly CDN service, independent of using the `beresp.hipaa` variable. The Fastly security program (/guides/compliance/security-program) and technology compliance (/guides/compliance/technology-compliance) guides provide more information about these safeguards.

---

§ PCI-compliant caching (/guides/compliance/pci-compliant-caching)

We have designed Fastly's core CDN service with Payment Card Industry Data Security Standard (PCI DSS) compliance in mind. With proper authorization on your account, you can use Fastly's `beresp.pci` VCL variable (/guides/vcl/miscellaneous-VCL-extensions) to automatically cache content in a manner that satisfies PCI DSS requirements.

Adding the `beresp.pci` variable to an object prevents writing of that object to non-volatile disk storage on the edge. Combined with frontend (/guides/securing-communications/ordering-a-paid-tls-option) and backend TLS (/guides/securing-communications/connecting-to-origins-over-tls), this feature allows you to cache and transmit flagged content through the Fastly network in compliance with our PCI certification.

Contact sales-ecommerce@fastly.com (mailto:sales-ecommerce@fastly.com) for more information on how to enable this feature for your account.

NOTE: Fastly's security and technology compliance program includes safeguards for the entire Fastly CDN Service, independent of using the `beresp.pci` variable. The Fastly security program (/guides/compliance/security-program) and technology compliance (/guides/compliance/technology-compliance) guides provide more information about these safeguards.

§ Security program (/guides/compliance/security-program)

Fastly's security program includes safeguards that help protect your data as it moves through the Fastly service. Information about these safeguards is organized by category. Our technology compliance (/guides/compliance/technology-compliance) guide describes additional safeguards we maintain.

Authentication and authorization

User account assignment. We assign individual user accounts to personnel who access Fastly systems and devices. These assignments help us monitor and enforce accountability of user activity.

User-level privileges. Our systems and devices enforce user roles or similar measures to control the extent of access we grant individual users.

Multi-factor authentication. We enforce multi-factor authentication to better secure our computing resources from unauthorized logins.

Application security

Secure software development. We provide annual training to Fastly developers to help identify and prevent common software vulnerabilities, including the OWASP Top 10 (https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet). Developer code undergoes peer review prior to deployment, and internal security engineers and third-party security validators periodically analyze code for software components with higher potential security risk.

Web application security review. A third party assesses the security of the Fastly web application annually. We address findings from this assessment according to the risk they pose to the security of the Fastly service.

Network and infrastructure security

Network security reviews. We regularly perform vulnerability scans and third-party penetration tests on the Fastly network. We review and address findings from these activities to help maintain the security of our network.

Configuration standards. We document and follow configuration standards to maintain secure systems and network devices. These standards include business justification for used ports, protocols, and services, as well as the removal of insecure default settings.

Vulnerability and patch management. To maintain awareness of potential security vulnerabilities, Fastly monitors public and private distribution lists, as well as reports submitted through our responsible disclosure (https://www.fastly.com/security/report-security-issue) process. We validate and implement security patches for critical vulnerabilities within 24 hours of discovery. For non-critical vulnerabilities and updates, we schedule and deploy vendor-provided patches on a regular basis.

Encryption

Secure data transmission. The Fastly service supports TLS configurations (/guides/securing-communications/) to encrypt connections both externally to end users and backend origin servers, as well as internally within the Fastly network.

Encryption key management. We maintain technology and procedures to secure private keys throughout their lifecycle.

Key storage and access security. We store private keys in encrypted repositories, and we restrict key storage access to personnel who support our key management processes.

Data center and physical security

Physical access restrictions. Our data centers are fully enclosed with perimeter protection such as fences, gates, and mantraps to prevent unwanted entry. Only authorized people (including data center personnel, our employees, and contractors) may enter and move within a data center.

Data center access management. We ensure movement within our data centers is monitored via onsite safeguards such as security guard assignment, facility access logging and review, and video surveillance. Additionally, we periodically review and adjust the list of personnel who may enter our data centers.

Secure asset installation. We install computer and network hardware in locked cages and racks. Only authorized individuals may physically access this equipment.

Environmental safeguards. Our data centers compensate for environmental disruptions with systems that control backup power, temperature and humidity, and fire suppression.

Business continuity and operational resilience

Service failover. If any of our points of presence (POPs (/guides/about-fastly-services/fastly-pop-locations)) experience issues serving content, we can redirect traffic to a neighboring POP without interrupting the delivery of content to end users.

Internet redundancy. Our data centers have connections with multiple Internet service providers. We do not rely on any single carrier to serve content to end users.

Service monitoring. We monitor multiple internal and external reporting channels to detect service-related issues. Personnel are available 24x7x365 to confirm and respond to disruptions of the Fastly service.

Communication and reporting. We update impacted customers using various communication methods (such as status.fastly.com (https://status.fastly.com/)), depending on an incident's scope and severity.

Security incident management

Incident response plan. We maintain a formal incident response plan with established roles and responsibilities, communication protocols, and response procedures. We review and update this plan periodically to adapt it to evolving threats and risks to the Fastly service.

Incident response team. Representatives from key departments help address security-related incidents we discover. These personnel coordinate the investigation and resolution of incidents, as well as communication with external contacts as needed.

Breach notification. Fastly will notify affected customers within 48 hours of validating an unauthorized disclosure of customer confidential information.

Logging and monitoring

Log analysis. We aggregate and securely store Fastly internal system activity. Monitoring these logs helps us discover and investigate potential security issues.

Change and configuration monitoring. We use multiple monitoring and alert mechanisms to enhance the visibility of technology changes and help ensure adherence to our change management process.

Intrusion detection. We maintain mechanisms to detect potential intrusions at the network and host level. Our Security department inspects and responds to events these detection measures discover.

Customer and end user data management

Cache data and configurations. Customers manage which content is cached, where, and for how long by setting policies that control that content. See our introduction to caching (/guides/about-fastly-services/how-fastlys-cdn-service-works) for more information. We may directly access or modify customer accounts or configurations to provide our services, prevent or address service or technical issues, as required by law, or as customers expressly permit. For the same reasons, we may also access or modify equipment, systems, or services that manage customer content.

Client IP addresses. As part of our caching network’s general interaction with the Internet, Fastly independently collects anonymized and aggregated client IP address information on a limited basis to provide and improve its services. Client IP addresses are retained in a non-anonymized, non-aggregated fashion for up to two business days, or up to seven days if those addresses are associated with transmission errors (such as 503 "Service Unavailable" errors (/guides/debugging/common-503-errors)), and are discarded thereafter.

Subscriber IP addresses. Fastly independently collects the IP addresses of users who access their services within the Fastly application or through the API. We make these IP addresses available to customers through our audit log functionality (https://www.fastly.com/blog/announcing-audit-logs). If customers define origin servers or syslog endpoints with IP addresses, we save those IP addresses as part of their configurations. We may retain IP addresses from audit logs or configurations indefinitely. Dynamically-resolved origin IP addresses may be retained for up to two business days, or up to seven days if those addresses are associated with transmission errors (such as 503 "Service Unavailable" errors (/guides/debugging/common-503-errors)), and are discarded thereafter.

IP addresses and security monitoring. Fastly may retain indefinitely any non-anonymized, non-aggregated client or subscriber IP addresses associated with suspicious activity that may pose a risk to the Fastly network or our customers, or that are associated with administrative connections to the Fastly service.

Content request data. Content enters, transits, and departs our network in response to requests. We retain and use data about the operation and reliability of our processing of requests to monitor, maintain, and improve our services, our business operations, and our security and compliance programs. Subject to confidentiality obligations to our customers, we only disclose this data in anonymized and aggregated form.

Subscriber log streaming. Subscribers may stream syslog activity (/guides/streaming-logs/), including end user IP addresses (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses#common-sources-of-new-content), to a remote endpoint for analysis and use. Fastly does not retain subscriber syslog activity, except as described above.

---

§ Technology compliance (/guides/compliance/technology-compliance)

Fastly's technology compliance program includes safeguards that help protect your data as it moves through the Fastly service. Information about these safeguards is organized by category. Our security program (/guides/compliance/security-program) guide describes additional safeguards we maintain.

Governance

Information security roles and responsibilities. We have formally assigned information security duties to Fastly personnel. Our Chief Security Officer and Security organization work with other departments to safeguard sensitive information related to the Fastly service.

Policies and procedures. Our policies and procedures help us maintain security in our systems, processes, and employee practices. Fastly's Security organization formally reviews these policies and procedures at least annually.

Risk management. We integrate risk assessment activities with various processes to identify and address information security risk to the company and customer data on our network.

Vendor security oversight. Fastly performs risk-based evaluations of the security measures of our vendors. We review these security measures before we begin using a vendor, and we ask the vendor to formally acknowledge these measures. We re-evaluate vendor security measures on a recurring basis thereafter.

Human resources security

Employee background screening. We screen new employees as part of the hiring process. Screening activities depend on applicable local regulations and may include criminal background checks and reference checks.

Confidentiality agreement. Our employees formally agree to safeguard the sensitive information they may view, process, or transmit as part of their job functions.

Security awareness training. We train our people to protect the data and devices they use. Each employee receives security awareness training as part of new hire procedures, and current employees take this training annually.

Data privacy

Privacy policy. Our privacy policy (https://www.fastly.com/privacy) describes how we collect, use, and protect the personal information of customer personnel using our websites or configuring services in the Fastly application. We certify our privacy policy and practices with TRUSTe (https://privacy.truste.com/privacy-seal/Fastly/validation?rid=e864daae-e54e-4aba-9ae0-6a76f78e0962).

Personal data transfer. The Fastly services by default do not process personal data. However, our service can be configured or used at the direction of the customer to process personal data. Our guide about our terms (/guides/about-our-terms/#can-i-use-transfer-data-from-the-eu-to-the-u-s-using-the-fastly-services) provides additional information about data privacy compliance related to the processing of personal data.

Technology change management

Change management process. We follow a defined set of procedures to develop and deploy technology changes. These changes include updates to software, configurations, and devices that support the Fastly service.

Testing. We test technology changes at various stages of development, and we confirm those tests are successful before completing a deployment into the Fastly service.

Change approval and notification. As part of our deployment process, we prepare, approve, and communicate change notices to maintain awareness among personnel who manage the Fastly network and systems.

Post-implementation review. We confirm the success of changes after their deployment. Should we experience issues during implementation, we also maintain procedures to revert changes.

Identity and access management

User requests and approval. We document and approve requests for user access to the Fastly network. Our security administrators confirm appropriate documentation is in place before granting requested user rights.

Access modification. We promptly update or remove an employee's access to the Fastly network to match that employee's current job function or employment status.

User access review. We periodically inspect access privileges to make sure our personnel have appropriate access to Fastly systems and data.

• Guides (/guides/) > Compliance and law (/guides/compliance-law) > Security measures (/guides/security-measures/)

• Guides (/guides/) > Compliance and law (/guides/compliance-law) > Third-party technology (/guides/third-party-technology/)

• Guides (/guides/) > Compliance and law (/guides/compliance-law) > Related offerings (/guides/related-offerings/)

• Guides (/guides/) > Compliance and law (/guides/compliance-law) > Translations (/guides/translations/)

• Guides (/guides/) > Compliance and law (/guides/compliance-law) > Archives (/guides/archives/)

• Guides (/guides/) > Basic setup (/guides/basic) > Basic setup (/guides/basic-setup/)

§ Adding CNAME records (/guides/basic-setup/adding-cname-records)

To use Fastly with your website or application, you will need to create a CNAME DNS record for your domain name (/guides/about-fastly-services/domain-names-and-fastly-services). Fastly provides many different hostnames for CNAME records. The URL you select will depend on whether or not you use a TLS certificate, and whether or not you want to cache resources globally or only in the US and Europe.

Setting the CNAME record

Consult your DNS provider's instructions to configure the CNAME record for your domain name. The steps to add a CNAME record will vary for each registrar's control panel interface.

If you cannot find your provider's CNAME configuration instructions, Google maintains instructions for most major providers (https://support.google.com/a/topic/1615038?hl=en). Keep in mind that these instructions are maintained by Google, not Fastly, and are tailored specifically for Google enterprise services.

If you have trouble configuring your CNAME records, contact us at support@fastly.com (mailto:support@fastly.com).

IMPORTANT: Make sure that your domain name is added to a service in the Fastly web interface, otherwise users may see an "unknown domain" error.

Using a wildcard CNAME record

If you're planning on using many third-level domain names (e.g., api.example.com and blog.example.com), you may want to create a single wildcard CNAME DNS record for all of them, as opposed to separate DNS records for each of them. To do this, you would create a CNAME record using `*` as the name and one of the hostnames listed below as the value of the record. This would point all of your third-level domain names for the second-level domain name (e.g., example.com) at Fastly.

After you've added the wildcard CNAME DNS record, you'll need to add each third-level domain name as a domain in the Fastly application. You will see an "unknown domain" error if the domain name has not been added to a service. You can direct traffic from third-level domain names at different origin servers by using conditions. For example, if you want all traffic to api.example.com to be routed to Origin Server A, create a condition (/guides/conditions/using-conditions) for Origin Server A with `req.http.host ~ "api.example.com"` in the **Apply If** field.

CNAME hostnames for non-TLS traffic

If you're not using a TLS certificate, use one of the following hostnames for your CNAME record:

- `global-nossl.fastly.net` - We recommend all customers use this hostname by default unless told otherwise by Fastly support personnel. TLS is disabled, and connections to port 443 are refused.
- `global.prod.fastly.net` - Global cache services are enabled. TLS certificates are not supported. Connections are accepted on port 443, but a warning is generated when used with a domain.
- `a.prod.fastly.net` - Same as above, but excludes all cache servers except those in the US and Europe.

CNAME hostnames for TLS traffic

If you're using a TLS certificate, use one of the following hostnames for your CNAME record:

- `[single letter].global-ssl.fastly.net` - Maps to customer wildcard TLS certificates. The letter you use will depend on the certificate to which your domain was added. Fastly support will provide you with additional information.

  IMPORTANT: Do not prepend your domain to the hostname. For example, do not create a CNAME record for `example.com.[single letter].global-ssl.fastly.net`. You should only create a CNAME record for the TLS hostname provided by Fastly support.

- `[single letter].ssl.fastly.net` - Same as above, but excludes all cache servers except those in the US and Europe.

For more information about using TLS certificates with Fastly, see our guide on ordering a paid TLS option (/guides/securing-communications/ordering-a-paid-tls-option). We also have a shared TLS wildcard certificate (/guides/securing-communications/setting-up-free-tls) that you can use for free.

CNAME hostname for custom DNS maps

For customers with custom DNS maps, Fastly provides `*.map.fastly.net`. You may receive your own map hostname if you upload a TLS certificate to Fastly, or if you have custom traffic management policies.

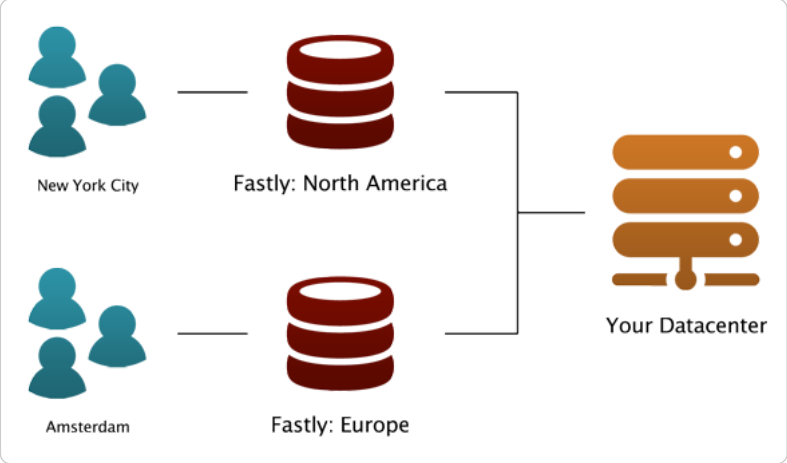
---

§ Getting started with Fastly (/guides/basic-setup/getting-started-with-fastly)

In this article, we explain what Fastly does and how best to use it with your site.

How Fastly works

Fastly works by storing the content of your website on servers all over the world and quickly delivering that content to your users. We do this using Varnish (https://www.varnish-cache.org), an open source web application accelerator.



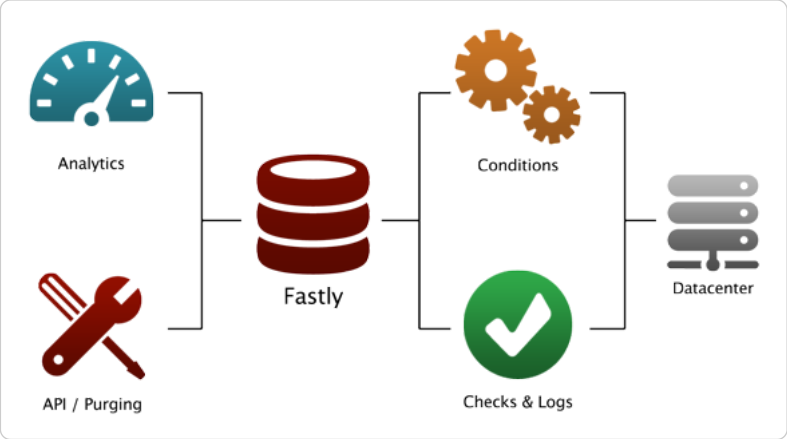
We track the geo-location of each user and make sure they are connecting to a server that is closest to them. This makes your site faster by reducing the time spent waiting for data to be sent from the server to the user.

We also give you full control over when and how we store content from your servers. You can set a Time To Live (TTL) for any path on your site and instantly invalidate or purge any path on your site using our Purge API (/api/purge).

By using these tools, you only have to generate pages one time for the site for many millions of page views. This saves time for your users and costs on your server bills.

Advanced features

Fastly also provides many advanced features that help you monitor how your data is accessed and customize your content delivery.



- **Instant Purging (/guides/purging/)** allows you to have better control over when and how content is updated. You can update your data when you want and as often as you want, rather than waiting up to 24 hours to change data at the edge.
- **Real Time Analytics** provides a top level view of your network and how your site is performing. Every second, we compile the relevant data about all of your traffic into an easy-to-read report.
- **Conditions (/guides/conditions/)** change how requests are routed, what headers to send, and how content is cached.
- **Health Checks (/guides/basic-configuration/health-checks-tutorial)** monitors the status of one or many of your back end servers. This way if anything goes wrong with your servers, you immediately know about it.
- **Streaming Logs (/guides/streaming-logs/setting-up-remote-log-streaming)** are quickly and easily configured to send information from your servers anywhere and in the format you want.
- **Varnish Configuration Language (VCL) (/guides/vcl/guide-to-vcl)** allows you to modify nearly every aspect of an HTTP request and response. You can upload VCL files with specialized configurations to your account.
- The **Fastly API (/api/)** can programmatically handle your configuration. This allows you to write scripts to handle basic configuration tasks and create your own administrative views (so they can be directly coupled with your existing admin software).

Getting started

If you do not have an account, sign up now (https://www.fastly.com/signup). Feel free to choose the developer plan so you can test how Fastly works on your site.

If you want to use Fastly but do not know where to begin, check the Basic setup (/guides/basic-setup/) documentation. You can learn everything you need to set up and configure your first service for your site.

If you want to explore more, check the Basic configuration documentation to learn more about caching, and features such as purging (/guides/purging/) and shielding (/guides/performance-tuning/shielding).

If you want information on advanced features, especially related to things like load balancing (/guides/performance-tuning/load-balancing-configuration) or the Varnish Control Language (/guides/vcl/) that we support, check the advanced configuration section of our help files or the API Reference (/api/), which includes a full reference to the Fastly API.

If you are having problems, send us a message at support@fastly.com (mailto:support@fastly.com).

§ Glossary of terms (/guides/basic-setup/glossary-of-terms)

These are common Fastly, HTTP, and networking terms you may encounter within our service guides.

ACL

Access Control List. A list of permissions that can be attached to an object (/guides/vcl/using-access-control-lists) allowing customers to quickly check a client's IP against a list of known net blocks and then make decisions based on the result.

Altitude

Fastly's customer summit (https://www.fastly.com/altitude).

app.fastly.com

The user interface through which customers access Fastly's CDN services (https://app.fastly.com/).

Backend

See *origin server*.

Cache-Control

The specific HTTP header (/guides/tutorials/cache-control-tutorial) that controls who can cache a response, under which conditions, and for how long. Fastly respects `Cache-Control` headers returned from origin servers as one approach to cache management. See also *surrogate-control*, *max-age*, and *s-maxage*.

community.fastly.com

Fastly's community discussion forum (https://community.fastly.com/).

Cookie

HTTP headers used to perform certain functions like authenticating login in secure website areas, information tracking, remembering user preferences, and customizing how information is presented.

cURL

An open-source command line tool (https://curl.haxx.se/) for transferring data with URL syntax from or to a server using one of many supported protocols. Fastly users can issue cURL commands to verify requests are caching (/guides/debugging/curl-and-other-caching-verification-methods) in the Fastly network.

DNS

Domain Name System. A system for naming computers and network services that translates a domain's numbered IP address into an easy-to-remember alphabetic name.

Edge Dictionary

A type of container Fastly users can create (/guides/edge-dictionaries/) to store data as key-value pairs and turn frequently repeated statements into a single function that acts as constant.

Egress traffic

Bandwidth used when traffic travels from Fastly points of presence (POPs) to the end user.

ESI

Edge Side Includes. An XML-based markup language (http://www.w3.org/TR/esi-lang) that allows content assembly by HTTP surrogates. Allows Fastly users to cache pages that contain both cacheable and uncacheable content (such as user-specific information).

Gzip

A way of compressing information to make it faster to transmit. Fastly allows users to dynamically gzip content (/guides/basic-configuration/enabling-automatic-gzipping) based on file extension or content type.

Header

An HTTP field that precedes the main content of information being sent in a request or response and describes the length of the content, type of content, or other characteristics of the information.

Host (header)

Information used in addition to the IP address and port number to uniquely identify a domain. See also *Normalizing the host header (https://www.fastly.com/blog/varnish-tip-normalize-host-header)*.

Ingress traffic

Bandwidth used when end users make requests that send traffic to Fastly points of presence (POPs).

Instant Purge

A feature of Fastly's purging functionality (/guides/purging/) that allows users to actively invalidate content in Fastly caches within milliseconds. See also *Soft Purge*.

max-age

An HTTP Cache-Control directive that specifies how long (in seconds) an object will remain in the cache before Fastly removes the object from storage. See also *surrogate-control*, *cache-control*, and *s-maxage*.

MTR

A tool that combines traceroute and ping programs in a single network diagnostic tool. Frequently used in debugging network connections (/guides/debugging/debugging-with-mtr).

Origin server

The location or address from which Fastly's network requests the content it will serve.

Origin Shield (Shielding)

A specific Fastly point of presence (POP) designated by users (/guides/performance-tuning/shielding#enabling-shielding) as the primary source of content through which all content requests from other POPs will be directed in lieu of contacting a customer's origins directly.

OTFP

On-the-fly packaging. A feature of Fastly's Video on Demand (https://www.fastly.com/services/video-demand) media and streaming services that allows customers to dynamically package video for delivery in multiple HTTP streaming formats. Also known as "just in time" video content packaging.

POP

Point of Presence. Datacenter within which Fastly's globally distributed cache servers (https://www.fastly.com/network-map) reside.

private

An HTTP Cache-Control directive that allows users to select which objects are not cached. Fastly will not cache responses with a Cache-Control value of `private`.

Purging

The process of picking out one or more objects from the Fastly cache and discarding it along with its variants. See also *Instant Purge* and *Soft Purge*.

Redirect

A function that directs requests for information from their originally intended locations to a more desirable destination (/guides/performance-tuning/generating-http-redirects-at-the-edge).

Set-Cookie

The header sent by a server in response to an HTTP request and then used to create a cookie on a user's origin. Fastly supports a method for extracting a named value (/guides/vcl/response-cookie-handling) out of `Set-Cookie` headers no matter how many there are. By default, Fastly will not cache responses that contain a `Set-Cookie` header.

s-maxage

An HTTP cache control directive similar to *max-age*, but applied only to shared caches. See also *surrogate-control* and *cache-control*.

Soft Purge

A type of purging (/guides/purging/) that allows users to easily mark content as outdated (/guides/purging/soft-purges) (expired) instead of immediately deleting it from Fastly's caches. See also *Instant Purge*.

status.fastly.com

Fastly's network status (https://status.fastly.com/) monitoring site. Allows customers to quickly check whether anomalies they see may be due to a known problem currently being worked on by Fastly or if their issues more likely stem from problems within their own infrastructure.

support@fastly.com

The main email address of Fastly's Customer Support (https://www.fastly.com/support) team through which customers can ask questions and receive assistance.

Surrogate-Control

An HTTP response header that allows origin servers to use control directives to dictate how intermediate caches, including Fastly, should handle response entities. `Surrogate-Control` will not affect browsers. See also *cache-control*, *max-age*, and *s-maxage*.

Surrogate Key

A unique identifier that allows customers to group content together for faster processing. Fastly uses surrogate keys (https://www.fastly.com/blog/surrogate-keys-part-1) as part of its purging strategy (/guides/purging/).

Synthetic response

Custom responses generated within the CDN that users can set if a specific URL is requested or a specific condition, such as a status code, is met. These responses require no origin server interaction.

TLS (SSL)

A cryptographic protocol Fastly follows (/guides/securing-communications/) that ensures privacy between communicating applications and their users on the internet.

URL

Uniform Resource Locator. An address used (/guides/about-fastly-services/domain-names-and-fastly-services) to find a site or application's objects on the internet.

Varnish

Caching software (https://www.fastly.com/blog/benefits-using-varnish) that helps content-heavy dynamic websites as well as heavily consumed APIs load faster. Fastly's core caching infrastructure is based on a heavily modified version of Varnish.

VCL

Varnish Configuration Language. A scripting language (/guides/vcl/guide-to-vcl) used to configure and add logic to Varnish caches. Fastly users can create custom VCL files with specialized configurations.

www.fastly-debug.com

A network debugging tool designed to provide key info to help a Fastly user troubleshoot issues with Fastly's Customer Support (https://www.fastly.com/support) team.

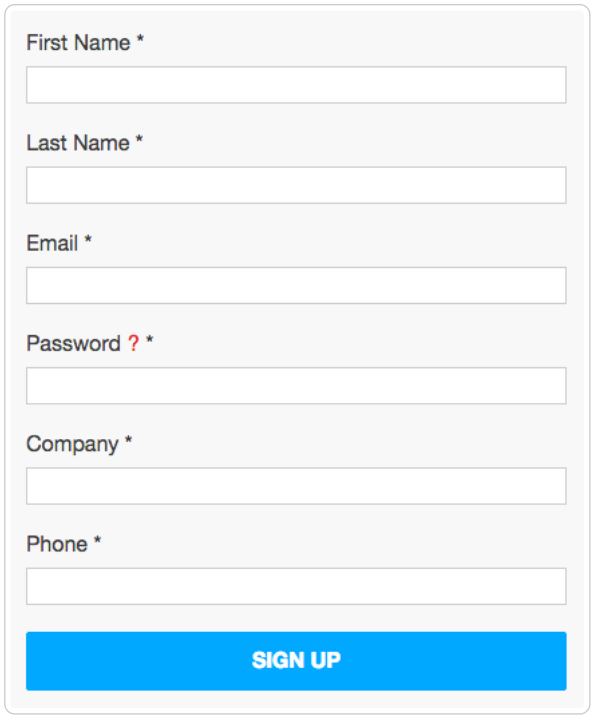
§ Sign up and create your first service (/guides/basic-setup/sign-up-and-create-your-first-service)

To create a Fastly account and set up your first service, follow the steps below.

Sign up at Fastly.com

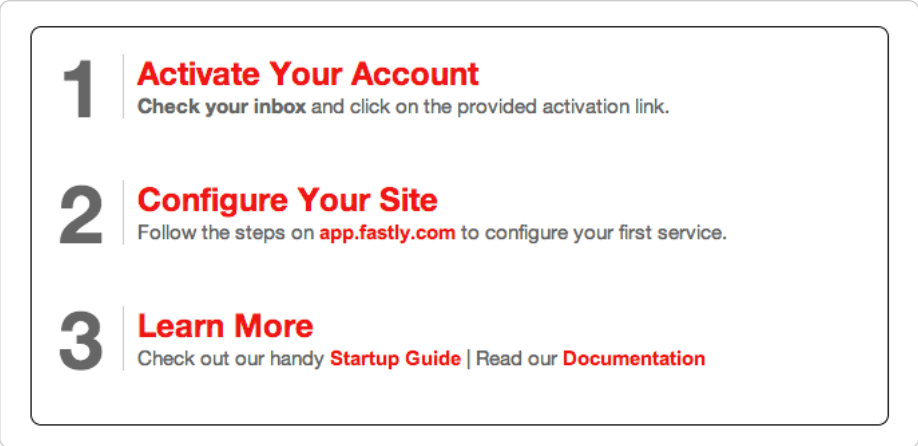
Before you do anything else, you must sign up for a Fastly account.

1. Click on any **Try Fastly Now** button on the Fastly.com website or simply point a browser to the signup form (https://www.fastly.com/signup).
2. When the signup form appears, fill in all the fields with your contact information. All the fields are required.



> **NOTE:** You'll be able to change your password and email address (/guides/user-access-and-control/email-and-password-changes) any time after signup. Without a valid email we can't send you account verification details during account setup. Without a valid telephone number, we can't assist you with specific kinds of account lockout issues (/guides/account-management-and-security/enabling-an-ip-whitelist-for-account-logins).

3. Click the **Sign Up** button. The confirmation screen will appear with instructions on what to do next and you'll be sent an e-mail that contains a verification link.



4. Check your inbox and find the confirmation email we sent you.
5. Click the verification link (we need to make sure you're not a spam robot and verify your email). The verification link will immediately take you to the first step of the quick start process so you can create your first service.

Create your first service and test it

Once you've verified your email, we log you into the application automatically and immediately take you through the quick start process to create your first service.

1. In the **Name Your Service** field, type the name of the first service you'd like to create. This name can be anything at all that will help you distinguish this service from any others you create in the future. You can rename this service (/guides/basic-setup/working-with-services#renaming-services) at any time.
2. In the **Server Address** field, type the IP address (or hostname) and port number (usually 80) for your website's server. We need this to make sure your cache updates properly. If you have many servers you can configure the rest later.
3. In the **Domain Name** field, type the domain name of your website. We need this information to properly route requests to your website.
4. Click the **Configure** button to create your first service. The system configures your service and displays a link for you to test.
5. Click the configuration test link that appears. Your website should appear, but it may take up to 60 seconds as all the systems start talking to each other.

CNAME your domain

To complete the process of creating your first service, you must set the CNAME DNS record for your domain to point to `global-nossl.fastly.net` in order to direct traffic to us. Each service provider does this a little differently, but we've provided a link to instructions for most major providers (/guides/basic-setup/cname-instructions-for-most-providers). Once you've completed this step, you should be all ready to go!

If you are having any problems, feel free to contact us at support@fastly.com (mailto:support@fastly.com).

§ Working with services (/guides/basic-setup/working-with-services)

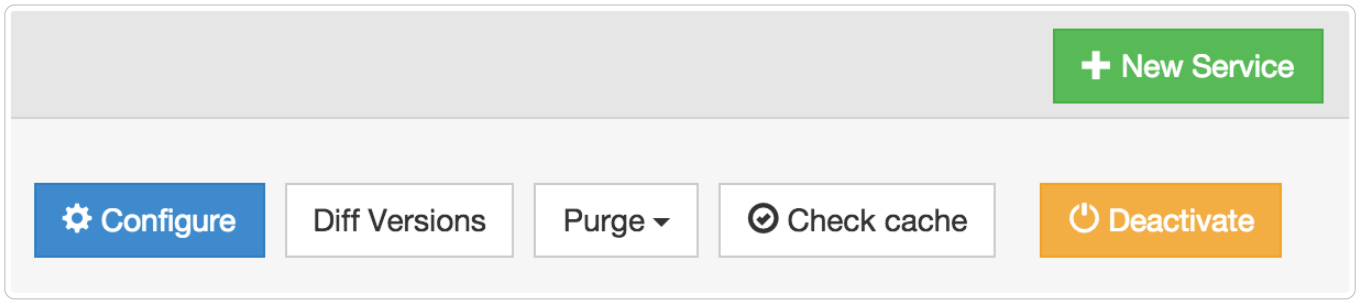
The Fastly application allows you to create new services or edit existing ones and then activate your changes once you have things configured. The application also allows you to do other things with existing services, like rename them, compare them to each other, deactivate and reactivate them, and delete them.

Creating a new service

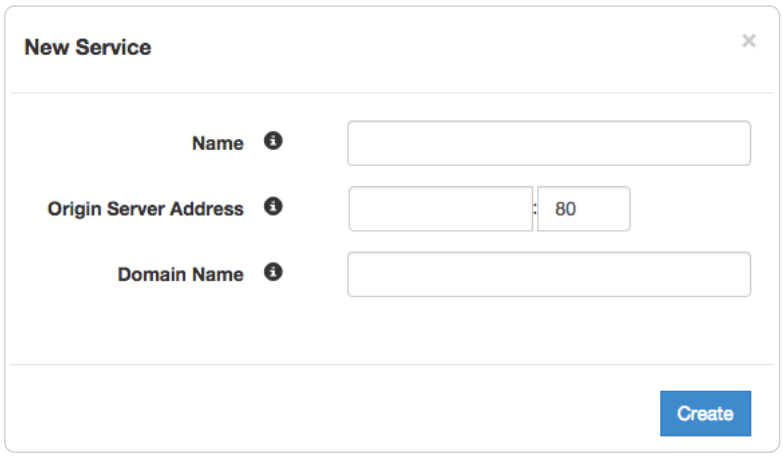
To create a new service, follow the steps below:

1. Log in to the Fastly application.
2. Click the **configure** button (the wrench icon at the top of the window).

3. At the top right of the window, click the green **New Service** button.



The New Service window appears.



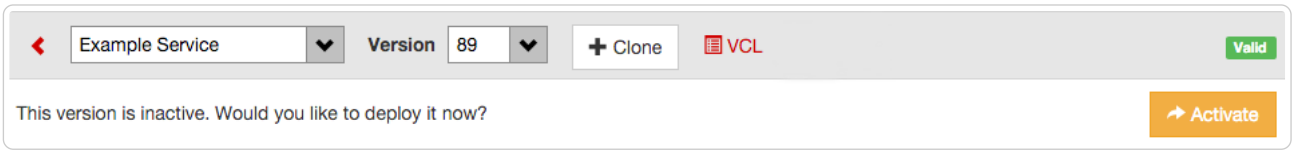
4. Fill out the **New Service** window as follows:
   - In the **Name** field, type the name of the first service you'd like to create. You can rename this service at any time.
   - In the **Origin Server Address** field, type the IP address (or hostname) and port number (usually `80`) for your website's server.
   - In the **Domain Name** field, type the domain name of your website. We need this information to properly route requests to your website.
5. Click the **Create** button. A new service appears in the list of services available.

Once you've created a new service, you still need to activate it so your configuration of that service will take effect.

Service versions and their activation

You must "activate" new services or changes to existing services in order to deploy their configurations. Any time you create a new service, the Fastly application places your new creation in an inactive or "locked" state. The same is true any time you edit an existing service. Configuration changes are never automatically deployed.

To immediately activate a new service or deploy changes to an existing service, click the **Activate** button:

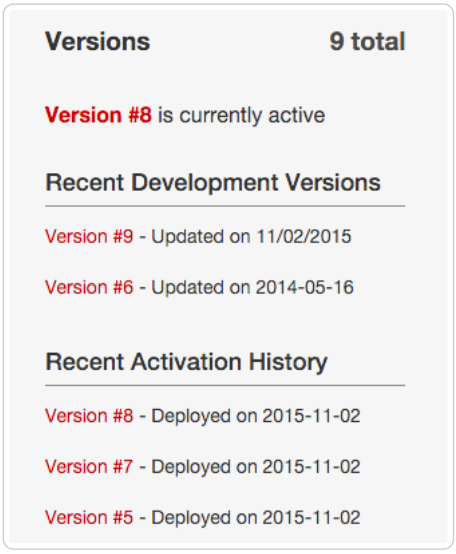


Editing an existing service

NOTE: Fastly locks versions of services you've already deployed to make rollbacks safer and provide version control.

Any time you activate a new version of a service, the system automatically creates a copy of it as a new development version for you to edit later.

For example, if we activate Version 8 of our service, the system will make a copy of it, label it "Version 9" and place it at the top of the list of recent development versions available for editing:



Click the linked development version to begin editing a copy of your existing service.

Alternatively, you can clone any existing service version, active or inactive, and begin editing that cloned version instead. To clone a service version:

1. Log in to the Fastly application and click the **configure** button (the wrench icon at the top of the window).
2. From the **Service** menu, select the appropriate service.
3. From the **Version** menu, select the number of the version you wish to clone.
4. Click the **Clone** button. A new service version appears in the list of recent development versions.

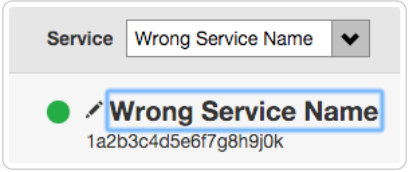
Other things you can do with services

In addition to creating or editing services, we allow you to rename them, compare versions of them, deactivate or reactivate specific version of them, and delete them.

Renaming services

To rename your service at any time, follow the steps below:

1. Log in to the Fastly application and click the **configure** button (the wrench icon at the top of the window).
2. From the **Service** menu, select the appropriate service.
3. Click the service name next to the pencil icon. The service name will become editable.



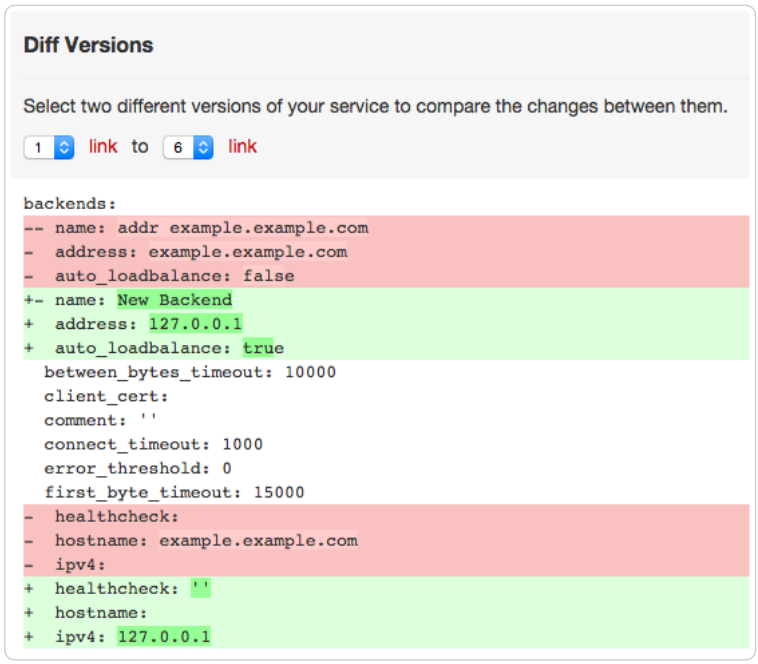
4. Type the new service name and then press return (or Enter) on your keyboard. The new service name appears.

Comparing different service versions

To compare two versions of a service, follow the steps below:

1. Log in to the Fastly application and click the **configure** button (the wrench icon at the top of the window).
2. From the **Service** menu, select the appropriate service.
3. Click the **Diff Versions** button to the right of the service name.

The Diff Versions window appears. Removals between compared versions are highlighted in red with a minus sign at the beginning of the line. Additions are highlighted in green with a plus sign at the beginning of the line.



You can change the compared service versions by selecting a different number in the selection menus.

Deactivating and reactivating services

To reactivate or deactivate a service, follow the steps below:

1. Log in to the Fastly application and click on the **configure** button (the wrench icon at the top of the application window).
2. From the **Service** menu, select the appropriate service.
3. Click the yellow **Activate** or **Deactivate** button to the right of the service name as appropriate.

You can also activate or deactivate a service via the API (/api/config#version). Did you accidentally delete a service? We can help.

Deleting a service

Fastly allows you to delete any service you create, along with all of its versions. Fastly does not offer a way to delete specific versions of a service, however. Service versions are meant to be an historic log of the changes that were made to a service. To undo changes introduced by a particular service version, you can always go back to a previous version and reactivate it or clone a new service version based on any old version.

NOTE: When a service is deleted from your configuration, Fastly maintains a copy in its databases in case the deletion was accidental. Any domains that were associated with that service will remain associated with it even after deletion. Before deleting a service, you should delete the domains within it first and then delete the service itself.

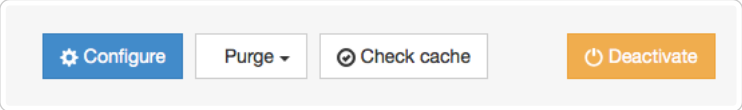
To delete any service along with all of its versions, follow the steps below.

1. Log in to the Fastly application and click on the **configure** button (the wrench icon at the top of the application window).
2. From the **Service** menu, select the appropriate service.
3. Click the blue **Configure** button.
4. Select the latest version of your service. If that version is locked, make a new version of it by clicking the **Clone** button at the top of the screen.

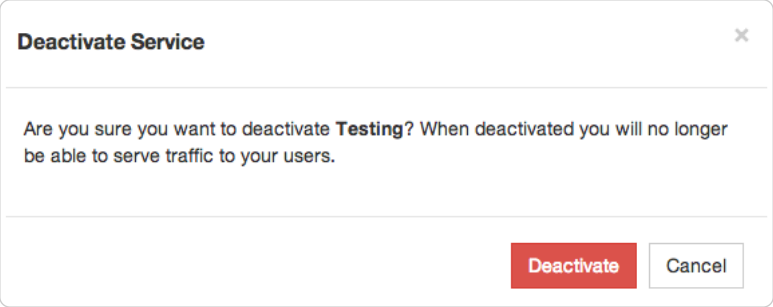
5. In the **Domains** area, click the gear icon to the right of each domain associated with this service and then select **Delete** from the menu that appears (you'll be asked to confirm the deletion).

   > **IMPORTANT:** Be sure to repeat this step until all domains associated with the service have been deleted. If any domains remain associated with a deleted service, the system may not allow you to associate them with a new service at a later date.

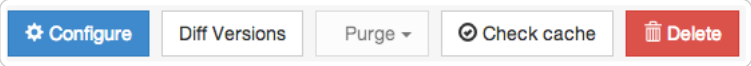
6. Click the **configure** tab at the top of the screen again. The configuration control panel reappears.
7. Click the yellow **Deactivate** button to the right of the service name.

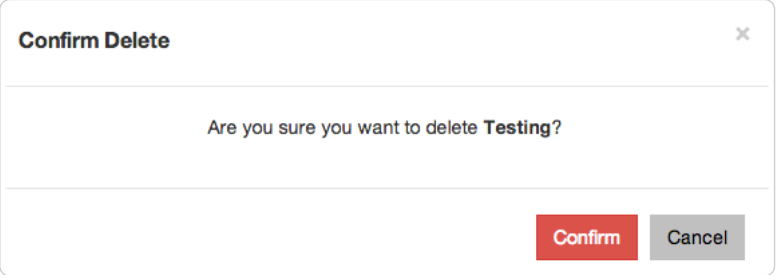
   The **Deactivate Service** warning appears.

8. Click the red **Deactivate** button to confirm you want to deactivate your service and acknowledge that you no longer want to serve traffic with it. The **Delete** button becomes active on the control panel.

9. Click the red **Delete** button to the right of the service name. The **Confirm Delete** window appears.

10. Click the red **Confirm** button to confirm that you want to delete the service.

Getting help with accidental service deletions

Services can be deactivated or deleted. Deactivated services can be reactivated at any time, but once they've been deleted you must contact Customer Support (mailto:support@fastly.com) to have them restored. When sending your request, please remember to include:

- your customer ID (/guides/account-management-and-security/finding-and-managing-your-account-info#finding-your-customer-id)
- your company name
- your service ID (/guides/account-management-and-security/finding-and-managing-your-account-info#finding-your-service-id) (the name of the service you want restored)

Customer Support will notify you when your service has been restored.

• Guides (/guides/) > Basic setup (/guides/basic) > Basic configuration (/guides/basic-configuration/)

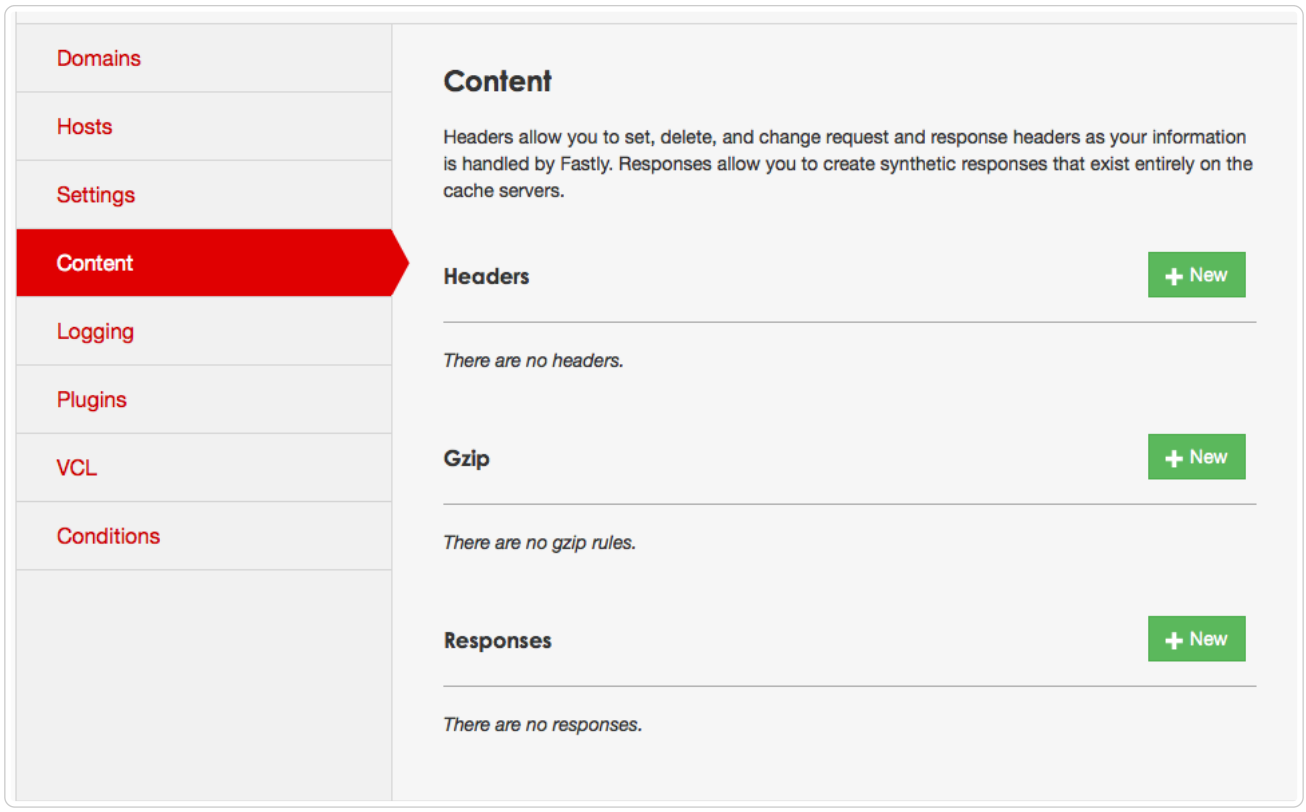
§ Adding or modifying headers on HTTP requests and responses

(/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses)

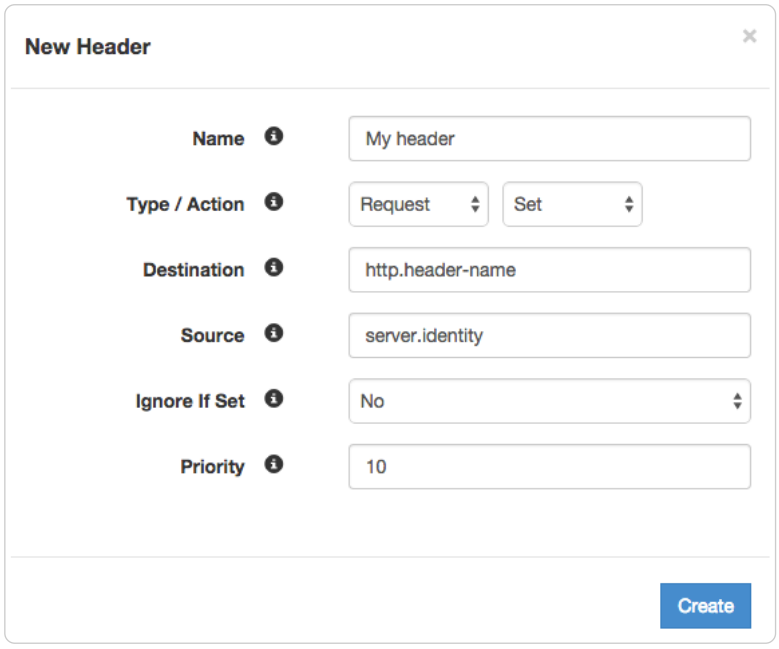
HTTP header fields are components of the header section of request and response messages in the Hypertext Transfer Protocol (HTTP). They define the operating parameters of an HTTP transaction. When you create and configure headers, you can determine how you want your content served to your users. The following steps show you how to add and edit headers.

Create new headers

1. Log in to the Fastly application.
2. Click **configure** (the wrench icon at the top of the window).
3. From the **Services** menu, select the appropriate service.
4. Click the blue **Configure** button.
5. Click the **Content** pane from the list on the left.



6. In the **Headers** area, click **New**. The New Header window appears.

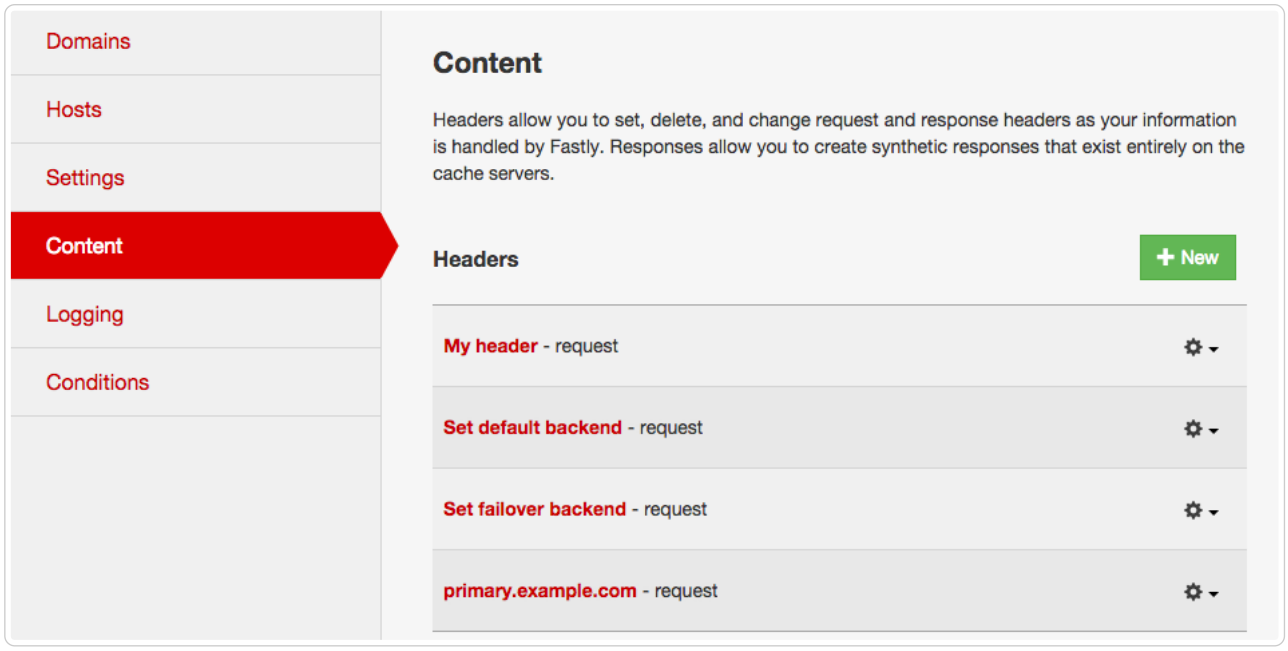


7. Fill out the **New Header** window as follows:
   - In the **Name** field, type the name of your header rule (for example, `My header`).
   - From the **Type/Action** menu, select the type of header that you have and the action that is performed on it.
   - In the **Destination** field, type the name of the header affected by the selected action.
   - In the **Source** field, type where the content for the header comes from.
   - From the **Ignore If Set** menu, select **No** if you want the header in the **Destination** field modified or select **Yes** if you don't want it modified.
   - In the **Priority** field, type the order the header rules execute.

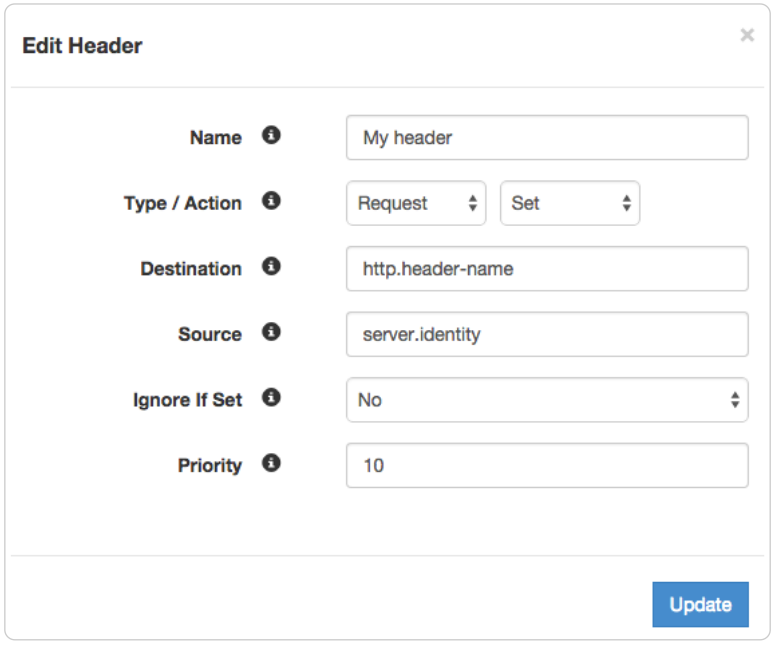
   The Field description table below provides additional details about each of these controls.
8. Click **Create**.

Edit headers

1. Log in to the Fastly application.
2. Click **configure** (the wrench icon at the top of the window).
3. From the **Services** menu, select the appropriate service.
4. Click the blue **Configure** button.
5. Click the **Content** pane from the list on the left.



6. In the **Headers** area, click the gear icon next to the header you want to configure and click **Edit**. The Edit Header window appears.



7. Fill out the **Edit Header** window as follows:
   - In the **Name** field, type the name of your header rule (for example, `My header`).
   - From the **Type/Action** menu, select the type of header that you have and the action that is performed on it.
   - In the **Destination** field, type the name of the header affected by the selected action.
   - In the **Source** field, type where the content for the header comes from.
   - From the **Ignore If Set** menu, select **No** if you want the header in the **Destination** field modified or select **Yes** if you don't want it modified.
   - In the **Priority** field, type the order the header rules execute.
8. Click **Update**.

Field description table

This table describes what each field in the Header window means:

Field	Description
Name	The Name field specifies a handle for you to recognize and remember a particular Header rule. Type whatever memorable word or phrase you would like in this field.

Type	The Type menu can be set to **Request**, **Response**, or **Cache**. Selecting **Request** modifies the request coming from the user, and this will carry through to the request that gets sent to your origin server. Selecting **Response** affects the HTTP response that is sent back to the user. Selecting **Cache** affects the HTTP response that your origin server returns before it gets stored on Fastly servers, meaning whatever changes you make there will be remembered on a cache hit.
Action	The Action menu can be set to **Set**, **Append**, **Delete**, **Regex**, and **Regex All**. Selecting **Set** (the default) will write a value into the header (potentially overwriting it, if it already exists). Selecting **Append** will add a value onto the end of a header or set it if it doesn't exist. Selecting **Delete** will remove a header. When selected, it hides the Source field in the Header window. Selecting **Regex** allows you to perform a find and replace on specific text and is based on a regular expression you type in. When selected, the Regex and Substitution controls appear in the Header window. Selecting **Regex All** allows you to perform the same function as **Regex** but it performs a find and replace multiple times. When selected, the Regex and Substitution controls appear in the Header window.
Destination	The Destination field determines the name of the header that is going to be affected by our Action. Because header rules can be used to affect more than just HTTP headers, your input to this field should be formatted like this: `http.Header-Name`.
Source	The Source field is available on Set, Append, Regex, and Regex All actions. This field becomes hidden in the Header window when you select Delete from the Action menu. It determines where the new content for the header comes from. There are a plethora of options for Source. The simplest is a static string such as `"My Static String"` (including the quotes). Other options include `client.ip`, `req.http.Another-Header`, and `geoip.city`. See the list of Common Sources below for more common sources of new content.
Regex	The Regex field only appears in the Header window when you select Regex or Regex All from the Action menu. It allows you to perform a find and replace on specific text and is based on a regular expression that you type in.
Substitution	The Substitution field only appears in the Header window when you select the Regex and Regex All from the Action menu. It replaces the text that was removed by the regex expression with the text you typed in the Substitution field.
Ignore If Set	By default this is set to No, which means that if the header you are modifying already exists, it will be modified.
Priority	The Priority field determines the order in which the header rules execute (e.g., a priority of 1 means the header rule executes first). This can be important if you set headers and then set other headers based on the earlier ones.

Common sources of new content

Name	Valid Types	Description
`req.http.Fastly-Client-IP`	Request, Cache, Response	The true IP address of the client.
`client.ip` and `client.identity`	Request, Cache, Response	The client IP address. These variables are available, but may not always display the source IP address. For instance, they may show the edge node IP when shielding is enabled. For the true client IP address use req.http.Fastly-Client-IP. **IMPORTANT:** In some cases, client IP data may be considered sensitive. Make sure you protect the sensitive IP data you stream or store.
`server.identity`	Request, Cache, Response	A unique identifier for the Fastly server processing the request.
`server.region`	Request, Cache, Response	The region in which the Fastly server resides.
`server.datacenter`	Request, Cache, Response	The data center in which the Fastly server resides.
`req.url`	Request, Cache, Response	The URL of the HTTP Request from the client.
`req.http.*`	Request, Cache, Response	The headers from the HTTP Request, access as: `req.http.HeaderName`
`beresp.status`	Cache	The status returned from the origin server.
`beresp.http.*`	Cache	The headers from the origin's HTTP Response, access: `beresp.http.HeaderName`

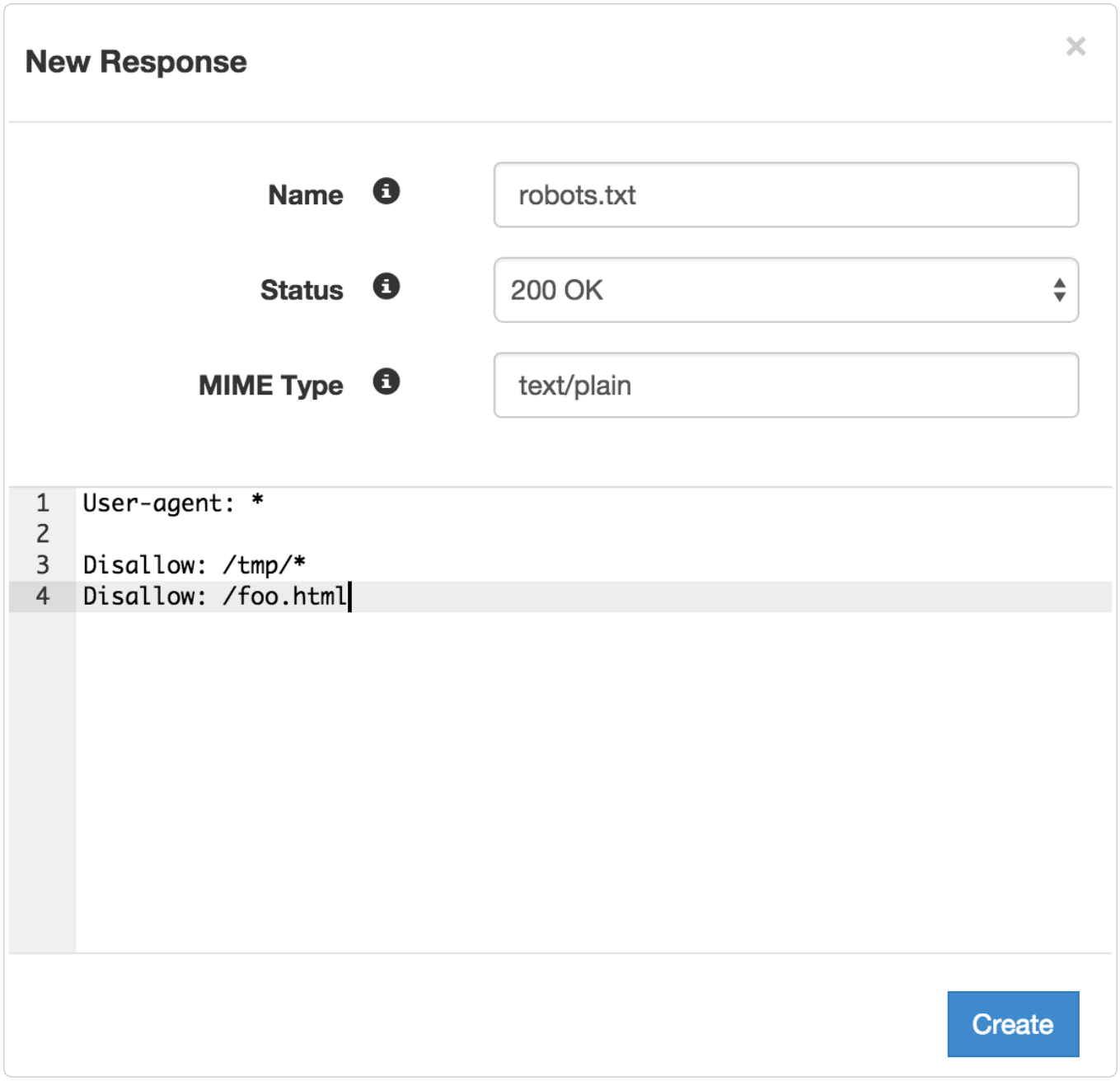
`resp.status`	Response	The status that is going to be returned to the client.
`resp.http.*`	Response	The headers in the HTTP Response to be returned to the client, access: `resp.http.HeaderName`
`geoip.*`	Request, Cache, Response	GeoIP values for the client's IP (see our GeoIP article (/guides/vcl/geoip-related-vcl-features) for more information).

§ Creating and customizing a robots.txt file (/guides/basic-configuration/creating-and-customizing-a-robots-file)

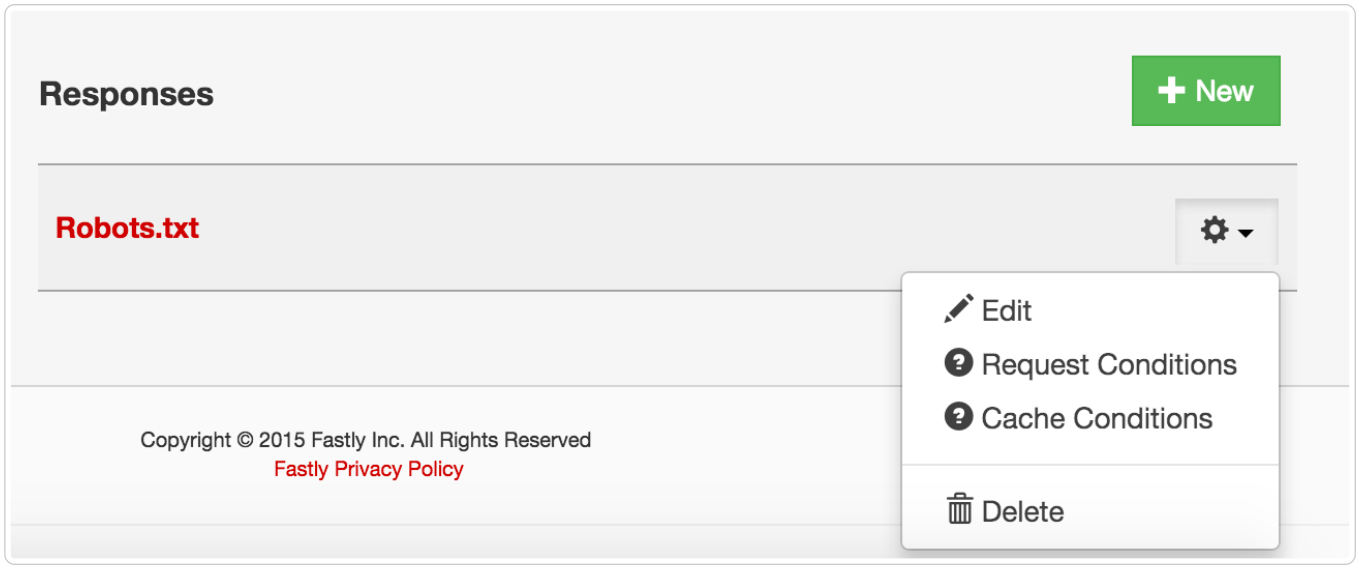
Once you've created a CNAME DNS record (/guides/basic-setup/cname-instructions-for-most-providers) to direct your domain's traffic to Fastly, you can set custom responses (/guides/vcl/custom-responses-that-dont-hit-origin-servers) through the Fastly application that will be served by the robots.txt file (https://en.wikipedia.org/wiki/Robots_exclusion_standard) on your website. To create and configure your robots.txt file via the Fastly application, follow the steps below.

1. Log in to the Fastly application and click the **configure** tab (wrench icon).

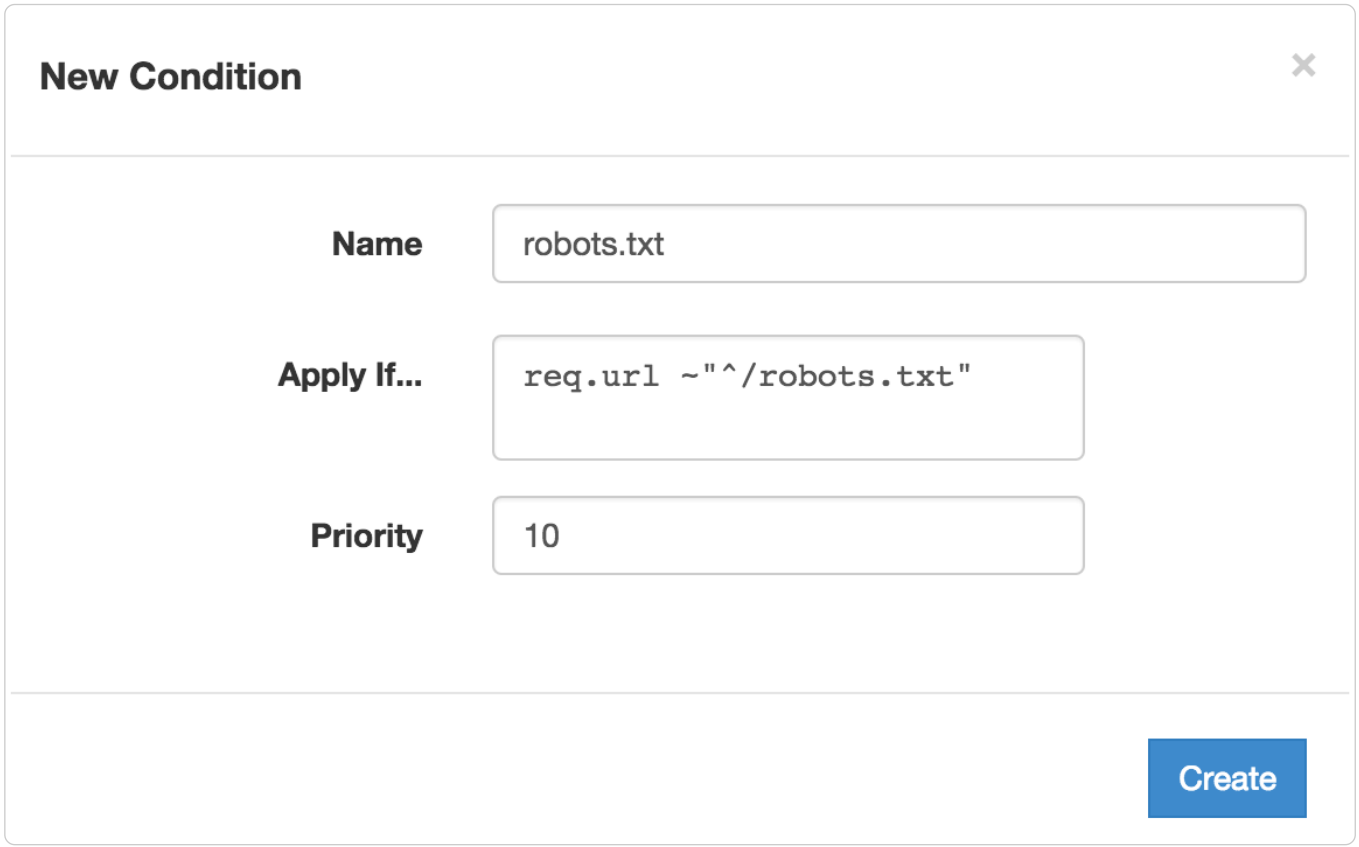
2. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for your selected service appear.
3. Click **Content** from the section list on the left.
4. In the **Responses** area at the bottom of the page click the **New** button. The New Response window appears.



5. Fill out the **New Response** controls as follows:
   - In the **Name** field, type an appropriate name. For example `robots.txt`.
   - Leave the **Status** menu set at its default `200 OK`.
   - In the **MIME Type** field, type `text/plain`.
   - In the text editor area at the bottom of the window, specify at least one User-agent string and one Disallow string. For instance, the above example tells all user agents (via the `User-agent: *` string) they are not allowed to crawl anything after `/tmp/` directory or the `/foo.html` file (via the `Disallow: /tmp/*` and `Disallow: /foo.html` strings respectively).
6. Click the **Create** button. Your new response appears in the list of responses.
7. Click the gear icon to the right of the new response you just created and select **Request Conditions** from the menu.



The New Condition window appears.



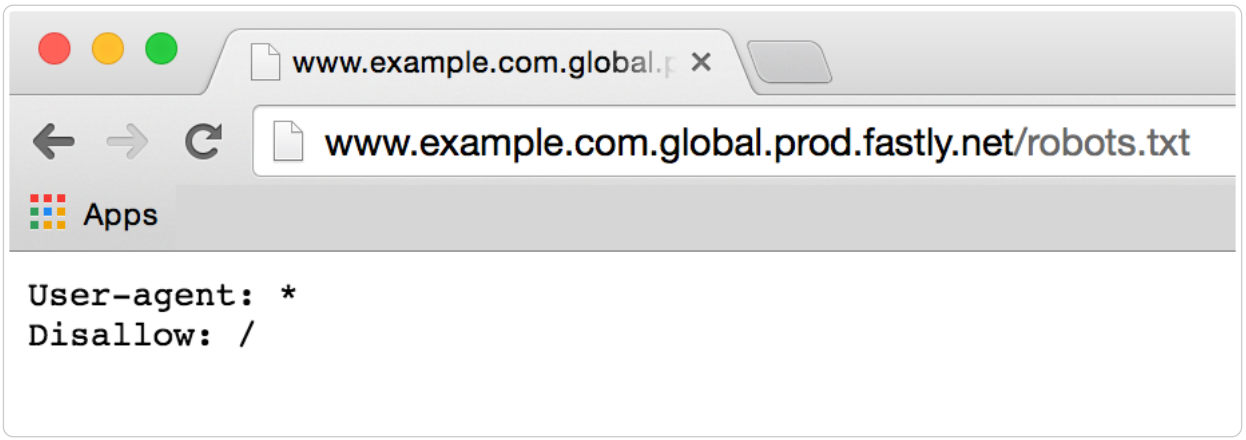
8. Fill out the fields of the **New Condition** window as follows:
   - In the **Name** field type a meaningful name for your condition (e.g., `robots.txt`).
   - In the **Apply If** field, type the logical expression to execute in VCL to determine if the condition resolves as True or False. In this case, the logical expression would be the location of your robots.txt file (e.g., `req.url ~ "^/robots.txt"`).
   - Leave the **Priority** set to 10.
9. Click **Create** to create the new condition.

NOTE: For an in-depth explanation of creating custom responses, check out our Responses Tutorial (/guides/basic-configuration/responses-tutorial).

Why can't I customize my robots.txt file with global.prod.fastly.net?

Adding the `.global.prod.fastly.net` extension to your domain (for example, `www.example.com.global.prod.fastly.net`) via the browser or in a cURL command can be used to test how your production site will perform using Fastly's services.

To prevent Google from accidentally crawling this test URL, we provide an internal robots.txt file that instructs Google's webcrawlers to ignore all pages for all hostnames that end in `.prod.fastly.net`.



This internal robots.txt file cannot be customized via the Fastly UI until after you have set the CNAME DNS record for your domain to point to `global.prod.fastly.net`.

§ Creating error pages with custom responses (/guides/basic-configuration/creating-error-pages-with-custom-responses)

The default error responses served by Fastly can be jarring for your users, especially when using Fastly for consumer applications. This tutorial shows you how to set up your service configuration to serve a custom page or a synthetic response when we receive an error code from your backend.

We assume you are already accustomed to editing and deploying configurations using the web-based configuration application (https://app.fastly.com/). If you are not familiar with basic editing using the app, please see our basic setup (/guides/basic-setup/) and configuration (/guides/basic-configuration/) information to learn more before you continue.

Overview

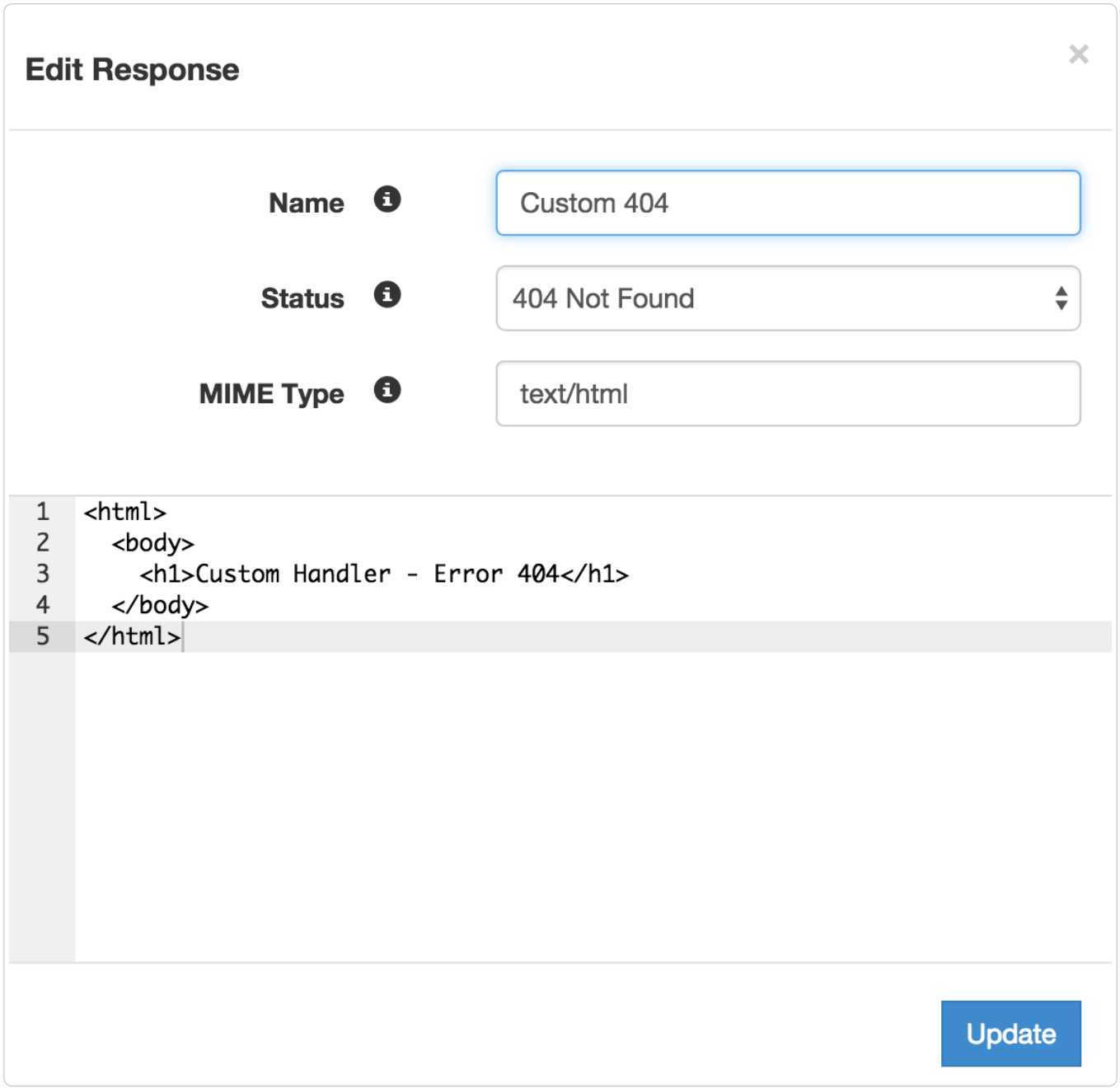
In this tutorial, we will create a response object that contains the HTML you want to serve for your error page. We provide example HTML, but you can use any HTML you see fit. The response object will require that you use a condition in order for it to be served. If you are not familiar with conditions or responses we suggest you read the following:

- Responses Tutorial (/guides/basic-configuration/responses-tutorial)
- Conditions guides (/guides/conditions/)

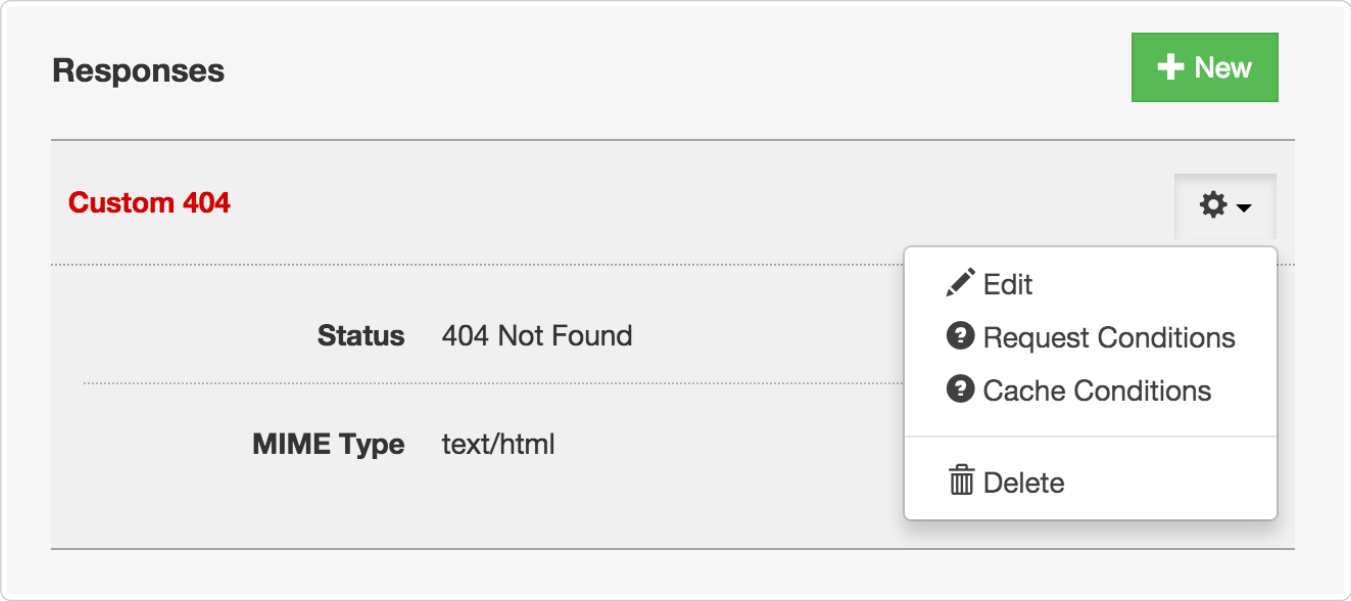
Creating the custom response

1. Log in to the Fastly application and click the **configure** tab (wrench icon).

2. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for your selected service appear.
3. Click **Content** from the section list on the left.
4. In the **Responses** area at the bottom of the page click the **New** button. The New Response window appears.



5. Fill out the **New Response** controls as follows:
   - In the **Name** field, give the response you're creating a name (e.g., Custom 404).
   - From the **Status** menu, select the appropriate status (e.g., 404 Not Found).
   - In the **MIME Type** field specify the Content-Type of the response (e.g., text/html).
6. Click the **Create** button to create your custom response. Your new response appears in the list of responses.
7. Click the gear icon to the right of the name of your new response and select **Cache Conditions** from the menu that appears.

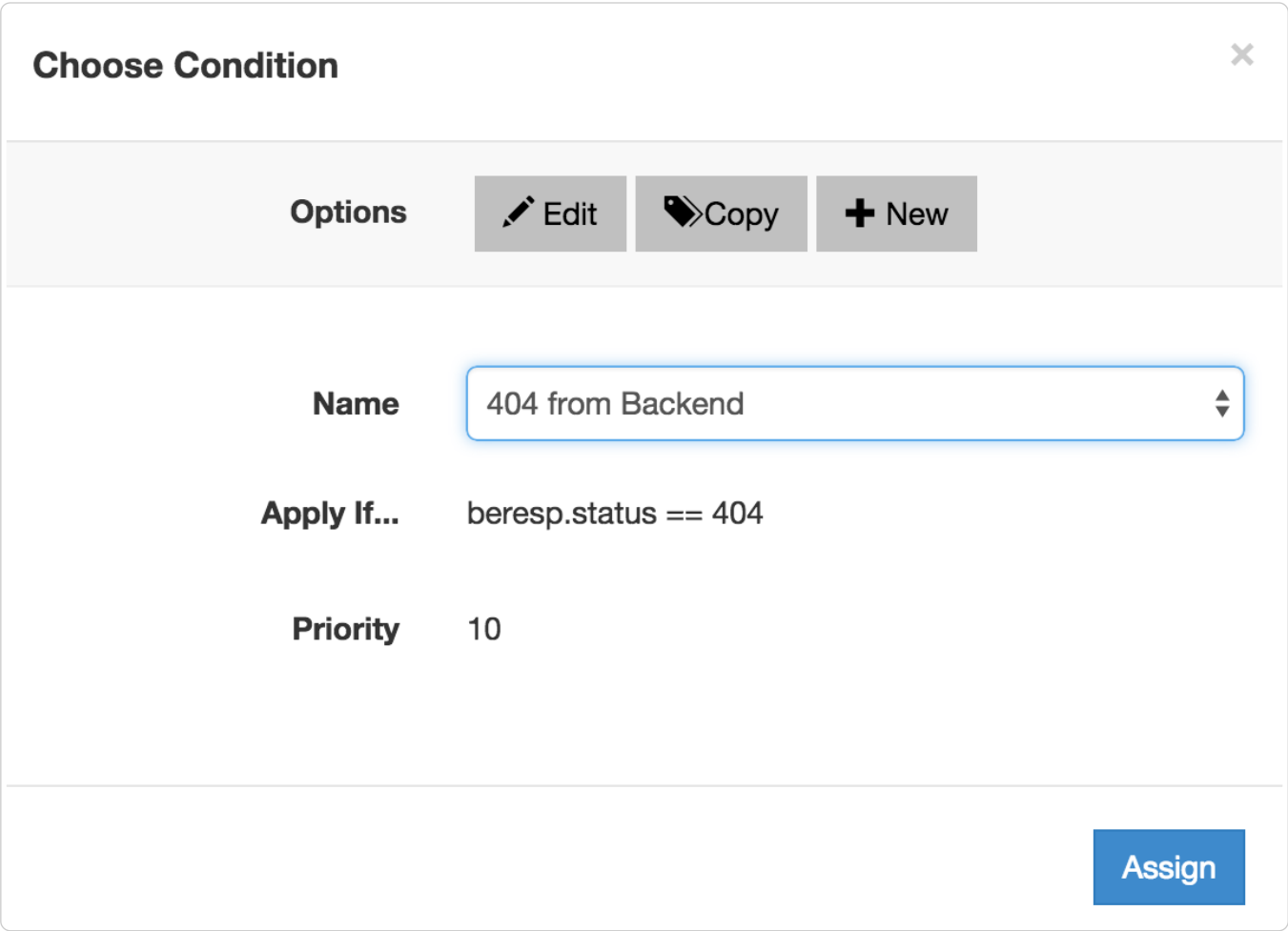


The **New Condition** window appears (if you have previously created a cache condition, click the new button at the top of the edit condition modal that appears).

8. In the **Name** field, give the condition you're creating a name (e.g., 404 Not Found).
9. In the **Apply If** field, type the condition under which the new response occurs. The condition should take the following format:

   `beresp.status == ###`

   where ### equals the status condition you're creating the response for. For instance, using our 404 error example above, your **New Condition** window would look like this:



Using the value of `beresp.status == 404` in the **Apply If** field here tells Fastly to use this response object whenever origin servers return a 404 status. (Remember: The Conditions guides (/guides/conditions/) have more detailed information on conditions.)

10. Click the **Create** button and the condition will be created and applied to the custom response object you made earlier.
11. Deploy your service. Fastly will now serve your custom HTML error page when required.

§ Creating multiple domains at one time (/guides/basic-configuration/creating-multiple-domains-at-one-time)

It is possible to create multiple domains within in a single service programmatically, using Fastly's API (/api/). Before you add domains, however, consider that it may be easier to serve multiple domains through custom VCL (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl). You'll be able to finely tune requests in this manner, yet remain flexible enough that you'll be able to implement as many processes as needed.

Are there domain creation limits?

We set a limit (/guides/customer-support/common-service-and-domain-errors) on the number of domains you can create per service by default. However, if you email support@fastly.com (mailto:support@fastly.com), we may be able to adjust this number for you by working with you to set up and fine-tune domain handling in your service.

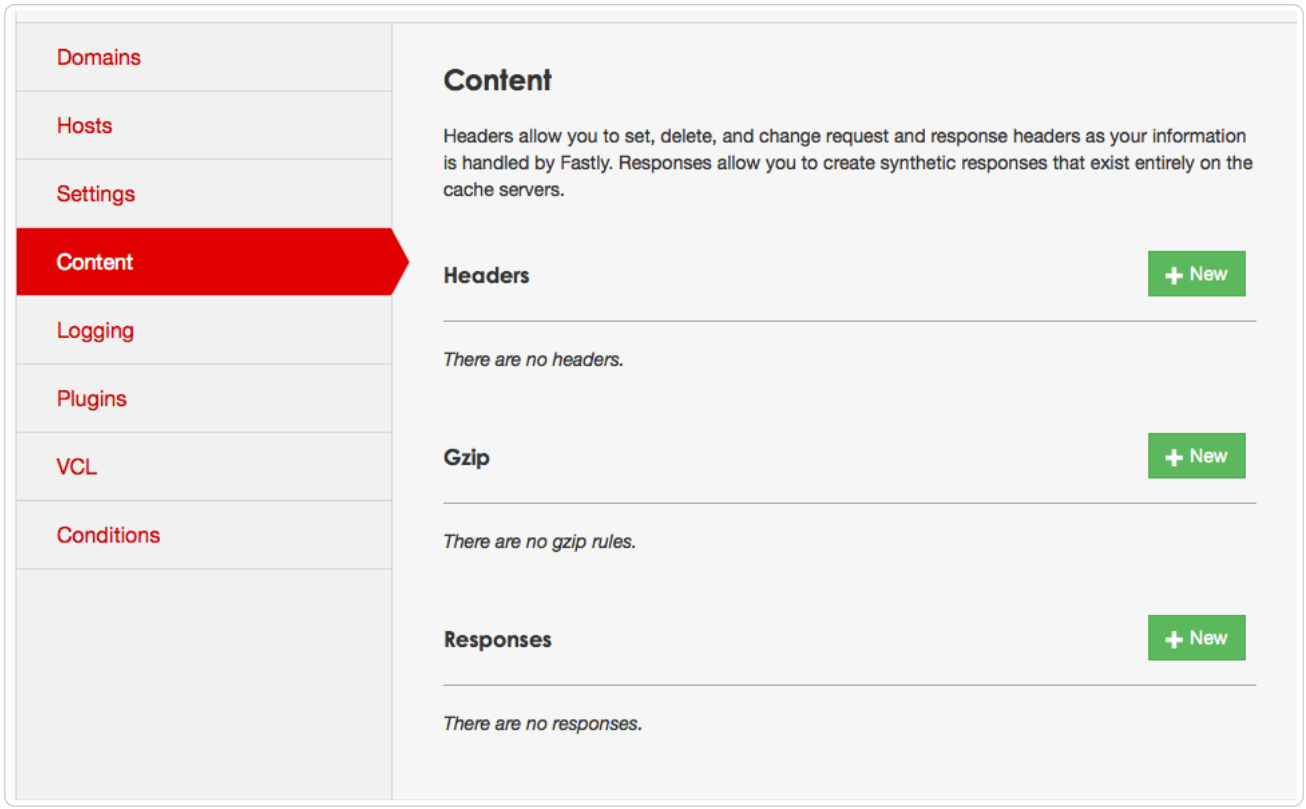
§ Enabling automatic gzipping (/guides/basic-configuration/enabling-automatic-gzipping)

To dynamically gzip content based on file extension or content-type, follow the steps below.

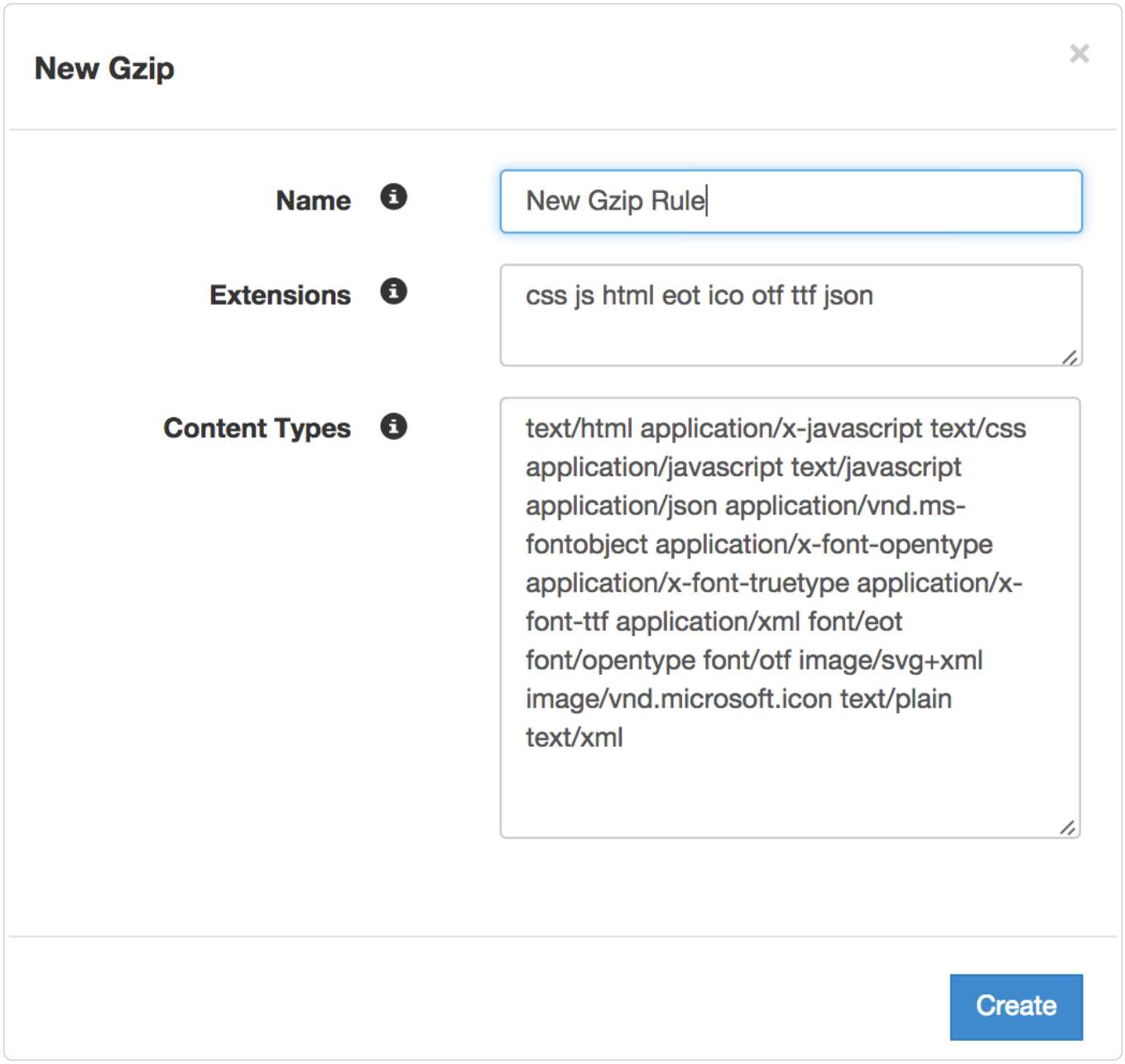
⚠ **WARNING:** Because dynamic gzipping fetches content from origin, compresses it, and then caches it, this feature doesn't work with our ESI feature (/guides/performance-tuning/esi-use). If you enable gzipping, Fastly will stop processing ESI language elements.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Content** section.



6. In the **Gzip** area, click the **New** button to create a new gzip rule. The New Gzip window appears.



7. Fill in the fields of the **New Gzip** window as follows:
   - In the **Name** field, type an arbitrary name for your new gzip rule.
   - In the **Extensions** field, type the file extensions for each file type you wish to have dynamically gzipped, separated by spaces. Only type the file extension; the dot (`.`) and regular expressions are not necessary. We recommend setting the Extensions to `css js html eot ico otf ttf json`.
   - In the **Content Types** field, type the content-type for each type of content you wish to have dynamically gzipped, separated by spaces. Do not use regular expressions. We recommended setting the Content Types to the following:

```
text/html application/x-javascript text/css application/javascript text/javascript application/json application/vnd.ms-fontobject application/x-font-opentype application/x-font-truetype application/x-font-ttf application/xml font/eot font/opentype font/otf image/svg+xml image/vnd.microsoft.icon text/plain text/xml
```

8. Click the **Create** button. The new GZIP rule appears in the GZIP area of the Content section.
9. Click **Activate** at the top of the window to apply the changes to your service.

Automatic exclusion from old browsers automatically

Some web browsers should not receive gzipped content. Fastly's default gzip configuration intentionally excludes old browsers. Specifically, we run the following VCL before any custom VCL:

```
if (req.http.Accept-Encoding) {
  if (req.http.User-Agent ~ "MSIE 6") {
    unset req.http.Accept-Encoding;
  } elsif (req.http.Accept-Encoding ~ "gzip") {
    set req.http.Accept-Encoding = "gzip";
  } elsif (req.http.Accept-Encoding ~ "deflate") {
    set req.http.Accept-Encoding = "deflate";
  } else {
    unset req.http.Accept-Encoding;
  }
}
```

§ Health check frequency (/guides/basic-configuration/health-check-frequency)

Fastly performs health checks (/guides/basic-configuration/health-checks-tutorial) on your origin server based on the Check Frequency setting you select in the New Health Check window. The Check Frequency setting you select will specify approximately how many requests per minute Fastly POPs (https://www.fastly.com/network-map) are checked to see if they pass. There is roughly one health check per Fastly POP per period. Any checks that pass will be reported as "healthy."

To determine origin health, you can configure health checks with the following frequency options:

- **Low** - Approximately 30 requests/minute, where "healthy" means 1 out of 2 must pass
- **Normal** - Approximately 120 requests/minute, where "healthy" means 3 out of 5 must pass
- **High** - Approximately 900 requests/minute, where "healthy" means 7 out of 10 must pass
- **Custom** - A custom frequency and request health pass levels

To set a custom frequency, select **Custom**; the following controls appear:

- **Threshold & Window** - The number of successes per total number health checks. For example, specifying `1/2` means 1 out of 2 checks must pass to be reported as healthy.
- **Initial** - The number of requests to assume as passing on deploy.
- **Interval & Timeout (ms)** - Interval represents the period of time for the requests to run. Timeout represents the wait time until request is considered failed. Both times are specified in milliseconds.

§ Health Checks tutorial (/guides/basic-configuration/health-checks-tutorial)

In this tutorial we will show you how to create health checks that will periodically contact your origin servers to make sure they are still working.

We will assume that you are already accustomed to editing and deploying configurations using the web-based configuration application (https://app.fastly.com). If you are not familiar with basic editing using the application, please see our Help Guides (/guides/) to learn more before moving forward.

Overview

Health checks, while simple in principle, are a little more involved than most other configuration objects. Let's run through the options you have when creating a health check so you have a place to start when working through the tutorial.

- Name - A human-readable identifier for the health check (e.g., "West Coast Origin Check").
- Request - An HTTP method and path to visit on your origin servers when performing the check.
- HTTP Host Header - The HTTP Host Header to set when making the request (e.g. "example.com").
- Expected Response - The HTTP status code the origin servers must respond with for the check to pass (usually "200 OK").
- Check Frequency - How often the origin server is checked with additional parameters that determine if the origin server is up or down. For more information about the additional parameters, see our guide on Health check frequency (/guides/basic-configuration/health-check-frequency).

Fastly will periodically check your origin server based on the options chosen. Pay special attention to the HTTP host header. A common mistake is setting the wrong host. If the origin server does not receive a host it expects, it may issue a 301 or 302 redirect causing the health check to fail. Also, Varnish requires the origin server receiving the health check requests to close the connection for each request. If the origin server does not close the

connection, health checks will time out and fail.

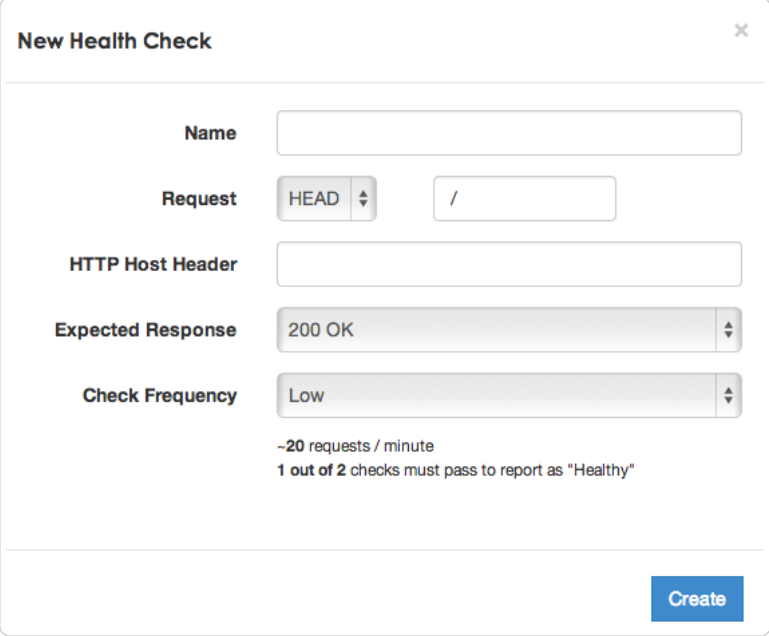
If an origin server is marked unhealthy due to health checks, Fastly will stop attempting to send requests to it. If all origin servers are marked unhealthy, Fastly will attempt to serve stale (/guides/performance-tuning/serving-stale-content). If no stale object is available, a 503 will be returned to the client.

Creating a Health Check

1. Log in to the Fastly application and click the **configure** tab (wrench icon).

2. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for your selected service appear.
3. Click **Hosts** from the section list on the left. The Hosts window appears.

4. In the **Health Checks** area at the bottom of the page, click the green **New** button. The New Health Check window appears.



5. Fill out the fields in the **New Health Check** window. For more information, review the field descriptions in the Overview section.
6. Click the **Create** button to create the health check.

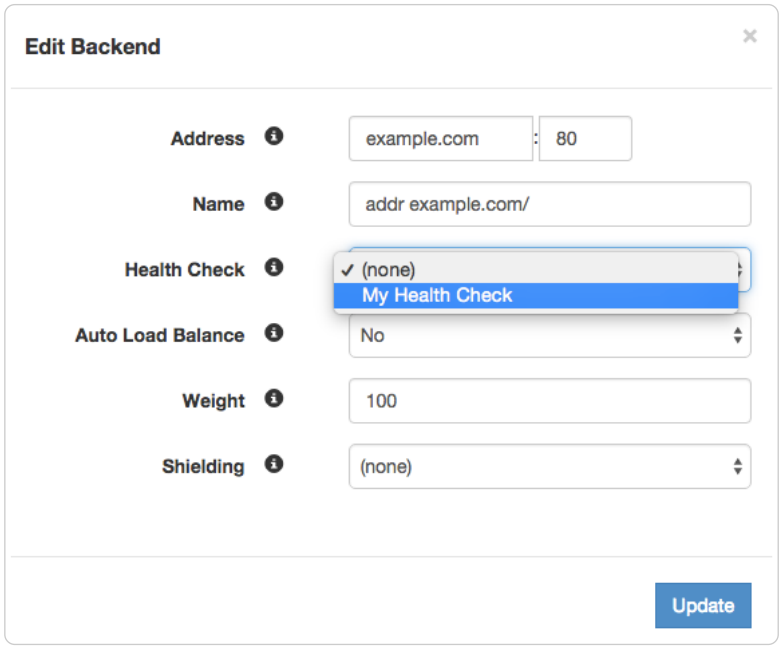
Your new health check will now appear in the list of checks.

Assigning a Health Check

Health checks do nothing on their own, but they can be added as a special parameter to an origin server in your configuration.

1. Edit one of your existing origin servers by clicking the gear to the right of the origin server's name, and then selecting the **Edit** option.

2. From the **Health Check** menu, select the health check you just created.



3. Click **Update**.

Fastly will now use the health check to monitor the selected origin server.

§ How request settings are applied (/guides/basic-configuration/how-request-settings-are-applied)

Requests settings are applied based on the Action you select in the Request Settings window. You can choose any one of the following settings:

- **N/A** - Applies the request settings within the policy and continues through to other request configurations.
- **Lookup** - Immediately searches the cache for content. If none is found (a miss), then the request is sent to the origin. Fastly executes this setting prior to other request configurations.
- **Pass** - Sends the request to the origin each time. Fastly executes this setting immediately and will ignore additional request configurations. See our info on understanding the different PASS action behaviors (/guides/vcl/understanding-the-different-pass-action-behaviors) to learn more.

§ Manipulating the X-Forwarded-For header (/guides/basic-configuration/manipulating-the-x-forwarded-for-header)

You can control what happens to the X-Forwarded-For HTTP header via the **Request Settings** window in the **Settings** pane on the **configure** tab. From the **X-Forwarded-For** menu, select one of the following behaviors:

- **Append** - Appends the client IP to the X-Forwarded-For header.
- **Append All** - Appends the client IP (and edge-cache IP, in case of shielding) to the X-Forwarded-For header. Creates the header if it does not exist yet.

- **Clear** - Clears the X-Forwarded-For header.
- **Leave** - Leaves the X-Forwarded-For header as is, if present.
- **Overwrite** - Overwrites the X-Forwarded-For header with just the client IP.

For more information about requests and responses, see our tutorial (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses).

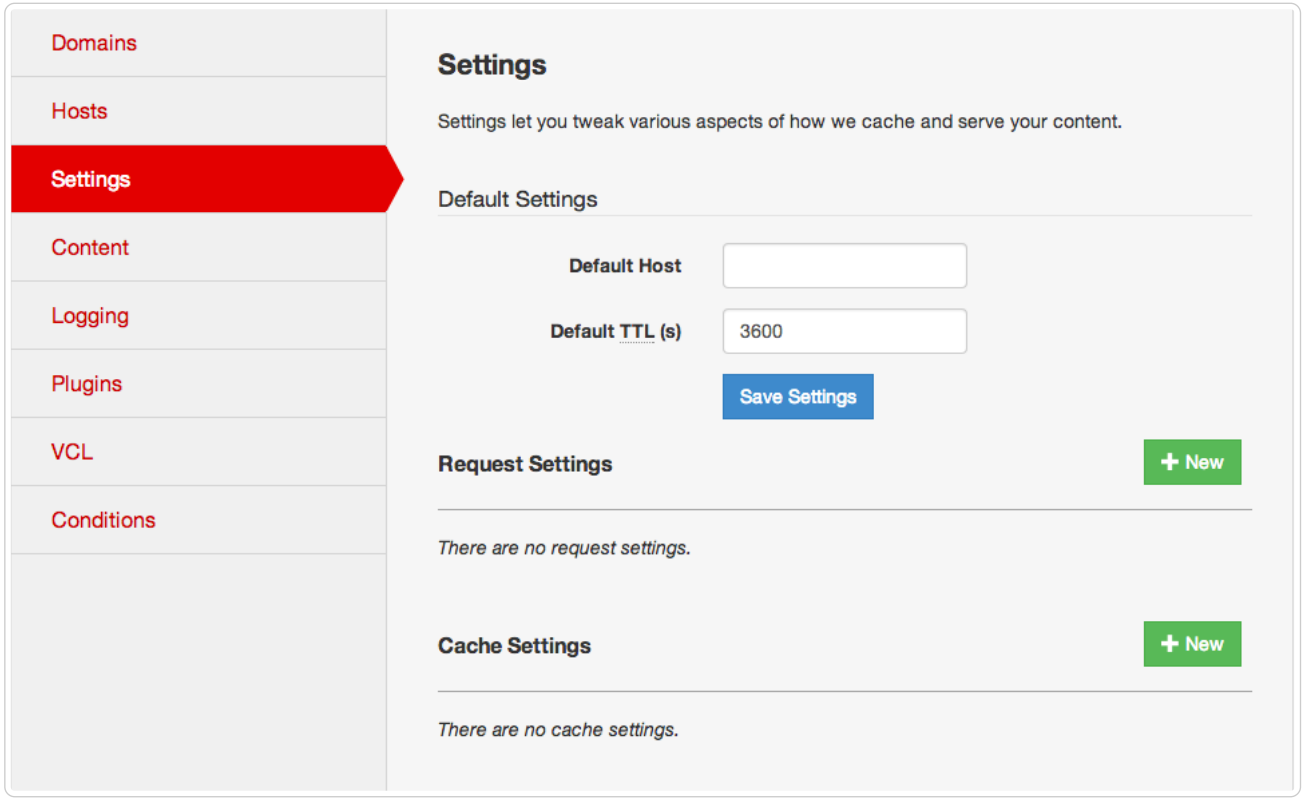
§ Overriding caching defaults based on a backend response (/guides/basic-configuration/overriding-caching-defaults-based-on-backend-responses)

In certain situations you may want to conditionally apply a different caching policy (/guides/performance-tuning/controlling-caching#conditionally-preventing-pages-from-caching) based on a backend response. In this particular case we have backend that on occasion returns 404 errors (e.g., document not found). We don't want those responses to be cached for full caching period of a day but only for 5 minutes. To override default caching we add a cache object and then creation conditions for it.

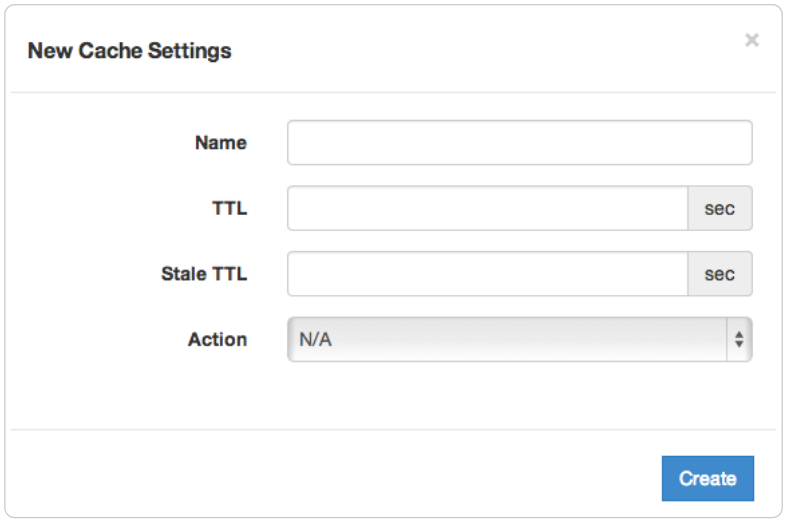
Creating the new Cache Object

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Settings** section.



6. In the **Cache Settings** area, click **New** button. The New Cache Settings window appears.

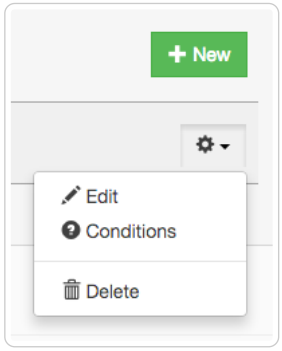


7. Fill out the **New Cache Settings** window as follows:
   - In the **Name** field, type a descriptive name for the new cache settings.
   - In the **TTL** field, type the amount of time, in seconds, to cache the objects (e.g., `300`).
   - In the **Stale TTL** field type the amount of time to serve stale or expired responses, in seconds, should the backend become unavailable (e.g., `300`).
   - From the **Action** menu, select **Deliver**.
8. Click the **Create** button. A new cache object appears in the Cache Settings area of the Settings section.

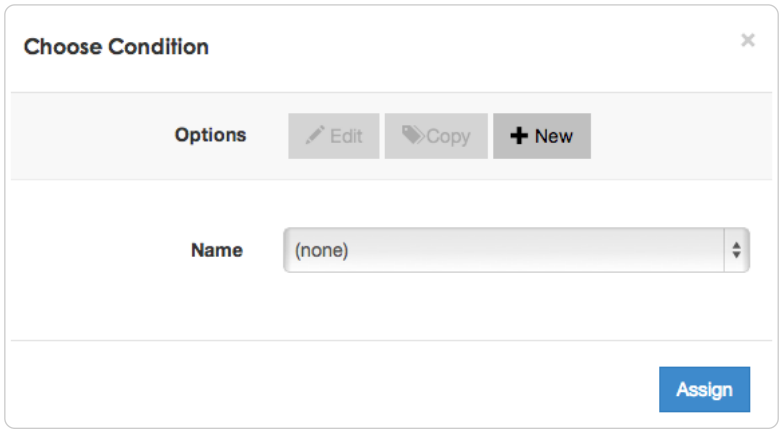
Creating an Override Condition for the new Cache Object

Once the object has been created you will need to add a condition to it.

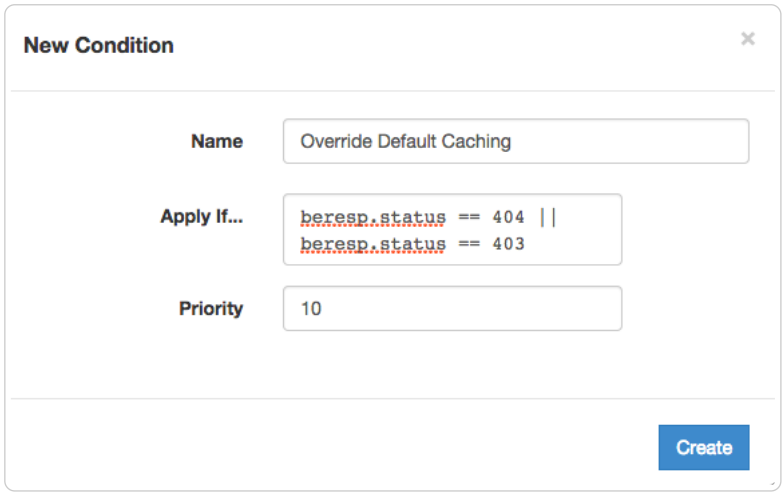
1. Click the gear icon to the right of the object and select **Conditions**.



The Choose Condition window appears with nothing selected.



2. Click **New** at the options at the top of the window. The New Condition window appears.



3. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a descriptive name for the new condition.
   - In the **Apply If** field, type an appropriate backend response header to specify when the condition will be applied. For example, `beresp.status == 404 || beresp.status == 403`
   - Leave the **Priority** field set at its default value.
4. Click **Create** to create the condition.

Once you create the condition, remember to assign it, then deploy the new config version and you are done.

Other notes

You could use any backend response header in the **Apply If** field to make decisions on caching.

For example, `beresp.http.Content-Type ~ "^text/html"` could be used to specify different caching rules for HTML documents.

---

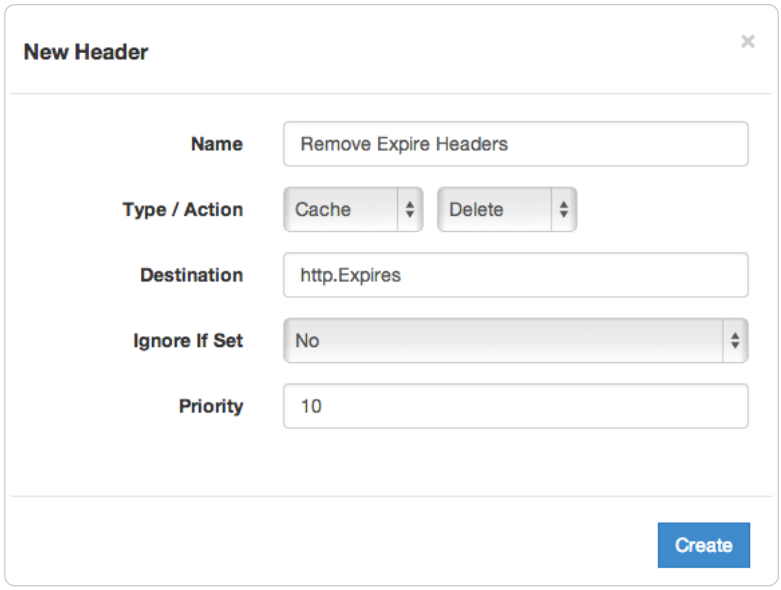
§ Removing headers from backend response (/guides/basic-configuration/removing-headers-from-backend-response)

You can remove headers from any backend response. This may be necessary if your application automatically sets headers. For example, Drupal can set the following Expires and Cache-Control headers to prevent caching:

```
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Last-Modified: Wed, 18 Jul 2012 18:52:16 +0000
Cache-Control: no-cache, must-revalidate, post-check=0, pre-check=0
```

To remove a header from the backend response, add a new header as follows:

1. Log in to the Fastly application.
2. Click **configure** (the wrench icon at the top of the window).
3. From the **Services** menu, select the appropriate service.
4. Click the blue **Configure** button.
5. Click the **Content** link from the list on the left.
6. In the **Headers** area, click **New**. The New Header window appears.



7. Fill out the **New Header** window as follows:
   - In the **Name** field, type a descriptive name for the header rule (e.g., `Remove Expire Headers`).
   - From the **Type** menu, select **Cache**, and from the **Action** menu, select **Delete**.
   - In the **Destination** field, type the name of the header (e.g., `http.Expires`).
   - From the **Ignore If Set** menu, select **No**.
   - In the **Priority** field, type `10`.
8. Click the **Create** button.

★ **TIP:** You may also be interested in our information on setting content type based on file extension (/guides/basic-configuration/setting-content-type-based-on-file-extension).

§ Responses tutorial (/guides/basic-configuration/responses-tutorial)

Fastly allows you to create custom HTTP responses that are served directly from the cache without storing the page on a server. Responses are commonly used to serve small static assets that seldom change and maintenance pages that are served when origins are unavailable. This tutorial shows you how to create your own responses.

NOTE: We assume that you already know how to edit and deploy configurations using the web-based configuration application (https://app.fastly.com/). If you are not familiar with basic editing using the application, please see our help guides (/guides/) to learn more.

Overview

A response has three basic attributes:

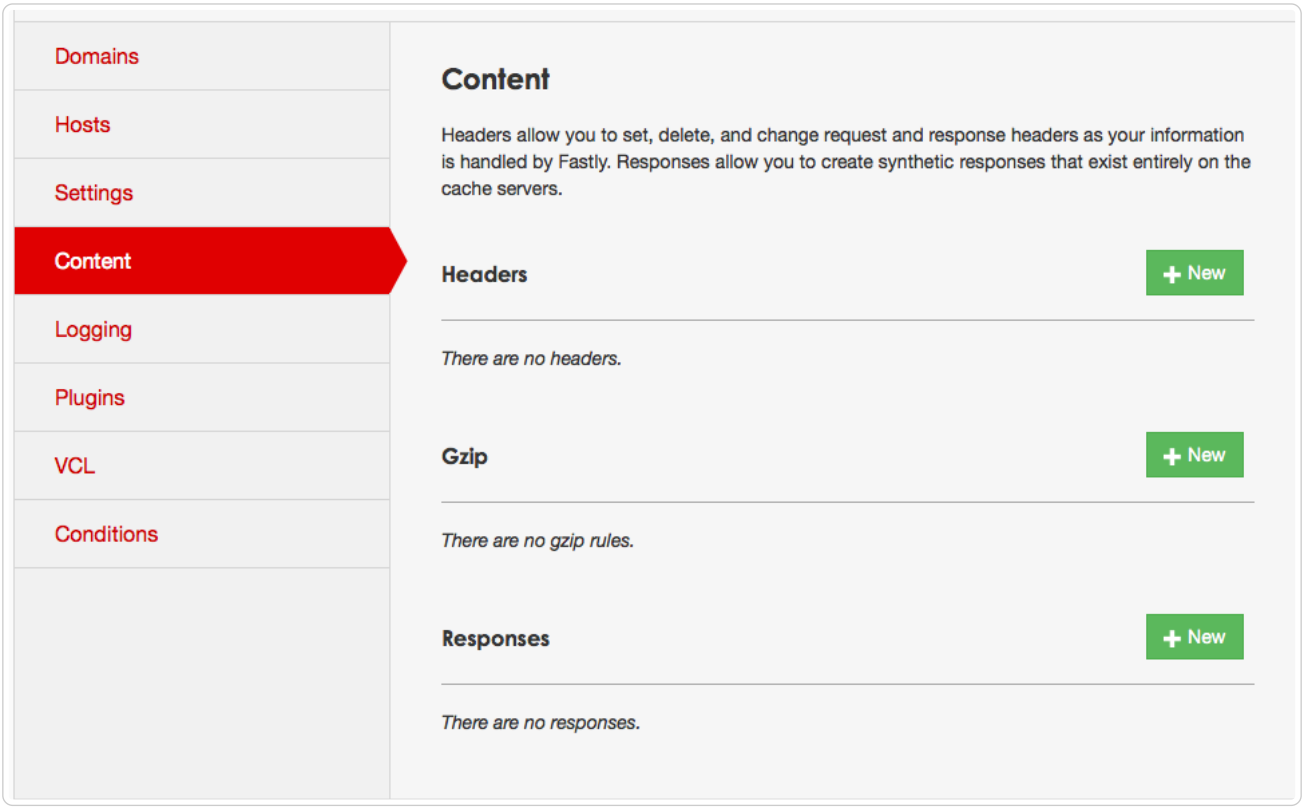
- Name - A human readable identifier for the response
- Status - An HTTP status code to include in the header of the response
- Content - The content to be served when delivering the response

By setting these three attributes and adding a condition to the response, you can very quickly get one up and running on your service. Let's get started!

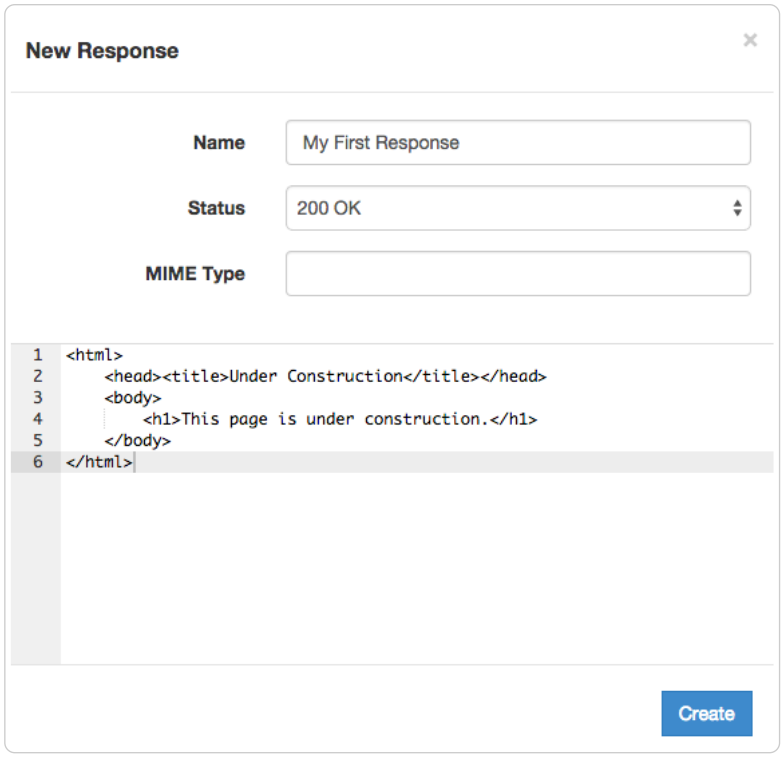
Creating a response

1. Log in to the Fastly application and click the **configure** tab (wrench icon).

2. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for your selected service appear.
3. Click **Content** from the section list on the left.



4. In the Responses area at the bottom of the page, click the **New** button. The New Response window appears.

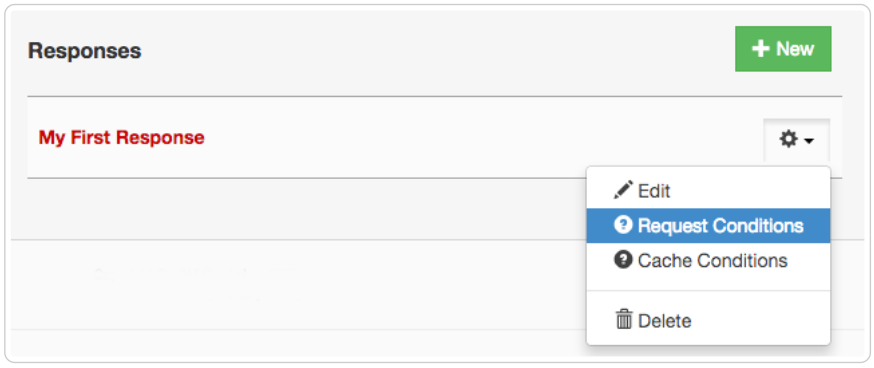


5. Fill out the New Response fields as follows:
   - In the **Name** field, type a human-readable name for the response (e.g., "My First Response")
   - From the **Status** menu, select the appropriate status (e.g., 200 OK).
   - In the **MIME Type** field, type the content type of the response (e.g., text/html).
   - In the main text field, type the response you want to appear when the conditions are met.
6. Click the **Create** button to create your custom response.

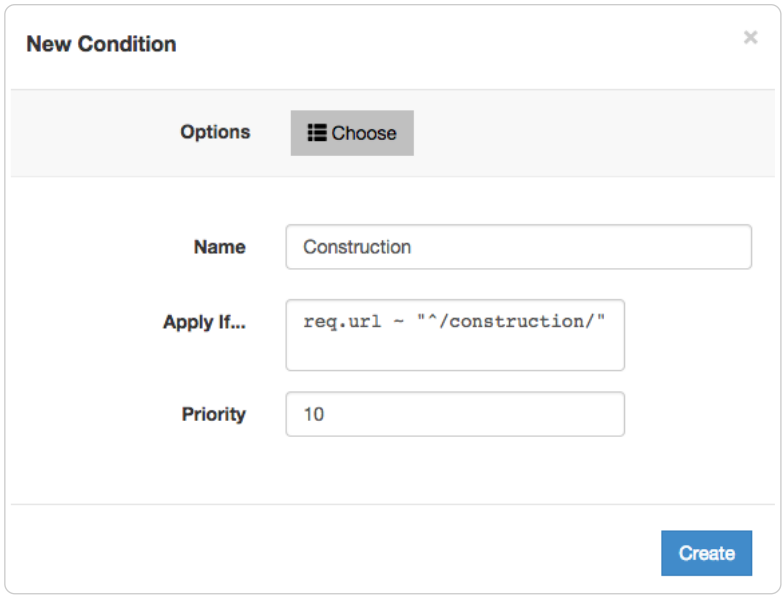
Your new response appears in the list of responses.

Adding conditions

1. Click on the gear menu, and then select **Request Conditions**.



2. If you have not yet added any conditions to your service, the New Condition window will appear automatically. If the new form does not appear, click **New**.



3. Fill out the New Condition fields as follows:
   - In the **Name** field, type a human-readable name for the condition so that it can be easily identified in the future.
   - In the **Apply If...** field, type the condition under which the new response occurs. The condition should take the following format: `req.url ~ "^/construction/"` equals the request condition you're creating the response for. The Conditions subcategory (/guides/conditions/) has more detailed information on conditions.
   - In the **Priority** field, type a priority if needed. Condition priorities are only needed in "interesting" cases, and can usually be left at the default "10" for all response conditions.
4. Click the **Create** button. The condition will be created and applied to the custom response object.
5. Deploy the service.

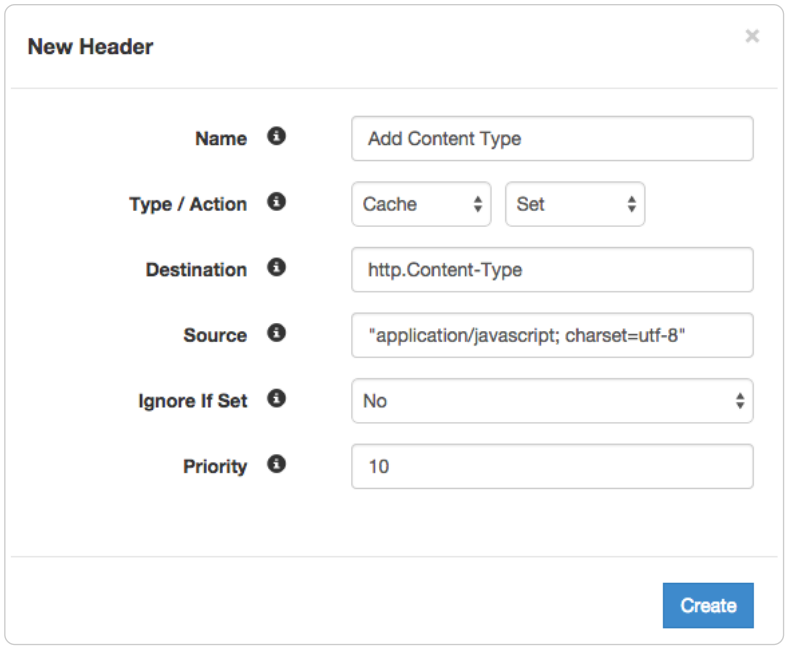
Fastly will now serve your custom response page when the condition is met.

§ Setting Content Type based on file extension (/guides/basic-configuration/setting-content-type-based-on-file-extension)

In some situations you may want to override the content type that a backend returns. To do that you will need to create a new header object and an associated condition.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Content** section.
6. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.



7. Fill out the **New Header** window as follows:
   - In the **Name** field, type an appropriate name, such as `Add Content Type`.
   - From the **Type/Action** menus, select **Cache** and **Set**.
   - In the **Destination** field type `http.Content-Type`.
   - In the **Source** field type the content type you want match, such as `"application/javascript; charset=utf-8"`.
   - From the **Ignore if Set** menu, select **No**.
   - In the **Priority** field, type `10`.
8. Click the **Create** button. A new header appears in the Headers area of the Content section.

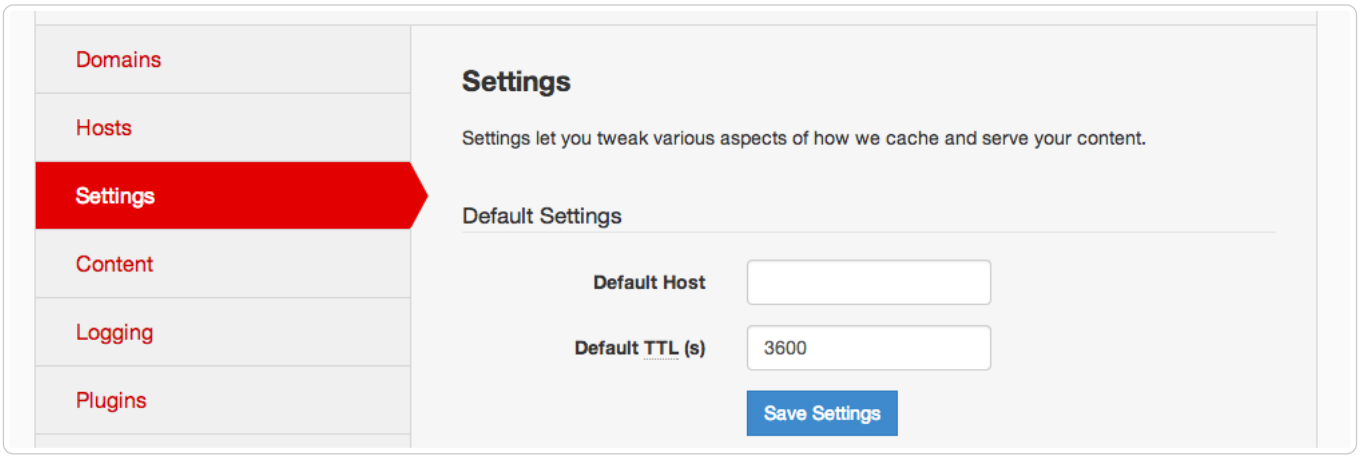
Once you have created the header object you will need to apply a condition (/guides/conditions/). Otherwise, that particular object will apply to all requests.

1. Click the gear icon to the right of the new header name and select **Cache Conditions** from the menu. The Choose Condition window appears.
2. In the **Options** section, click the **New** button to add a new condition. The New Condition window appears.
3. Fill out the next **New Condition** window as follows:
   - In the **Name** field, type a descriptive name, such as `Files ending with .js`.
   - In the **Apply If** field, type the condition that matches your request, such as `req.url.ext == "js"` (to match our request for files ending in .js).
4. Click **Create** to create the new condition.
5. Deploy your configuration changes.

★ **TIP:** You may also be interested in our guide to Removing headers from backend response (/guides/basic-configuration/removing-headers-from-backend-response).

§ Specifying a default host (/guides/basic-configuration/specifying-a-default-host)

Default host settings are used when responding to requests to your origin server. You can specify the name (/guides/basic-configuration/using-a-named-domain-as-an-origin) of your default host on the **configure** tab under the **Settings** section for a specific service.



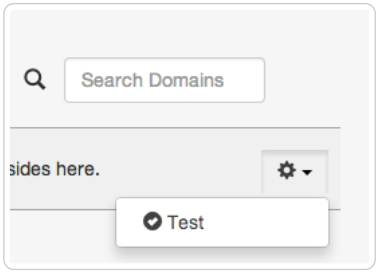
Use this setting especially if you host multiple sites on a single server.

§ Testing setup before changing domains (/guides/basic-configuration/testing-setup-before-changing-domains)

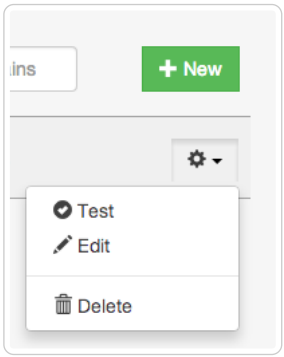
After you deploy your service, but before you change your DNS entries to send your domain to our servers, you can check via a special URL to see how your service is pulled through our network and help you identify if a problem is a DNS issue, or a Fastly configuration problem.

You can test this URL by following these steps:

1. Log in to the Fastly application.
2. Click the **configure** button (the wrench icon at the top of the window).
3. From the **Service** menu, select the service you want to test.
4. Click the **Configure** button.
5. Click **Domains** to view the Domain settings.
6. In the **Domains** area click the gear icon next to the name of the domain you wish to test. One of the following will be true:
   - If the service is currently deployed, only a **Test** selection appears under the menu.



   - If the service is not deployed, both **Test** and **Edit** appear as selections under the menu.



7. Select **Test** from the menu. The domain you are testing will appear in a new browser window.

The test URL generally appears in the format `$domain.global.prod.fastly.net`. For example, if your website were `example.com`, your test URL would appear as `example.com.global.prod.fastly.net`.

IMPORTANT: There are a few special instances where this format is not the case. If the message "unknown domain" appears, this indicates

there are no currently deployed services set up with the specified domain, and Fastly's network does not know how to route the traffic. Contact support@fastly.com (mailto:support@fastly.com) for help if this error appears.

§ Using a named domain as an origin (/guides/basic-configuration/using-a-named-domain-as-an-origin)

Fastly allows the use of domain names (/guides/about-fastly-services/domain-names-and-fastly-services) as origins. Obviously there is a performance impact, but it allows things like EC2 machines and App Engine instances to be used. In fact, when we detect certain conditions, such as the host being part of EC2, we optimize our routing to be more efficient.

§ Using Fastly with apex domains (/guides/basic-configuration/using-fastly-with-apex-domains)

Some customers use only their second level or apex domain (e.g., `example.com` rather than `www.example.com`) as their canonical domain. Due to limitations in the DNS specification, we don't recommend placing a CNAME record at the apex domain or using the CNAME Flattening (e.g., ALIAS or ANAME) features offered by some DNS providers. Instead, we offer Anycast IP addresses for content that must be hosted on a second-level or apex domain.

Where problems exist and why

The DNS instructions in RFC1034 (https://www.ietf.org/rfc/rfc1034.txt) (section 3.6.2) state that, if a CNAME record is present at a node, no other data should be present. This ensures the data for a canonical name and its aliases cannot be different. Because an apex domain requires NS records and usually other records like MX to make it work, setting a CNAME at the apex would break the "no other data should be present" rule.

In general, the problem with apex domains happens when they fail to redirect to their `www` equivalents (`example.com` points nowhere instead of pointing to `www.example.com`). A couple of workaround options exist:

- Only use Fastly for API or AJAX calls, images, and other static assets (e.g., serve `example.com` yourself and CNAME to Fastly for assets at `assets.example.com`)
- Redirect from the apex domain to the version proxied by Fastly (e.g., redirect any requests for `example.com` to `www.example.com`)

Neither of these workarounds, however, are ideal.

Anycast option

Fastly can provide Anycast (https://en.wikipedia.org/wiki/Anycast) IP addresses for content that must be hosted on a second-level or apex domain. We do not charge extra for this service, however, you must be using one of Fastly's paid plans (/guides/account-types-and-billing/accounts-and-pricing-plans) (with or without a contract) in order to take advantage of it.

Our Anycast option allows you to add A records that point your apex domain at Fastly. If the Fastly Anycast IP addresses change we will notify you at status.fastly.com (https://status.fastly.com/). Because Anycast doesn't give us as much flexibility in routing your requests, this option may not be as performant as our CNAME-based system (/guides/basic-setup/adding-cname-records). We recommend you use our CNAME-based system for as much content as possible, particularly for large files or streaming video.

For questions about Anycast and second-level or apex domains, contact support@fastly.com (mailto:support@fastly.com).

• Guides (/guides/) > Basic setup (/guides/basic) > About Fastly services (/guides/about-fastly-services/)

§ About Fastly's Application Programming Interface (API) (/guides/about-fastly-services/about-fastlys-application-programming-interface)

Fastly provides an application programming interface (API) that can be accessed via a number of popular interactive clients. The Fastly API allows customers to manage Fastly services via remote procedure calls instead of the web-based user interface (/guides/about-fastly-services/about-the-web-interface-controls). This currently includes features such as:

- Authentication (/api/auth)
- Configuration (/api/config)
- Historical Stats (/api/stats)
- Purging (/api/purge)
- Remote Logging (/api/logging)

The API features do not include customer account setup, which can only occur through the user interface controls. For examples of each API call in action, including full descriptions of the fields used and examples of requests and responses, see Fastly's API Reference (/api/).

Available API clients

The API's main entry point is https://api.fastly.com (https://api.fastly.com). It can be accessed via the following interactive clients:

- a Perl module
- a Ruby gem
- two different Python libraries
- a Node.js client
- a Scala client

Fastly's API Client web page (/api/clients) contains links to GitHub repositories where these clients can be found. When third-party organizations have supplied these clients, we've noted so on the web page.

DISCLAIMER: Fastly makes no warranty on third-party software. We assume no responsibility for errors or omissions in the third-party software or documentation available. Using such software is done entirely at your own discretion and risk.

Authentication via the API

Nearly all API calls require requests to be authenticated. The Fastly API provides two methods for authenticating API calls: API key and username-password. Most API calls can be authenticated using an API key alone. Those calls that don't use an API key alone (usually at the account level) typically require using an authenticated session cookie instead.

Authentication Method	Description
API Key	Allows the API key (/guides/account-management-and-security/finding-and-managing-your-account-info#finding-and-regenerating-your-api-key) located on a customer's account page (https://app.fastly.com/#account) to be included as a Fastly-Key header.
Username and Password	Allows a POST HTTP command to be issued with the user and password parameters, which returns a cookie that can be stashed and used in subsequent requests.

§ About Fastly's Cached Content Control features (/guides/about-fastly-services/about-fastlys-cached-content-control-features)

Fastly has no set hard limit on how long objects will remain cached (/guides/performance-tuning/controlling-caching#how-long-fastly-caches-content). Instead, Fastly supports customer-configurable Time to Live (TTL) settings and customer controlled content purging.

Time to Live support

Fastly supports the expiration of content via customer-configurable Time to Live (TTL) settings. TTL settings work as timers on cached customer content. When content has resided in the cache for the entire TTL interval, that content is given the status of "expired." Before Fastly delivers requested content that is expired, the cache checks to see if the content is still valid by checking with the customer's application server first.

If the application server says the content remains unchanged, the cache sets the content's status to "valid" and resets its TTL value. If the object has been changed, it is declared "invalid" because the content has expired. The application server delivers updated content. Fastly CDN Service caches the updated content with the status of "valid", and its TTL timer begins to run.

The fetch and refresh process may take a second or more, and during that time, a Fastly cache may receive dozens or hundreds of end-user requests for that content. Fastly's request collapsing feature groups these requests and fulfills them at once when the application server response is received.

Fastly offers customers the option of setting a global, default TTL for cached content control. When set, Fastly's CDN service caches objects in a consistent manner even when applications are inconsistent in doing so.

Instant Purge support

Fastly supports an Instant Purge feature that allows customers to actively invalidate content (/guides/purging/). Rather than requiring a customer's network operations and application staff to guess how frequently each bit of content may change, Fastly allows customers to generate an HTTP Purge method that is sent to the CDN Service whenever an application changes or deletes data in its database. The Fastly CDN Service invalidates the associated content throughout the service's cache network, causing a new version of that content to be retrieved from the application server the next time it is requested.

Fastly allows URL-based and key-based purging, as well as purging of all content at once via specific, configurable purging commands (/api/purge). Fastly currently supports Ruby, Python, PHP, and Perl libraries (/api/clients) for instant purging.

When purging by URL or surrogate key, Fastly's CDN Service can process thousands of changes per second. The invalidation process takes less than 300 milliseconds, making it possible to deliver dynamic content that changes rapidly and unpredictably. Using Instant Purge, customers can eliminate cache-to-origin HTTP traffic that all other CDN services generate to determine if expired objects are still valid.

§ About Fastly's Cloud Accelerator (/guides/about-fastly-services/about-fastlys-cloud-accelerator)

Fastly's Cloud Accelerator (https://www.fastly.com/blog/announcing-fastlys-cloud-accelerator-collaboration-with-google-cloud-platform) provides an integration between Fastly and Google services. Specifically, the integration enables customers of Google's Cloud Platform (https://cloud.google.com) service to connect via peered network interconnections (direct PNIs) directly to Fastly's content delivery network services, thus speeding up content delivery and optimizing backend workload.

When customers sign up for Fastly services (/guides/basic-setup/sign-up-and-create-your-first-service) and configure a Google Cloud Platform service as their origin server, they designate a specific point of presence (POP) to serve as an Origin Shield that handles cached content (/guides/about-fastly-services/how-fastlys-cdn-service-works) from their servers.

Requests from Fastly POPs (/guides/about-fastly-services/fastly-pop-locations) to the Cloud Accelerator Origin Shields are routed over Fastly's network, which leverages optimized TCP connection handling, quick-start and opened connections to ensure fast response times between POPs and through to the end-user. Fastly ensures that requests go directly to the Origin Shield instead of the origin servers. Only requests that the entire network has never handled will go back to the Google Cloud Platform service.

§ About Fastly's Delivery Acceleration features (/guides/about-fastly-services/about-fastlys-delivery-acceleration-features)

Request collapsing

Cached content sometimes must be refreshed when that content becomes "stale" or expires. When multiple end-users request content that is in the process of being refreshed, request collapsing (/guides/performance-tuning/request-collapsing) groups those requests to be satisfied together, as soon as the content is received. This accelerates content delivery by keeping Fastly's CDN Service from repeating duplicate requests to a customer's origin server. Request collapsing is enabled by default.

Grace mode

When an application server becomes unavailable for any reason, end users will normally receive error messages indicating the content they've requested cannot be retrieved. When enabled, grace mode instructs Fastly's CDN Service to accelerate content delivery to end users by continuing to serve stale or expired (but likely still valid) content for a set amount of time. This allows customers to return otherwise unavailable application servers to normal operations and still serve content rather than error messages to end users. By default, grace mode is not configured. Customers must specifically configure their service to serve stale content (/guides/performance-tuning/serving-stale-content).

HTTP request fulfillment

The Fastly CDN Service responds to HTTP GET requests initiated from end users' using a customer's website, or from a program making calls to an Internet-hosted API.

Header support

Fastly's CDN Service supports forwarding HTTP headers to end users when they are received from a customer's origin server. Alternatively, headers can be added, removed, or modified using our edge scripting language either before or after caching a response from the origin. This includes the Cache-Control and Expires headers as well as the Surrogate-Control header. HTTP header support allows customers to send one set of instructions

to the Fastly cache servers and another set of instructions to downstream caches, such as proxies or browsers. In particular, the Surrogate-Control header allows customers to specify how to forward and transform specific header types.

§ About Fastly's On-the-Fly Packaging service (/guides/about-fastly-services/about-fastlys-onthefly-packaging-service)

Fastly offers "on-the-fly," dynamic, video-on-demand content packaging service. Rather than requiring you to pre-package all protocols of a viewer-requested video, Fastly allows you to dynamically package video content in different HTTP streaming formats in real time, using source files. That video content then becomes immediately available to viewers.

IMPORTANT: Fastly's On-the-Fly Packaging (OTFP) service is an add-on service. Our Professional Services team will assist with configuration and testing. To enable OTFP and begin this process, contact your account manager or email sales@fastly.com (mailto:sales@fastly.com) for more details.

Supported On-the-Fly Packaging features

Fastly's OTFP service supports the following specific features:

- **HDS, HLS, and MPEG-DASH packaging.** Fastly provides support for version 1 of the Adobe HTTPS Dynamic Streaming (HDS) specification and support for the ISO/IEC 23009-1:2014 specification defining Dynamic Adaptive Streaming over HTTP (MPEG-DASH). We support all features included in up to version 3 (draft 6) of the HTTP Live Streaming (HLS) specification, and cherry picked features from later versions to support additional customer use cases, like subtitle support and trick play for HLS.
- **Standard codecs.** Fastly supports H.264 and MPEG-4 Part 10 Advanced Video Coding (AVC). Fastly also supports Advanced Audio Coding (AAC), including AAC-LC (low complexity) and HE-AAC (high efficiency), MPEG-1 or Audio Layer III (MP3), and the Dolby AC3 codecs.
- **Source video container format.** Fastly supports the Progressive MP4 specification (specifically the .mp4, unencrypted .mov, and audio-only .m4a extensions) as source container format for packaging into all supported HTTP streaming formats.
- **HLS multi-language subtitles and closed captions.** Fastly provides support for both in-band (EIA-608 (https://en.wikipedia.org/wiki/EIA-608) and CEA-708 (https://en.wikipedia.org/wiki/CEA-708)) and out-of-band (Web Video Text Tracks or WebVTT (https://w3c.github.io/webvtt/)) subtitle and closed caption delivery.
- **HLS timed metadata injection.** Fastly supports HLS time-based metadata (https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/HTTP_Live_Streaming_Metadata_Spec/Introduction/Introduction.htr which allows you embed custom metadata or ad markers about a stream into video segments at specified time instances in ID3v2 format.
- **HLS trick play.** Fastly supports trick play (also called trick mode), a feature that displays video scenes during fast-forwarding and rewinding. The HLS Authoring Specification (https://developer.apple.com/library/tvos/documentation/General/Reference/HLSAuthoringSpec/Requirements.html#//apple_ref/doc/uid/TP4001659 CH2-SW1) requires this feature for distributing video on the Apple TV.
- **Media encryption.** Fastly can encrypt videos packaged into HLS (Envelope or entire segment) and MPEG-DASH (ISO/IEC 23001-7: Common encryption in ISO base media file format file) streaming formats by generating a unique content encryption key for each video, enabling secure video delivery to viewers.
- **Clip creation (also known as "timeline trimming").** Fastly supports clip creation features for all supported packaging formats, allowing you to deliver sections of video without segmenting a longer archived video.

Fastly also provides the following features as part of standard content delivery network services:

- Token-based authentication (/guides/tutorials/enabling-token-authentication) for increased response time by placing validation at the edge
- Geo-IP location (/guides/vcl/geoip-related-vcl-features) and device detection (/guides/vcl/delivering-different-content-to-different-devices) for content targeting
- Edge dictionaries (/guides/edge-dictionaries/) for real-time business rules and decision making at the edge
- Remote log streaming (/guides/streaming-logs/) for data aggregation and viewer diagnostics
- Transport Layer Security (TLS) (/guides/securing-communications/) for secure communications delivery

How Fastly's On-the-Fly Packaging service works

Fastly's OTFP service gets configured between our caching network and your origin storage (e.g., Amazon S3 or Google Cloud Storage, Rackspace Cloudfiles).

When users request manifests or video segments, those requests initially come to Fastly caches instead of going to your origin storage. Fastly's edge caches deliver those objects if they are available and valid. If the objects don't already exist in the edge caches, the requests will be passed on to a designated shield cache (/guides/performance-tuning/shielding) to be delivered instead as long as the objects are available and valid. If neither the edge caches nor the shield cache can deliver the objects, the requests for those objects will go directly to and be fulfilled by the OTFP service which acts as an origin for Fastly's cache nodes.

The OTFP service will make the necessary request to your origin storage to fulfill the original request from the user. The OTFP service also maintains a small, local, in-memory cache for video metadata indexes. These indexes are created using mp4 moov atom (or movie atom) that provide information about the video file such as its timescale, duration, audio and video codec information, and video resolution (among other characteristics).

For adaptive bitrate playback (/guides/on-the-fly-packaging/adaptive-bitrate-playback-url-guidelines), the OTFP service will cache indexes of each quality level requested. If a user requests a manifest, OTFP will look for the corresponding indexes and, if it is available and valid, OTFP will generate the manifest and deliver it to the user. Otherwise, OTFP will fetch the moov atom from origin storage to generate the corresponding index. If a user requests video segments, OTFP will look for the corresponding audio and video sample entries in the cached index, download those samples from origin storage, and package them in the format requested.

§ About Fastly's Origin Shielding features (/guides/about-fastly-services/about-fastlys-origin-shielding-features)

When configuring Fastly's CDN Service during the self-provisioning process (/guides/about-fastly-services/self-provisioned-fastly-services), customers can choose an "origin shield" as a specific point of presence (POP) designated to host cached content from their servers. This server is referred to as a "shield" because it protects a customer's application servers from continuous requests for content.

Shield POPs

Customers can designate a specific POP to serve as a "shield" (/guides/performance-tuning/shielding) for their origin servers. If Fastly's caches do not have the content being requested, they fetch it from the shield server instead of the customer's origin servers. Fastly caches fetch content from a customer's origin server only when the shield server does not have the content being requested.

Load balancing

Customers can designate multiple servers as their origin servers. When two or more application servers are provisioned as origin servers, Fastly's CDN Service will distribute fetch requests for content across those application servers using the round-robin method of distribution. This type of load balancing (/guides/performance-tuning/load-balancing-configuration) is enabled by default; customers must explicitly disable it.

Health checks

Customers have the option to configure Fastly's CDN Service to perform health checks (/guides/basic-configuration/health-checks-tutorial) on their application servers and measure their responsiveness. Health checks are not enabled by default; the customer must specifically enable them. Customers can use health check responsiveness measurements to fine-tune the distribution of fetch requests. Request collapsing is enabled by default.

Request collapsing

Cached content sometimes must be refreshed when that content becomes "stale" or expires. When multiple end-users request cached content that is in the process of being refreshed from origin servers, request collapsing (/guides/performance-tuning/request-collapsing) groups those requests to be satisfied together. This protects customer application servers by keeping Fastly's CDN Service from sending duplicate requests to the origin information.

Grace mode

When an application server becomes unavailable for any reason, end-users will normally receive error messages indicating the content they've requested cannot be retrieved. When enabled, grace mode shields application servers by instructing Fastly's CDN Service to continue to serve stale or expired (but likely still valid) content to end-users for a set amount of time. This allows customers to return otherwise unavailable application servers to normal operations and still serve content rather than error messages to end-users. By default, grace mode is not configured. Customers must specifically configure their service to serve stale content (/guides/performance-tuning/serving-stale-content).

§ About Fastly's Real-Time Log Streaming features (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features)

Fastly's log streaming features allow logging configuration information to be sent over TLS (Transport Layer Security). This means that logging information can be encrypted, which allows customers to send potentially sensitive information to log files without exposing data.

Supported protocols and logging providers

Fastly supports a variety of syslog-compatible logging providers, such as Sumo Logic (/guides/streaming-logs/log-streaming-sumologic), Papertrail (/guides/streaming-logs/log-streaming-papertrail), and Logentries (/guides/streaming-logs/log-streaming-logentries). We also support other methods of sending logs besides the syslog (/guides/streaming-logs/log-streaming-syslog) protocol. We allow pushing of log files to Amazon S3 (/guides/streaming-logs/log-streaming-amazon-s3) buckets as well as any S3-compatible providers (such as DreamHost's DreamObjects). Finally, Fastly supports FTP uploading (/guides/streaming-logs/log-streaming-ftp).

How Real-Time Log Streaming works

As Varnish Cache software processes transactions, it writes instances to a memory log for engineers' diagnostic use. These memory instances are quickly overwritten, but selected data may be formatted into log lines and those can be continually transmitted or "streamed" in real time to a customer's chosen logging service or any other standard syslogd server for reporting purposes.

§ About Fastly's Video Caching service (/guides/about-fastly-services/about-fastlys-video-caching-service)

For customers with their own video packaging infrastructure, Fastly can act as a globally distributed HTTP cache to improve quality of service and increase viewer capacity. When a manifest or video segment is requested by a customer's player, a Fastly edge or shield POP will pull the requested content from the customer's origin media server. Subsequent requests for that content will be served from Fastly's cache servers instead of the customer's origin (read *How Fastly's CDN Services Work* (/guides/about-fastly-services/how-fastlys-cdn-service-works) for more information).

Fastly can cache and deliver any HTTP based media streaming protocol including:

- HTTP Live Streaming (HLS),
- HTTP Dynamic Streaming (HDS),
- HTTP Smooth Streaming (HSS), and
- Dynamic Adaptive Streaming over HTTP (MPEG-DASH).

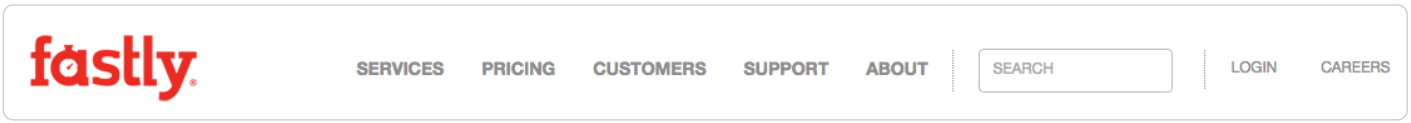
In addition, both live and on-demand streams are supported.

§ About the web interface controls (/guides/about-fastly-services/about-the-web-interface-controls)

In addition to being accessible via Fastly's application programming interface (API), Fastly services can also be accessed via a web-based user interface for users with the appropriate access permissions.

Access to Fastly's user interface controls

Access to Fastly's user interface controls requires users sign up for a Fastly account. Signup is free. Once signed up, customers access the user interface controls via the Fastly login page. The Login page (https://app.fastly.com/) can be accessed using any standard web browser or by clicking the Login button at the top right of almost all pages at Fastly's website (https://www.fastly.com).



Once logged in to a Fastly account via a web browser, the user interface controls appear based on the user's roles and permissions (/guides/user-access-and-control/user-roles-and-how-to-change-them). The controls are grouped by like functions and customers access each functional set by clicking on an appropriate icon at the top of the window. Groups of controls may sometimes be referred to as "tabs" (e.g., the Analytics tab). The default control groups appear as follows:

Not all Fastly features are enabled by default. Some features (e.g., custom VCL (/guides/vcl/uploading-custom-vcl)) must be specifically requested by contacting Fastly Customer Support at support@fastly.com (mailto:support@fastly.com). Once enabled, the appearance of the user interface controls will change to include these services.

About the Account Settings controls

The Account settings controls allow customers to view and modify overall account access settings as well as personal profile settings for a logged in user

What customers can control with the Account Settings

If their logged in permissions (/guides/user-access-and-control/user-roles-and-how-to-change-them) allow it, users can also change their password (/guides/user-access-and-control/email-and-password-changes) to the account and administer the access of multiple users on that account. The Account settings area also displays the logged in user's customer ID, company name, and API key (/guides/account-management-and-security/finding-and-managing-your-account-info#finding-your-service-id) (which can be regenerated (/guides/account-management-and-security/finding-and-managing-your-account-info#finding-and-regenerating-your-api-key) from this location as well). Customers can cancel their account (/guides/account-types-and-billing/accounts-and-pricing-plans#canceling-your-account) via the Account Settings page if they have the appropriate permissions at the time they're logged in.

Account settings controls appear automatically for logged in users with the appropriate access permissions. (/guides/user-access-and-control/user-roles-and-how-to-change-them)

About the Analytics Dashboard

The Analytics Dashboard, sometimes simply referred to as "the Dashboard," allows customers to monitor caching for each of their services, one at a time in real time, as they operate on a second-by-second basis.

What the Analytics dashboard tells users

The Dashboard specifically displays the following information about a service:

- the percentage of requests per second delivered from cache (via the Fastly hitometer)
- a metrics table showing Total Requests, Hit Ratio, Errors, Hits, Hit Time, Misses, and Miss Time
- a global traffic profile (via the global traffic map)

In addition, the Dashboard provides two scrolling graphs. The first graph displays either the number of requests, the bandwidth, or the hit ratio over a two-minute window for any single service as selected by the user. The two-minute rolling average for each of these measurements can also be displayed as an overlay. The second graph displays a histogram showing the distribution of miss latency times. This tells you how quickly your origin is responding to Fastly. One minute after Analytics Dashboard measurement data in these graphs has rolled off the screen, it becomes available for retrieval by the Historical Stats controls (/guides/about-fastly-services/about-the-web-interface-controls#about-the-historical-stats-tool).

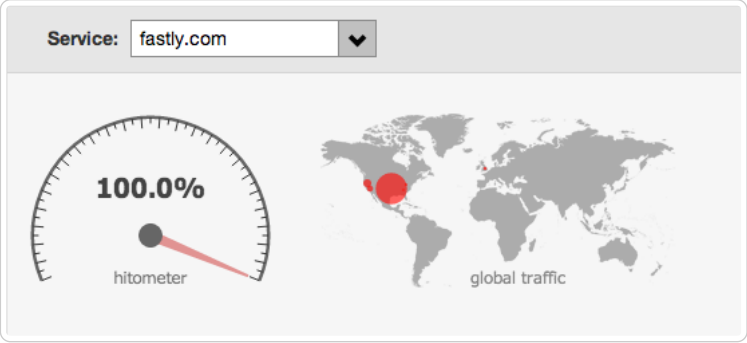
The Dashboard appears automatically for logged in users with the appropriate access permissions (/guides/user-access-and-control/user-roles-and-how-to-change-them); however, it may appear grayed out and blank to some users, with no information displayed in the controls, when a customer's service does not receive enough requests for Fastly to display meaningful information about it in real time.

About the global traffic map

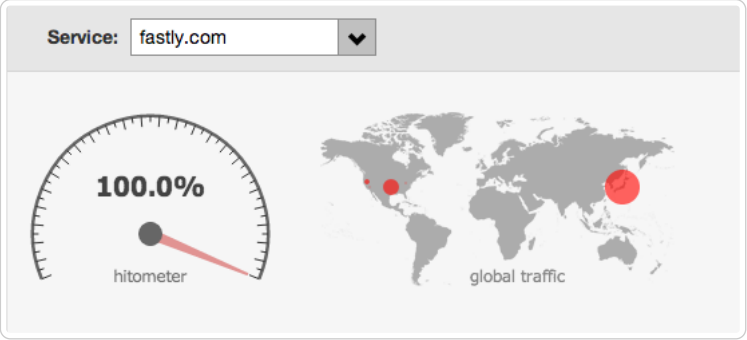
The global traffic map at the top of the Fastly Analytics Dashboard (/guides/about-fastly-services/about-the-web-interface-controls#about-the-analytics-dashboard) represents a real-time visual representation of the general regions of the world in which Fastly's points of presence (POPs) are receiving requests for your service. As each moment passes, the traffic changes, and these changes are reflected in the map visually by red dots that increase or decrease in size as your request traffic ebbs and flows.

For example, the following images show how requests to the Fastly.com website changed over the course of three seconds, at one second intervals:

The first image displays traffic coming into Fastly's POPs on the west coast of the United States (Los Angeles, to be exact), the central region of the US (in this case, Dallas), and in Europe via our Amsterdam POP.



The second image, taken one second later, shows how the global traffic has changed, with a great deal more traffic coming into our central US region, a smaller amount requested from two POPs in the western US region (Los Angeles and San Jose), and a still smaller amount going to the Europe via our London POP.



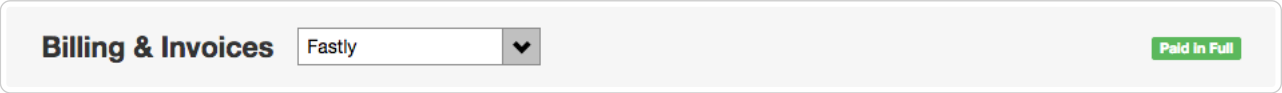
Finally, our third image, again taken one second later, shows how the global traffic has changed once again with traffic going through our Asia/Pacific POP in Tokyo, and smaller amounts going through Dallas and Los Angeles.

About the Billing and Invoice Controls

The Billing and Invoices controls provide customers with an overview of charges accrued to date (/guides/account-types-and-billing/how-we-calculate-your-bill) for the current month. These charges are displayed both in aggregate and detailed by POP region (/guides/about-fastly-services/fastly-pop-locations). Customers can also access their complete billing history, payment statuses, payment amounts, and viewable invoices.

What customers can control via Billing and Invoices

This page shows customers at a glance whether or not their account is paid in full and allows them to change information about the credit card that should be charged for account payment.



Billing controls appear automatically for logged in users assigned the billing or super user roles (/guides/user-access-and-control/user-roles-and-how-to-change-them).

About the Configuration Control Panel

The configuration Control Panel allows customers to define exactly how each instance of their cache should behave and deliver content from each data sources. Customers use the Control Panel to create versions of each service's configuration settings and then use the controls to deploy or deactivate (/guides/basic-setup/working-with-services) them.

What customers can control with the Configuration Control Panel

Specifically, customers with the appropriate permissions can configure and manage (/guides/basic-configuration/):

- the domains used to route requests to a service
- the hosts used as backends for a site and how they should be accessed
- the health checks that monitor backend hosts
- various request and cache settings, headers, and responses that control how Fastly caches and serves content for a service
- how logging (/guides/streaming-logs/setting-up-remote-log-streaming) should be performed and where server logs should be sent (as specified by an RSYSLOG endpoint)
- various third-party plugins available for use with Fastly services (e.g., WordPress (https://github.com/fastly/WordPress-Plugin))
- custom Varnish configuration language (/guides/vcl/) (VCL) files if custom VCL is enabled (/guides/vcl/uploading-custom-vcl)
- how conditions (/guides/conditions/) are mapped and used for a service at various times (e.g., during request processing, when Fastly receives a backend response, or just before an object is potentially cached)

Users can activate configuration changes immediately and roll back those changes just as quickly should they not have the intended effect. The Control Panel also allows users to compare differences (/guides/basic-setup/working-with-services#comparing-different-service-versions) between two configuration versions.

The Control Panel appears automatically for logged in users with the appropriate access permissions (/guides/user-access-and-control/user-roles-and-how-to-change-them).

About the Historical Stats Tool

Fastly's Historical Stats tool provides you with statistical information about your website. We display the information as a series of graphs derived from your sites metrics starting from the moment you first start using Fastly.

Using Fastly's Historical Stats tool allows you to view two core categories of metrics:

- **Caching and Performance**. These metrics help you optimize your website's speed.
- **Traffic**. These metrics help you analyze your website's traffic as it evolves over time.

Taken together, these statistics can help you not only optimize the overall performance of your website but also expose issues that may be hidden by caching, such as degradation of origin performance or unusually high numbers of errors and redirects for a particular timeframe. By analyzing metrics beginning from the very moment you start using Fastly, you can see how your traffic has grown over the lifetime of using Fastly. Use the information to not only diagnose non-trivial issues, but also make informed marketing and business development decisions by identifying clear traffic growth and patterns for your site.

About each of the core metrics

The Historical Stats tool provides a graphical interface to our popular Historical Stats API (/api/stats). Two types of metrics are displayed: caching-performance metrics and displayed traffic metrics.

The displayed caching and performance metrics include the following:

- **Hit Ratio** metrics tell you how well you are caching content using Fastly. This metric represents the proportion of cache hits versus all cacheable content (hits + misses). Increasing your hit ratio improves the overall performance benefit of using Fastly.
- **Cache Coverage** metrics show how much of your site you are caching with Fastly. This metric represents the ratio of cacheable requests (i.e., non "pass" requests) to total requests. Improving your cache coverage by reducing passes can improve site performance and reduce load on your origin servers.
- **Caching Overview** metrics compare Cache Hits, Cache Misses, Synthetic Responses (in VCL edge responses), and Passes (or requests that cannot be cached according to your configuration).

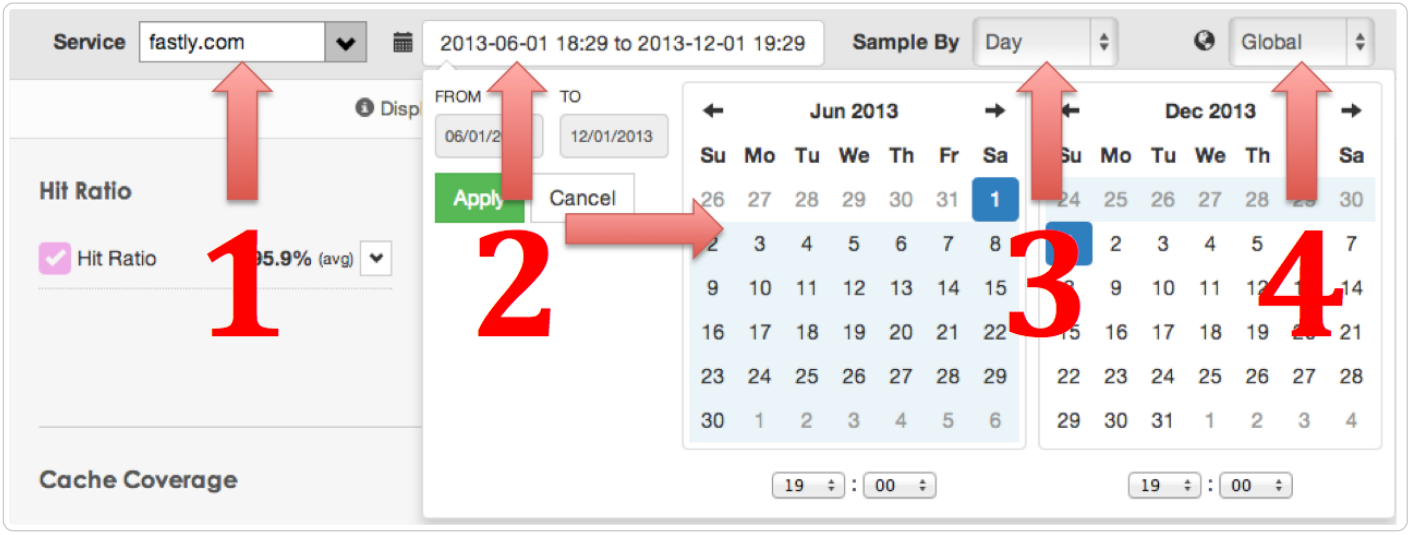
The displayed traffic metrics include the following:

- **Requests** metrics show you the total number of requests over time that were received for your site by Fastly.
- **Bandwidth** metrics show you the amount of bandwidth (measured in bytes) served for your site by Fastly.
- **Header & Body Bandwidth** metrics show the relative proportion of bandwidth (measured in bytes) used to serve the body portion of HTTP requests and the header portion of the requests for your site.
- **Origin Latency** metrics show you the average amount of time to first byte (measured in milliseconds) on a cache miss. High origin latency means that your backends are taking longer to process requests.
- **Error Ratio** metrics show you the ratio of error responses (4XX & 5XX status code errors (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)) compared to the total number of requests for your site. This metric allows you to quickly identify error spikes at given times.
- **HTTP Info, Success, & Redirects** metrics shows the number of HTTP Info (1XX), Success (2XX), and Redirect (3XX) statuses served for your site using Fastly.
- **HTTP Client and Server Errors** metrics shows the number of HTTP Client Errors (4XX), and Server Errors (5XX) served for your site by Fastly.

IMPORTANT: Display of historical statistics are delayed between 15 to 30 minutes. and there must be a statistically measurable amount of traffic to see the graphs.

Controlling the amount of historical information displayed

You have full control over how you view the historical information.

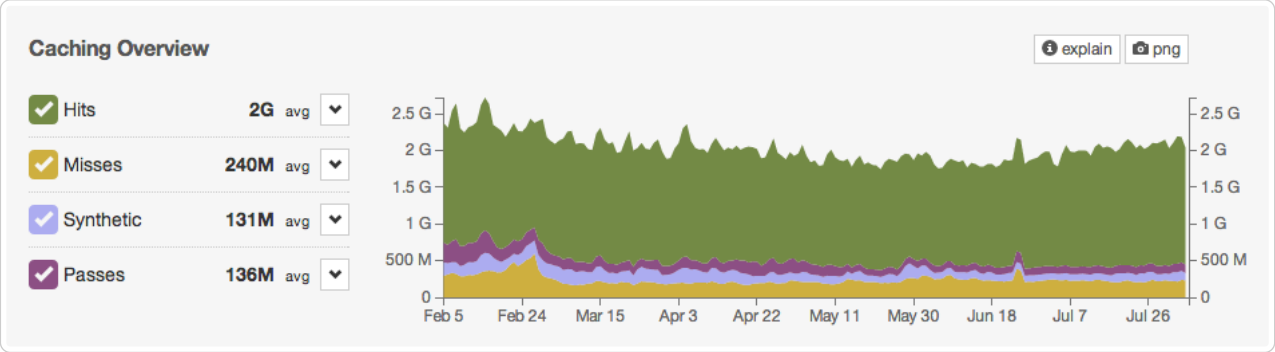


Specifically you can choose:

1. A single service or all services aggregated together
2. The exact date and time range you wish to see
3. How often to sample the data (by Day, Hour, or Minute)
4. Whether to view worldwide data or only data from a specific region (USA, Europe, Australia/New Zealand, or Asia Pacific)

About the graph controls

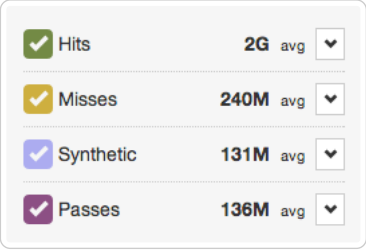
All historical information, regardless of which metric you're viewing, appears as a graph with three major sections.



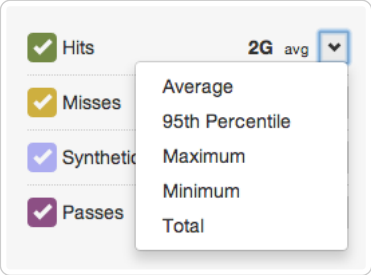
Specifically, you can interact with:

1. The Statistics Display controls
2. The time-series graph
3. The utility buttons

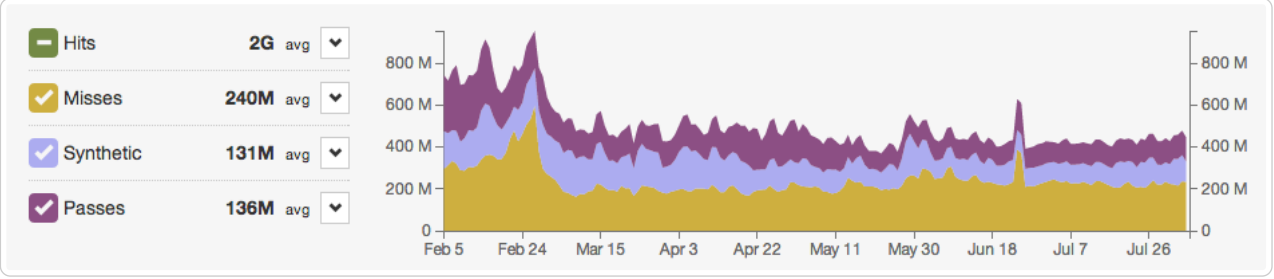
Use the Statistics Display Controls to view other information about the statistics displayed in each time-series graph. For example, the Caching Overview graph includes the statistics for hits, misses, synthetic responses, and passes:



Notice the word "avg" and the menu button to the right of each colored label. The menu buttons control the statistical function used to display the values in each graph. You can choose to display each statistic independently as an average, as a 95th percentile, as a minimum, as a maximum, or their total. Simply select the menu button next to each statistical value and set them to the desired function:

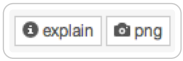


You can even exclude certain statistics entirely. For example, in this Caching Overview graph, we've completely hidden Hits from the display by clicking the checkmark button next to the word "Hits" in the controls.



Notice that the checkmark in the statistical controls changes to a minus sign when a statistic is hidden. The actual numbers still appear in the controls but the graph doesn't display it.

Two utility buttons appear at the top, far right of each time-series graph:



When clicked, the explain button provides a quick explanation of the displayed graph in its default state and how to interpret the information it presents. Clicking the png button downloads the currently displayed time-series graph as a .png file, which can then be used in presentations and reports.

When the Historical Stats Tool appears

Historical Stats controls appear automatically for logged in users with the appropriate access permission; however, the general display of historical stats are delayed between 15-30 minutes and require a statistically measurable amount of traffic to appear in the graphs at all.

---

§ Content and its delivery (/guides/about-fastly-services/content-and-its-delivery)

Content types delivered Fastly

The underlying protocol used by the World Wide Web to define how content is formatted and transmitted is called the Hypertext Transfer Protocol (HTTP). Fastly's CDN Service delivers all HTTP-based file content (e.g., HTML, GIF, JPEG, PNG, JavaScript, CSS) including the following:

- Static content
- Dynamic content
- Video content

Each content type is described below.

Static content

Static content includes content that remains relatively unchanged. Fastly can control static content in two ways:

- using the time to live (TTL) method, where Fastly's cache re-validates the content after expiration of the TTL, or
- using Fastly's Instant Purge functionality, in which content remains valid until the cache receives a purge request (/guides/purging/) that invalidates the content.

Examples of static content include images, css, and javascript files.

Dynamic content

Dynamic content includes content that changes at unpredictable intervals, but can still be cached for a fraction of time. We serve this dynamic content by taking advantage of Fastly's Instant Purge functionality. Using this functionality, dynamic content remains valid only until a Fastly cache receives a purge request (/guides/purging/) that invalidates the content. Fastly understands that the rate of those purge requests cannot be predicted. Dynamic content may change frequently as a source application issues purge requests in rapid succession to keep the content up to date. Dynamic content can, however, remain valid for months if there are no changes requested.

Examples of dynamic content include sports scores, weather forecasts, breaking news, user-generated content, and current store item inventory.

Video content

Video content includes:

- Live video streams
- Video on Demand (VOD) content libraries

Video content can be served using standard HTTP requests. Specifically, Fastly supports HTTP Streaming standards, including HTTP Live Streaming (HLS), HTTP Dynamic Streaming (HDS), HTTP Smooth Streaming (HSS), and MPEG-DASH. For Fastly's CDN Service to deliver video, the video must be packaged.

Content sources supported by Fastly

Fastly caches deliver various types of content from many different sources. Supported sources include:

- Websites

- Internet APIs
- Internet Applications
- Live and Live Linear Video
- Video on Demand (VOD) Libraries

Regardless of the content source, the content's source server must communicate using HTTP. HTTP defines specific types of "methods" that indicate the desired action to be performed on content. The manner in which those HTTP methods are used (the standard, primary methods being GET, POST, PUT, and DELETE) can be labeled as being RESTful (https://en.wikipedia.org/wiki/Representational_state_transfer) or not. Fastly supports RESTful HTTP by default, but also can support the use of non-RESTful HTTP as long as the method used is mapped to its appropriate cache function. Each of the content sources supported by Fastly are described in more detail below.

Websites

Websites are servers that provide content to browser applications (e.g., Google's Chrome, Apple's Safari, Microsoft's Internet Explorer, Opera Software's Opera) when end users request that content. The content contains both the requested data and the formatting or display information the browser needs to present the data visually to the end user.

With no CDN services involved, browsers request data by sending HTTP GET requests that identify the data with a uniform resource locator (URL) address to the origin server that has access to the requested data. The server retrieves the data, then constructs and sends an HTTP response to the requestor. When a CDN Service is used, however, the HTTP requests go to the CDN rather than the origin server because the customer configures it to redirect all requests for data to the CDN instead. Customers do this by adding a CNAME or alias for their origin server that points to Fastly instead.

Internet APIs

Application program interfaces (APIs) serve as a language and message format that defines exactly how a program will interact with the rest of the world. APIs reside on HTTP servers. Unlike the responses from a website, content from APIs contain only requested data and identification information for that data; no formatting or display information is included. Typically the content serves as input to another computing process. If it must be displayed visually to an end user, a device application (such as, an iPad, Android device, or iPhone Weather application) does data display instead.

Legacy internet applications

Legacy Internet applications refer to applications not originally developed for access over the Internet. These applications may use HTTP in a non-RESTful manner. They can be incrementally accelerated without caching, benefiting only from the TCP Stack optimization done between edge Fastly POPs and the Shield POP, and the Shield POP to the origin. Then caching can be enabled incrementally, starting with the exchanges with the greatest user-experienced delay.

Live and live linear video streams & video on demand libraries

Live and live linear video content (for example, broadcast television) is generally delivered as a "stream" of information to users, which they either choose to watch or not during a specific broadcast time. Video on demand (VOD), on the other hand, allows end users to select and watch video content when they choose to, rather than having to watch at a specific broadcast time.

Regardless of which type of video content an end user experiences, a video player can begin playing before its entire contents have been completely transmitted. End users access the video content from a customer's servers via HTTP requests from a video player application that can be embedded as a part of a web browser. Unlike other types of website content, this content does not contain formatting or display information. The video player handles the formatting and display instead.

When the video content is requested, the customer's server sends the content as a series of pre-packaged file chunks along with a manifest file required by the player to properly present the video to the end user. The manifest lists the names of each file chunk. The video player application needs to receive the manifest file first in order to know the names of the video content chunks to request.

"Pre-packaging" in this context refers to the process of receiving the video contents, converting or "transcoding" the stream into segments (chunks) for presentation at a specific dimension and transmission rate, and then packaging it so a video player can identify and request the segments of the live video a user wants to view.

To request video delivery on your account, please contact your Fastly Account Representative at sales@fastly.com (mailto:sales@fastly.com).

---

§ Domain names and Fastly's services (/guides/about-fastly-services/domain-names-and-fastly-services)

A domain name is a component of a Uniform Resource Locator (https://en.wikipedia.org/wiki/Uniform_Resource_Locator) (URL). A domain name represents an Internet Protocol (https://en.wikipedia.org/wiki/Internet_Protocol) (IP) resource and, in this context, refers to the IP address of a server computer hosting a website (https://en.wikipedia.org/wiki/Web_site), the website itself, or any other service communicated via the Internet. Customers associate their domain names with their origin (content source) when provisioning a Fastly service.

Domain names are made up of components as follows:

Domain Name Component	Example
URL	http://www.example.com/index.html
Top-Level Domain Name	com
Second-Level Domain Name	example.com
Hostname	www.example.com

Domain names are registered with a domain name registrar. Fastly is not a domain name registrar.

Fastly supports the use of multiple subdomains for the same origin server, and allows the specification of any number of subdomains for each origin. Some customers use only their second level or apex domain (/guides/basic-configuration/using-fastly-with-apex-domains) (e.g., example.com rather than www.example.com) as their canonical domain. Due to limitations in the DNS specification, we don't recommend placing a CNAME record at the apex domain or using the CNAME Flattening (e.g., ALIAS or ANAME) features offered by some DNS providers. Instead, we offer Anycast IP addresses for content that must be hosted on a second-level or apex domain.

§ Fastly POP locations (/guides/about-fastly-services/fastly-pop-locations)

Our points of presence (POPs) on the Internet are strategically placed at the center of the highest density Internet Exchange Points around the world. Fastly's Network Map (https://www.fastly.com/network-map) shows a detailed view of the current and planned locations of all Fastly POPs. For more information, about how we choose these locations, check out our blog post How Fastly Chooses POP Locations (https://www.fastly.com/blog/how-fastly-chooses-pop-locations).

Once you're signed up for Fastly service (either through a test account (/guides/account-types-and-billing/accounts-and-pricing-plans) or a paid plan) you can a see a live, real-time visual representation (/guides/about-fastly-services/about-the-web-interface-controls#about-the-global-traffic-map) of the general regions of the world in which Fastly's points of presence (POPs) receive requests for your service.

Will Fastly ever adjust POP locations or service regions? How will I be notified?

Fastly continues to grow its network footprint, adding new service POPs in the process. At times, expansion may result in the addition of new billable regions to our network. We'll announce new POP locations and new billable regions in advance through our network status page (/guides/debugging/fastlys-network-status) at status.fastly.com (https://status.fastly.com/). Please contact sales@fastly.com with specific contract or billing questions.

§ How caching and CDNs work (/guides/about-fastly-services/how-caching-and-cdns-work)

Fastly is a Content Delivery Network, or CDN. CDNs work on the principle that once a piece of content has been generated it doesn't need to be generated again for a while so a copy can be kept around in a cache. Cache machines - ours especially! - are optimized to serve small files *very very* quickly. CDNs typically have caches placed in data centers all around the world - when a user requests information from a customer's site they're actually redirected to the set of cache machines closest to them instead of the customer's actual servers. This means that a European user going to an American site gets their content anywhere from 200-500ms faster. CDNs also minimize the effects of a cache miss. A cache miss occurs when a user requests a bit of content and it is not in the cache at that moment (either because it's expired, because no-one has asked for it before, or because the cache got too full and old content was thrown out).

What can be cached?

CDNs are quite good at managing a cache of small, static resources (for example, static images, CSS files, Javascripts, and animated GIFs). CDNs are also popular for offloading expensive-to-serve files like video and audio media.

At Fastly, our architecture (known as a *reverse proxy*) is designed to enable customers to go a step further and cache entire web pages for even more efficient handling of your traffic.

★ **TIP:** Static files + media objects + web pages = your whole site. With the right service configuration (which we can assist you in setting up) Fastly can reduce your backend traffic by orders of magnitude with no loss in control over the content your users see.

Managing the Cache

Caching serves as a powerful weapon in your make-the-site-faster arsenal. However, most objects in your cache aren't going to stay there permanently. They'll need to expire so that fresh content can be served. How long that content should stay in the cache might be mere seconds or a number of minutes or even a year or more.

How can you manage which of your content is cached, where, and for how long? By setting policies that control the cached data. Most caching policies are implemented as a set of HTTP headers sent with your content by the web server (as specified in the config or the application). These headers were designed with the client (browser) in mind but CDNs like Fastly will also use those headers as a guide on caching policy.

Expires

The `Expires` header is the original cache-related HTTP header and tells the cache (typically a browser cache) how long to hang onto a piece of content. Thereafter, the browser will re-request the content from its source. The downside is that it's a static date and if you don't update it later, the date will pass and the browser will start requesting that resource from the source every time it sees it.

If none of the following headers are found in the request, Fastly will respect the `Expires` header value.

Cache-Control

The `Cache-Control` headers (introduced in the HTTP 1.1 specification) cover browser caches and in most cases, intermediate caches as well:

- `Cache-Control: public` - Any cache can store a copy of the content.
- `Cache-Control: private` - Don't store, this is for a single user.
- `Cache-Control: no-cache` - Re-validate before serving this content.
- `Cache-Control: no-store` - Don't store this content. Ever. At all. Please.
- `Cache-Control: public, max-age=[seconds]` - Caches can store this content for *n* seconds.
- `Cache-Control: s-maxage=[seconds]` - Same as max-age but applies specifically to proxy caches.

Only the `max-age`, `s-maxage`, and `private` Cache-Control headers will influence Fastly's caching. All other Cache-Control headers will not, but will be passed through to the browser. For more in-depth information about how Fastly responds to these Cache-Control headers and how these headers interact with Expires and Surrogate-Control, check out our Cache Control Tutorial (/guides/tutorials/cache-control-tutorial).

NOTE: For more information on the rest of the Cache-Control headers, see the relevant section in Mark Nottingham's Caching Tutorial (https://www.mnot.net/cache_docs/#CACHE-CONTROL).

Surrogate Headers

`Surrogate` headers are a relatively new addition to the cache management vocabulary (described in this W3C tech note (http://www.w3.org/TR/edge-arch/)). These headers provide a specific cache policy for proxy caches in the processing path. `Surrogate-Control` accepts many of the same values as `Cache-Control`, plus some other more esoteric ones (read the tech note for all the options).

One use of this technique is to provide conservative cache interactions to the browser (for example, `Cache-Control: no-cache`). This causes the browser to re-validate with the source on every request for the content. This makes sure that the user is getting the freshest possible content. Simultaneously, a `Surrogate-Control` header can be sent with a longer `max-age` that lets a proxy cache in front of the source handle most of the browser traffic, only passing requests to the source when the proxy's cache expires.

With Fastly, one of the most useful `Surrogate` headers is `Surrogate-Key`. When Fastly processes a request and sees a `Surrogate-Key` header, it uses the space-separated value as a list of tags to associate with the request URL in the cache. Combined with Fastly's Purge API (/api/purge) an entire collection of URLs can be expired from the cache in one API call (and typically happens in around 1ms). `Surrogate-Control` is the most specific.

Fastly and Cache Control Headers

Fastly looks for caching information in each of these headers as described in our Cache-Control docs (/guides/tutorials/cache-control-tutorial). In order of preference:

- `Surrogate-Control:`
- `Cache-Control: s-maxage`
- `Cache-Control: max-age`

- `Expires:`

Shielding

When an object or collection of objects in the cache expires, the next time any of those objects are requested, the request is going to get passed through to your application. Generally, with a good caching strategy, this won't break things. However, when a popular object or collection of objects expires from the cache, your backend can be hit with a large influx of traffic as the cache nodes refetch the objects from the source.

In most cases, the object being fetched is not going to differ between requests, so why should every cache node have to get its own copy from the backend? With Shield Nodes, they don't have to. Shielding configured through the Fastly application (/guides/performance-tuning/shielding) allows you to select a specific data center (most efficiently, one geographically close to your application) to act as a shield node. When objects in the cache expire, the shield node is the only node to get the content from your source application. All other cache nodes will fetch from the shield node, reducing source traffic dramatically.

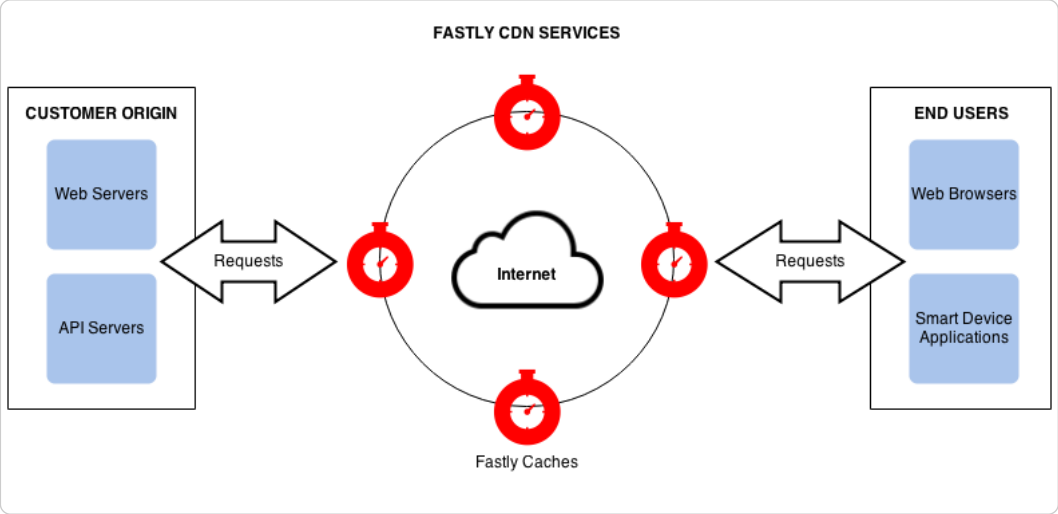
Resources

- Wikipedia: Reverse Proxy (https://en.wikipedia.org/wiki/Reverse_proxy)
- Fastly's Cache-Control docs (/guides/tutorials/cache-control-tutorial)
- Mark Nottingham's Caching Tutorial (https://www.mnot.net/cache_docs/#CACHE-CONTROL)
- Surrogate header W3C tech note (http://www.w3.org/TR/edge-arch/)

§ How Fastly's CDN Service Works (/guides/about-fastly-services/how-fastlys-cdn-service-works)

Fastly is a content delivery network (https://en.wikipedia.org/wiki/Content_delivery_network) (CDN). We serve as an Internet intermediary and offer the Fastly CDN Service to make our customers' transmission of their content to their end users more efficient.

Our customers make content available through their websites and their Internet-accessible (hosted) application programming interfaces (APIs). A customer can create content (customer-generated content), as can a customer's end users (user-generated content). Fastly's CDN Service then makes the transmission of that content (which we sometimes refer to as "content objects") more efficient by automatically storing copies at intermediate locations on a temporary basis. The process of storing these copies is known as "caching" and the server locations in which they are stored are referred to as "caches."

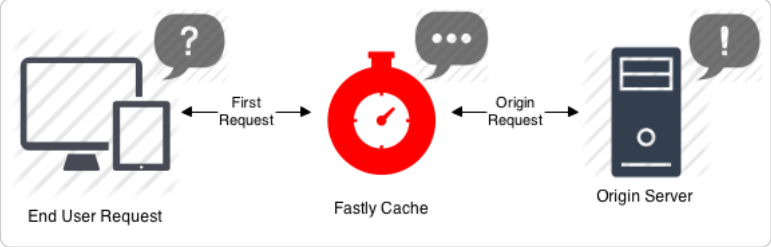


Fastly's delivers its CDN service from key access points to the Internet called "points of presence" (POPs). Fastly places POPs (https://www.fastly.com/blog/how-fastly-chooses-pop-locations) where their connectivity to the Internet reduces network transit time when delivering content to end-users. Each POP has a cluster of Fastly cache servers. When end users request a customer's content objects, Fastly delivers them from whichever of the cache locations are closest to each end user.

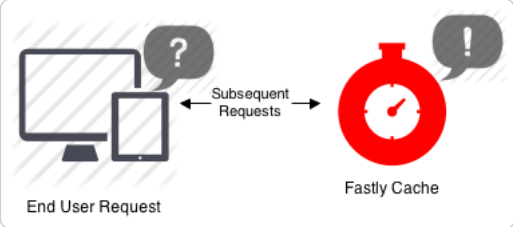
Fastly's caches only receive and process customers' end user requests for content objects. Customers decide which objects will be cached, for how long, who can access them, whether they are to be encrypted when transmitted over the Internet, and when the objects will be deleted from the caching service. Customers make these decisions by specifically configuring Fastly's CDN Service with these requirements. We refer to this configuration process as "provisioning."

To provision Fastly's CDN service (/guides/basic-setup/sign-up-and-create-your-first-service), customers must identify which of their application servers will provide the original content objects for each of their various domains (e.g., company.com, myco.com). Their application servers can be physical servers in a customer data center or hosting facility, or applications running on cloud services like Amazon, or any combination. Fastly refers to these source servers as "origin" and "backend" servers interchangeably.

The first time each Fastly cache receives a request for a content object, it fetches the object from the appropriate origin server. If multiple origin servers are specified, the cache will distribute the processing load for the fetches across all of them (based on the configuration criteria set by the customer). After the content object is fetched, the cache stores a copy of it and forwards its response to the end user.



Each time after the first time an end user requests that same content object, the Fastly cache fulfills requests by retrieving the cached copy from storage (or memory) and immediately delivering it to the end user – the fetch step to the original copy is not repeated until the content object either expires or becomes invalidated.



Can Fastly host my content?

We accelerate your site by caching both static assets and dynamic content by acting as a reverse proxy (https://en.wikipedia.org/wiki/Reverse_proxy) to your origin server (also known as "Origin Pull"), but we do not provide services for uploading your content to our servers.

In addition to using your own servers as the source, we also support various "cloud storage" services as your origin, such as Amazon Elastic Compute Cloud (/guides/integrations/amazon-s3) (EC2), Amazon Simple Storage Service (/guides/integrations/amazon-s3) (S3), and Google Cloud (/guides/integrations/google-cloud-storage) as your file origin. Our partnership with Google (https://www.fastly.com/partner/gcp) in particular enables us to have direct connectivity to their cloud infrastructure.

§ HTTP status codes cached by default (/guides/about-fastly-services/http-status-codes-cached-by-default)

Fastly caches the following response status codes by default. In addition to these statuses, you can force an object to cache under other states using conditions (/guides/conditions/) and responses (/guides/basic-configuration/responses-tutorial).

Code	Message
`200`	OK
`203`	Non-Authoritative Information
`300`	Multiple Choices
`301`	Moved Permanently
`302`	Moved Temporarily
`404`	Not Found
`410`	Gone

§ Self-provisioned Fastly services (/guides/about-fastly-services/self-provisioned-fastly-services)

Fastly customers configure or "provision" Fastly caching and video services personally, independent of Fastly staff, via the Fastly application interface (https://app.fastly.com). Fastly calls this "self-provisioning." Self-provisioning tasks include things like:

- creating and deploying services
- adding domains and origin servers
- configuring load balancing
- modifying how services handle HTTP headers
- submitting purge commands

Once provisioned, Fastly services can be deployed immediately. If self-provisioned tasks fail to operate in an appropriate or expected manner, you can find answers to a variety of frequently asked questions in Fastly's guides and tutorials (/guides/). You can also receive personalized assistance by submitting requests (/guides/customer-support/submitting-support-requests) directly to Fastly's Customer Support staff.

• Guides (/guides/) > Advanced setup (/guides/advanced) > Securing communications (/guides/securing-communications/)

§ Accessing Fastly's IP ranges (/guides/securing-communications/accessing-fastlys-ip-ranges)

To help you whitelist Fastly's services through your firewall, we provide access to the list of Fastly's assigned IP ranges. You can access the list via URL:

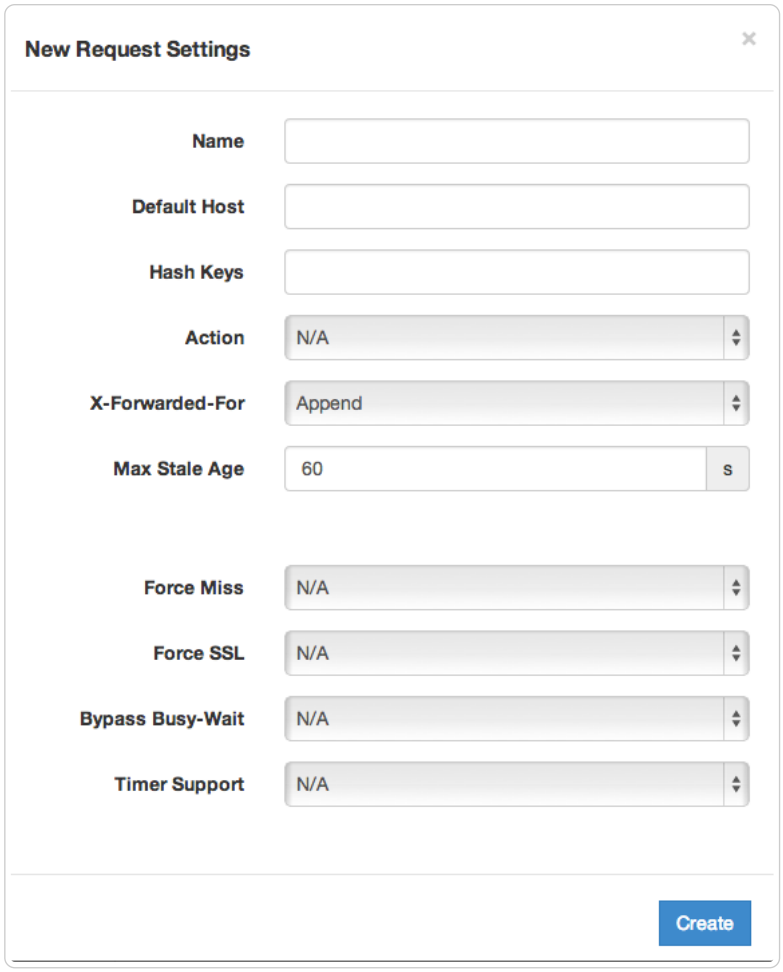
https://api.fastly.com/public-ip-list (https://api.fastly.com/public-ip-list)

You can then automate the API call (for example, by running a script (https://github.com/jondade/IP-Whitelist-cron) as a cron job) to request the list of IPs to detect when the IP ranges change.

§ Allowing only TLS connections to your site (/guides/securing-communications/allowing-only-tls-connections-to-your-site)

If you want to only allow TLS on your site, we have you covered. There is a switch built into the "Request Settings" that will allow you to force unencrypted requests over to TLS. It works by returning a **301 Moved Permanently** response to any unencrypted request, which redirects to the TLS equivalent. For instance, making a request for **http**://www.example.com/foo.jpeg would redirect to **https**://www.example.com/foo.jpeg.

1. Log in to the Fastly application and click the **configure** button (wrench icon).
2. From the **Service** menu, select the appropriate service and then click the blue **Configure** button (with the gear icon) to the right of the service name.
3. Click the **Settings** pane and find the **Request Settings** section.
4. Click the **New** button to the right of the **Request Settings** section. The New Request Settings window appears.



5. Type a name in the **Name** field. ('Default' works fine, if it's your only one.)
6. Select **Yes** from the **Force SSL** menu.
7. Click **Create** to save your request setting changes.

Once you deploy your changes, you're done!

For more information about TLS-related issues, see our TLS guides (/guides/securing-communications/) or contact support@fastly.com (mailto:support@fastly.com) with questions.

§ Connecting to origins over TLS (/guides/securing-communications/connecting-to-origins-over-tls)

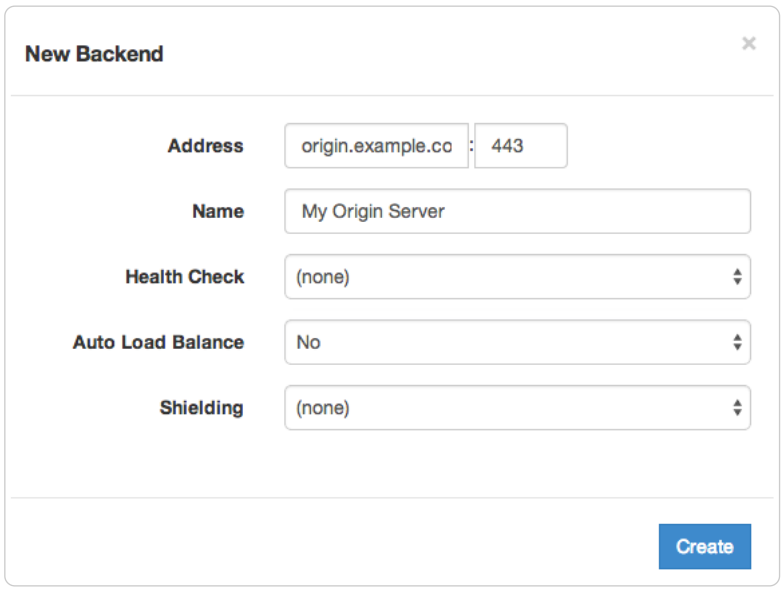
We offer two ways to use TLS to communicate to your origin servers.

The simple way

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.

5. Click the **Hosts** section.
6. In the **Backends** area, click the **New** button to create a new backend for your non-secure server. The New Backend window appears.



7. Fill out the **New Backend** window as follows:
   - In the **Address** field, type the address of your secure server (for example, `origin.example.com`).
   - In the **Port** field, type `443`.
   - In the **Name** field, type the name of your server (for example, `My Origin Server`).
   - Leave the **Health Check**, **Auto Load Balance**, **Weight**, and **Shielding** controls set to their default values.
8. Click the **Create** button. The server appears in the Backends area.
9. If there are multiple backend servers in your service, be sure to specify a default host (/guides/basic-configuration/specifying-a-default-host).

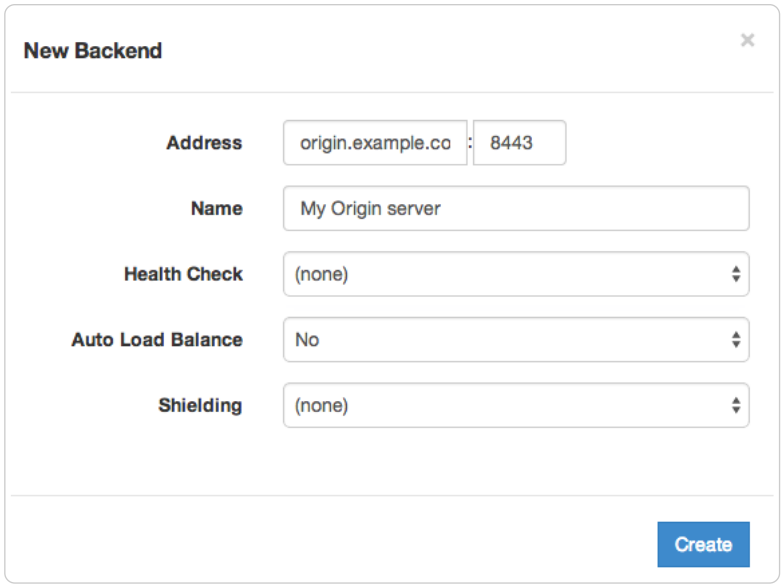
IMPORTANT: For port 443, and only port 443, we will enable TLS automatically in our system. No other configuration is needed and using this port often can help avoid triggering TLS redirect loops. If you find that setting the port specifically to 443 doesn't produce the results you're hoping for or simply isn't working, consider trying the more complicated settings detailed below.

The detailed way

To use a different port, or for a little more control over the TLS connections, follow these steps.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Hosts** section.
6. In the **Backends** area, click the **New** button to create a new backend for your non-secure server. The New Backend window appears.



7. Fill out the **New Backend** window as follows:
   - In the **Address** field, type the address of your secure server (for example, `origin.example.com`).
   - In the **Port** field, type a port number. For example, `8443`.
   - In the **Name** field, type the name of your server (for example, `My Origin Server`).
   - Leave the **Health Check**, **Auto Load Balance**, **Weight**, and **Shielding**. controls set to their default values.
8. Click the **Create** button. The server appears in the Backends area.
9. Click the gear icon next to the Backend you just created.
10. From the menu that appears, select **TLS Options**. The TLS Options window appears.
11. From the **Use TLS for Connection** menu, select **Yes**.
12. From the **Verify Certificate** menu, select **Yes**.

    ⚠ **WARNING:** Selecting No from the Verify Certificate menu means the certificate will not be verified. Not verifying the certificate has serious security implications, including vulnerability to man-in-the-middle attack (https://www.owasp.org/index.php/Man-in-the-middle_attack). Consider uploading a CA certificate instead of disabling certificate validation.

13. Click the **Update** button.
14. If there are multiple backend servers in your service, be sure to specify a default host (/guides/basic-configuration/specifying-a-default-host).
15. Activate a new version of the service to complete the configuration changes.

And that's all you really need to do. Everything else is optional, but just in case you'd like to set them, we've included the information below.

Setting the TLS hostname

Normally we check the server certificate against the hostname portion of the address for your origin entered in the **New Backend** window. To have Fastly verify the certificate using a different hostname, specify it via the **Certificate Hostname** and **SNI Hostname** fields in the **TLS Options** window.

This information also gets sent to the server in the TLS handshake. If you are using Server Name Indication (https://en.wikipedia.org/wiki/Server_Name_Indication) (SNI) to put multiple certificates on your origin, specifying it in the **SNI Hostname** field will select which one is used.

Specifying a TLS CA certificate

If you're using a certificate that is either self-signed or signed by a Certificate Authority (CA) not commonly recognized by major browsers (and unlikely to be in the Ubuntu bundle that we use), you can provide the certificate in PEM format via the **TLS CA Certificate** field in the **TLS Options** window. The PEM format looks like this:

```
-----BEGIN CERTIFICATE-----
MIIDrzCCApegAwIBAgIQCDvgVpBCRrGhdWrJWZHHSjANBgkqhkiG9w0BAQUFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBD
QTAeFw0wNjExMTAwMDAwMDBaFw0zMTExMTAwMDAwMDBaMGExCzAJBgNVBAYTAlVT
MRUwEwYDVQQKEwxEaWdpQ2VydCBJbmMxGTAXBgNVBAsTEHd3dy5kaWdpY2VydC5j
b20xIDAeBgNVBAMTF0RpZ2lDZXJ0IEdsb2JhbCBSb290IENBMIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4jvhEXLeqKTTo1eqUKKPC3eQyaKl7hLOllsB
CSDMAZOnTjC3U/dDxGkAV53ijSLdhwZAAIEJzs4bg7/fzTtxRuLWZscFs3YnFo97
nh6Vfe63SKMI2tavegw5BmV/Sl0fvBf4q77uKNd0f3p4mVmFaG5cIzJLv07A6Fpt
43C/dxC//AH2hdmoRBBYMql1GNXRor5H4idq9Joz+EkIYIvUX7Q6hL+hqkpMfT7P
T19sdl6gSzeRntwi5m3OFBqOasv+zbMUZBfHWymeMr/y7vrTC0LUq7dBMtoM1O/4
gdW7jVg/tRvoSSiicNoxBN33shbyTApOB6jtSj1etX+jkMOvJwIDAQABo2MwYTAO
BgNVHQ8BAf8EBAMCAYYwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUA95QNVbR
TLtm8KPiGxvDl7I90VUwHwYDVR0jBBgwFoAUA95QNVbRTLtm8KPiGxvDl7I90VUw
DQYJKoZIhvcNAQEFBQADggEBAMucN6pIExIK+t1EnE9SsPTfrgT1eXkIoyQY/Esr
hMAtudXH/vTBH1jLuG2cenTnmCmrEbXjcKChzUyImZOMkXDiqw8cvpOp/2PV5Adg
06O/nVsJ8dWO41P0jmP6P6fbtGbfYmbW0W5BjfIttep3Sp+dWOIrWcBAI+0tKIJF
PnlUkiaY4IBIqDfv8NZ5YBberOgOzW6sRBc4L0na4UU+Krk2U886UAb3LujEV0ls
YSEY1QSteDwsOoBrp+uvFRTp2InBuThs4pFsiv9kuXclVzDAGySj4dzp30d8tbQk
CAUw7C29C79Fv1C5qfPrmAESrciIxpg0X40KPMbp1ZWVbd4=
-----END CERTIFICATE-----
```

Specifying acceptable TLS protocol versions

If your origin server is configured with support for modern TLS protocol versions, you can customize the TLS protocols Fastly will use to connect to it by setting a **Minimum TLS Version** and **Maximum TLS Version** in the via the **TLS Options** window. We recommend setting both to the most up-to-date TLS protocol, currently 1.2, if your origin can support it.

Use the `openssl` command to verify your origin supports a given TLS protocol version. For example:

```
openssl s_client -connect origin.example.com:443 -tls1_2
```

Replace `-tls1_2` with `tls1_1` and `tls1_0` to test other protocol versions. Fastly does not support SSLv2 or SSLv3.

Specifying acceptable TLS cipher suites

Fastly supports configuring the OpenSSL cipher suites used when connecting to your origin server. This allows you to turn specific cipher suites on or off based on security properties and origin server support. The **Allowed Ciphersuites** setting in the TLS Options window accepts an OpenSSL formatted cipher list (https://www.openssl.org/docs/manmaster/apps/ciphers.html). We recommend using the strongest cipher suite your origin will support as detailed by the Mozilla SSL Configuration Generator (https://mozilla.github.io/server-side-tls/ssl-config-generator/).

Use the `openssl` command to verify your origin supports a given cipher suite. For example:

```
openssl s_client -connect origin.example.com:443 -tls1_2 -cipher ECDHE-RSA-AES128-GCM-SHA256
```

Replace `-cipher ECDHE-RSA-AES128-GCM-SHA256` with the cipher suite to test.

Specifying a TLS client certificate and key

To ensure TLS connections to your origin come from Fastly and aren't random, anonymous requests, set your origin to verify the client using a client certificate. Simply paste the certificate and private key in PEM form into the appropriate text boxes on the **TLS Options** window.

> **IMPORTANT:** The private key must not be encrypted with a passphrase.

Then configure your backend to require client certificates and verify them against the CA cert they were signed with. Here are some ways of doing that:

- Apache (http://httpd.apache.org/docs/2.4/ssl/ssl_howto.html#accesscontrol)
- Nginx (http://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_client_certificate)
- IIS (https://ondrej.wordpress.com/2010/01/24/iis-7-and-client-certificates/)

---

§ Domain validation for TLS certificates (/guides/securing-communications/domain-validation-for-tls-certificates)

When you purchase our Shared Certificate (/guides/securing-communications/ordering-a-paid-tls-option#shared-certificate) or Shared Wildcard Certificate (/guides/securing-communications/ordering-a-paid-tls-option#shared-wildcard-certificate-service) TLS options, our partner Certificate Authority (GlobalSign) must verify you control the domains requested and that you authorize us to request a certificate service on your behalf. You can choose:

- DNS text record verification (preferred)
- URL verification

- Email verification

Regardless of the verification method you use, be sure to follow our instructions (/guides/securing-communications/ordering-a-paid-tls-option) to begin the TLS ordering process.

DNS text record verification

We provide you with a unique DNS TXT record that you need to add for the zone origin ("@"). The text of this entry will change depending on the certificate to which your domain is added, but will generally look something like this (where the `{meta tag}` will change depending on the certificate):

```
@ IN TXT "globalsign-domain-verification={meta tag}"
```

Consult the documentation for your DNS server or hosted DNS provider for more information about how to add the record. This text record must be wholly separate from other text records. A prepended, inserted, or appended record will not work.

URL verification

We provide you with an HTML meta tag that must be included in the `<head>` section of the web page served at the root of the domain to be added. The meta tag will be formatted similar to the following (where the `{meta tag}` text will change depending on the certificate):

```
<meta name="globalsign-domain-verification" content="{meta tag}" />
```

Note that this text must be served from the actual requested domain or root domain. For example, GlobalSign will specifically query `http://www.example.com`, so the verification tag must be served from whatever resource is returned from that URL. GlobalSign will not follow redirects or request a file on that domain (such as `http://www.example.com/verify.html` or `http://www.example.com/index.html`). It will also work on `http://example.com`.

Email verification

GlobalSign will give Fastly a list of acceptable email addresses to which they can send a validation email. Generally these email addresses match those that appear on the WHOIS record of the domain requested, plus the following:

- admin@domain.com
- administrator@domain.com
- hostmaster@domain.com
- postmaster@domain.com
- webmaster@domain.com

For entries requested for a subdomain, each of those addresses `@subdomain.domain.com` will also work (e.g., `admin@subdomain.domain.com`).

We will send you the list of acceptable email address. You will need to tell us which email address to use. GlobalSign will then send a verification email to the email address you specify. Once you receive the verification email, you will need to click on a link in that email and follow the instructions to complete the validation.

---

§ Enabling HSTS through Fastly (/guides/securing-communications/enabling-hsts-through-fastly)

The HTTP Strict Transport Security (https://tools.ietf.org/html/rfc6797) (HSTS) security enhancement specification provides a way to force modern browsers to communicate only via the Transport Layer Security (TLS) protocol. Once enabled, it will force the browser to redirect (typically with a status code 307) to the HTTPS URL.

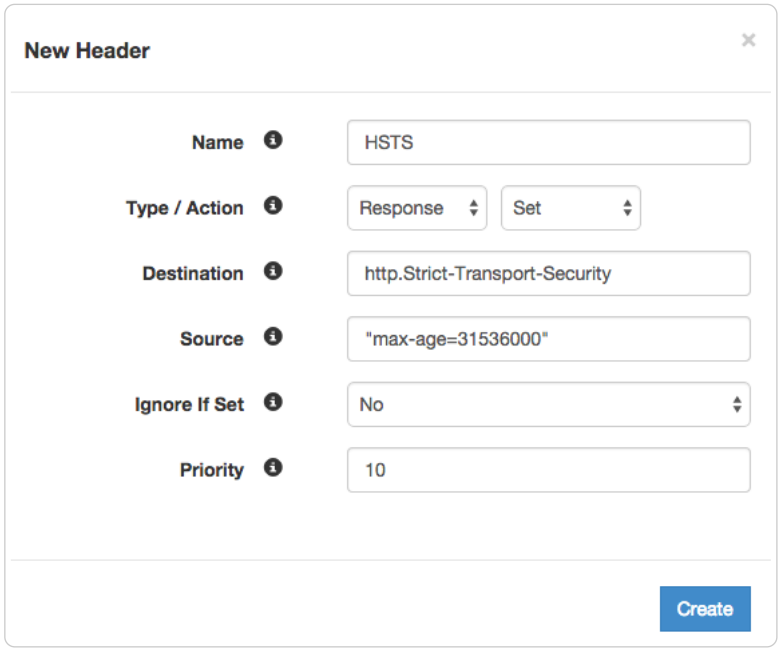
> ℹ **NOTE:** HSTS only takes effect *after* a site has been visited on a trusted HTTPS connection. It doesn't replace the need to have redirects from your HTTP site.

Enabling HSTS

To enable HSTS, add a new header (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses) as follows:

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Content** section.
6. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.



7. Fill out the **New Header** window as follows:
   - In the **Name** field, type an appropriate name, such as `HSTS`.
   - From the **Type/Action** menus, select **Response** and **Set**.
   - In the **Destination** field type `http.Strict-Transport-Security`.
   - In the **Source** field type `"max-age=<max age in seconds>"`. For example, `"max-age=31536000"`. As described below, `max-age` is required and two additional HSTS options can be specified.
   - Leave the **Ignore if Set** menu and the **Priority** field set to their defaults (or set them as appropriate for your service).
8. Click the **Create** button. A new header appears in the Headers area of the Content section.

HSTS options

HSTS requires the max-age directive (https://tools.ietf.org/html/rfc6797#section-6.1.1) be set in order to function properly. It specifies how long in seconds to remember that the current domain should only be contacted over HTTPS. The example shown above sets `max-age` to one year (31536000 seconds = 1 year). You may want to experiment using a smaller value than what is shown.

Two additional options can be specified with the HSTS response header:

- `includeSubdomains` - This token applies HSTS to all of your site's subdomains. Before you include it, be certain none of your subdomains require functionality on HTTP in a browser. Ensure your TLS certificate is a wildcard or has coverage for all subdomain possibilities.

  IMPORTANT: All subdomains will be unreachable on HTTP by browsers that have seen the HSTS header once `includeSubdomains` is enabled.

- `preload` - This token allows you to submit your domain for inclusion in a preloaded HSTS list that is built into several major browsers. Although the token is not part of the HSTS specification, including it in the header is a prerequisite for submitting to this preloaded list.

  WARNING: Don't request browser preload inclusion unless you're sure that you can support HTTPS for the long term. Inclusion in the

HSTS Preload List cannot be undone easily. See https://hstspreload.appspot.com/ (https://hstspreload.appspot.com/) for submission instructions and more information.

Combining all of these options together in the **Source** field would look like this:

```
"Strict-Transport-Security: max-age=<max age in seconds>; includeSubDomains; preload"
```

To disable HSTS for whatever reason, simply set the `max-age` to `0` on an HTTPS connection.

The HSTS Preload List is managed by a third party, not by Fastly. See https://hstspreload.appspot.com/ (https://hstspreload.appspot.com/) for more information.

Additional reading

- RFC 6797 (http://tools.ietf.org/html/rfc6797), which describes the HSTS specification
- the Wikipedia description (https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security) of HSTS, including the currently known limitations and a browser support list
- the OWASP.org explanation (https://www.owasp.org/index.php/HTTP_Strict_Transport_Security) of HSTS, including descriptions of the threats it addresses
- the Chromium Projects description (https://www.chromium.org/hsts) of HSTS and preloading HSTS sites

§ Ordering a paid TLS option (/guides/securing-communications/ordering-a-paid-tls-option)

The Fastly Transport Layer Security (TLS) service provides privacy and data security for your network. You can request different TLS service options for your account. For specific details about pricing for each of Fastly's TLS options, see our pricing page (https://www.fastly.com/pricing) or contact sales@fastly.com (mailto:sales@fastly.com) for our current rates.

To order TLS service, open a ticket via the support tab or email support@fastly.com (mailto:support@fastly.com). Remember to put "TLS Certificate Request" in the subject of the request and let us know which service option you're interested in. We'll walk you through the setup process for the option you choose.

Fastly has four domain service options.

Shared domain

This free option uses the Fastly SAN certificate's wildcard entry for `global.ssl.fastly.net`. To use this service, add a new domain in the Fastly application and set up an origin server for that domain. You can learn more about how to do that in our guide on setting up free TLS (/guides/securing-communications/setting-up-free-tls).

Shared certificate

This service option uses the Fastly SAN certificate. You can read about SAN certificates here (https://www.globalsign.com/en/ssl/multi-domain-ssl/). Specifically:

- You get to use your domain, but Fastly does the certificate administration.
- You provide your domain name list to Fastly and we add those names to the Certificate SAN field.

Our registrar explains the shared (SAN) certificate (https://www.globalsign.com/en/ssl/multi-domain-ssl/) as "a way to conserve IP addresses by putting multiple hostnames or domains on one certificate. There are no security implications... Addition of your name to the certificate still needs to be authorized by you."

Shared wildcard certificate service

This service option uses the Fastly SAN certificate. You can read about SAN certificates here (https://www.globalsign.com/en/ssl/multi-domain-ssl/). Specifically:

- You get to use your domain, but Fastly does the certificate administration.
- You provide one or more wildcard domain name entries to Fastly and we add those names to the Certificate SAN field.

Customer certificate hosting

This service option uses your TLS certificates, which includes Extended Verification (EV) certificates. Specifically:

- You provide the certificates to Fastly.

- Fastly installs those certificates into the caches and allocates IP addresses on each cache.
- Fastly creates a new (customer-specific) DNS Global Domain Map that associates the certificate with the allocated IP addresses.
- Fastly maintains the domain map and the certificate on the caches.

IMPORTANT: Fastly supports SHA-256 certificates signed by publicly trusted certificate authorities that have a minimum key size of 2048 bits for RSA or DSA, and 256 bits for ECDSA. For performance reasons, we strongly recommend using a 2048-bit key size for RSA and DSA when larger key sizes are not required for your application.

SNI customer certificate hosting

Fastly provides, on a limited availability basis, customer certificate hosting that uses Server Name Indication (SNI) for certificate selection. To see if your company meets our SNI certificate hosting qualification criteria, contact sales@fastly.com (mailto:sales@fastly.com). For SNI customer certificate hosting:

- You provide the certificates to Fastly.
- Fastly hosts your certificate on a shared domain map and uses SNI to select the correct certificate to serve based on the domain name sent.

IMPORTANT: All modern browsers support SNI. Clients that do not support SNI (such as those on Windows XP and Android 2.x or earlier) will see a certificate error.

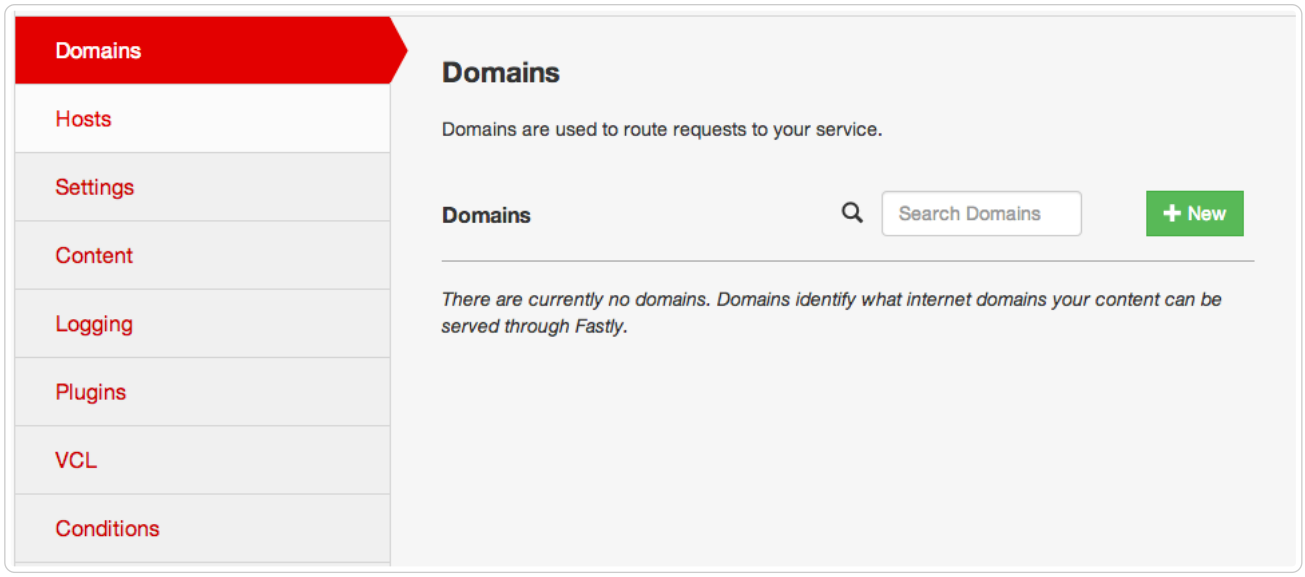
NOTE: Any certificates provided by GlobalSign are subject to the terms of GlobalSign's Subscriber Agreement, which can be found at https://www.globalsign.com/repository/ (https://www.globalsign.com/en/repository/).

§ Setting up free TLS (/guides/securing-communications/setting-up-free-tls)

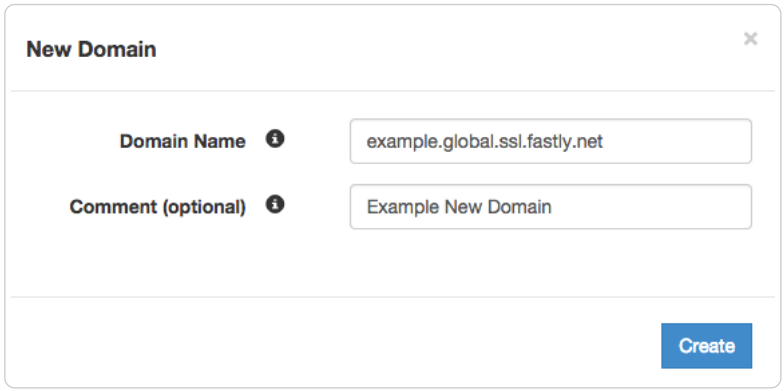
Customers can use our shared domain TLS wildcard certificate for free. Follow the steps below to set up free TLS:

1. Log in to the Fastly application and click the **configure** tab (wrench icon).

2. From the **Service** menu, select the appropriate service, and then click the blue **Configure** button. The main controls for the selected service appear.
3. Click **Domains** from the section list on the left. The Domains window appears.



4. Click the **New** button. The New Domain pane appears.



5. In the **Domain Name** field, type `[name].global.ssl.fastly.net`, where `[name]` is a single word. If the name has already been taken, you will need to pick a different one.

> **IMPORTANT:** `[name]` can only be a single word. You cannot use a dot-separated name such as `www.example.com.global.ssl.fastly.net` because nesting in TLS certificates is not supported.

6. Click **Create** to save the domain. The new domain appears in the list of domains.
7. Deploy your service.

You should be able to access the domain via the following URL: `https://<name>.global.ssl.fastly.net/`. Note that you don't need to add CNAME records (/guides/basic-setup/adding-cname-records) to use the shared domain certificate.

> **IMPORTANT:** If you DNS alias your own domain (`www.example.com`) to that name (`example.global.ssl.fastly.net`) a TLS name mismatch warning will appear in the browser. The only way to fix the mismatch is to add your domain to our TLS certificate. For more information, see our TLS options guide (/guides/securing-communications/ordering-a-paid-tls-option).

§ Support for App Transport Security (/guides/securing-communications/support-for-app-transport-security)

Apple uses App Transport Security (https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW35) (ATS) to improve the security of connections between web services and applications installed on devices using iOS 9 or later, and OS X 10.11 (El Capitan) and later. Fastly is fully compliant with all ATS requirements. You shouldn't run into any issues supporting iOS or OS X users while using our service.

Results from the ATS diagnostics tool

We used Apple's ATS diagnostics tool to ensure that Fastly is compliant with all ATS requirements. You can review the output from the diagnostics tool below.

```
$ /usr/bin/nscurl --ats-diagnostics https://www.fastly.com
Starting ATS Diagnostics

Configuring ATS Info.plist keys and displaying the result of HTTPS loads to https://www.fastly.com.
A test will "PASS" if URLSession:task:didCompleteWithError: returns a nil error.

Use '--verbose' to view the ATS dictionaries used and to display the error received in URLSession:task:didCompleteWithError:.
================================================================================

Default ATS Secure Connection
---
ATS Default Connection

Result : PASS
---

================================================================================
Allowing Arbitrary Loads

---
Allow All Loads

Result : PASS
---

================================================================================

Configuring TLS exceptions for www.fastly.com

---
TLSv1.2

Result : PASS
---

---
TLSv1.1

Result : PASS
---

---
TLSv1.0

Result : PASS
---

================================================================================

Configuring PFS exceptions for www.fastly.com

---
Disabling Perfect Forward Secrecy

Result : PASS
---

================================================================================

Configuring PFS exceptions and allowing insecure HTTP for www.fastly.com

---
Disabling Perfect Forward Secrecy and Allowing Insecure HTTP

Result : PASS
---

================================================================================

Configuring TLS exceptions with PFS disabled for www.fastly.com

---
TLSv1.2 with PFS disabled

Result : PASS
---
```

```
---
TLSv1.1 with PFS disabled

Result : PASS
---

---
TLSv1.0 with PFS disabled

Result : PASS
---

================================================================================

Configuring TLS exceptions with PFS disabled and insecure HTTP allowed for www.fastly.com

---
TLSv1.2 with PFS disabled and insecure HTTP allowed

Result : PASS
---

---
TLSv1.1 with PFS disabled and insecure HTTP allowed

Result : PASS
---

---
TLSv1.0 with PFS disabled and insecure HTTP allowed

Result : PASS
---
```

§ TLS termination (/guides/securing-communications/tls-termination)

Identifying TLS terminated requests

To maintain optimal caching performance, Fastly uses a TLS terminator (https://en.wikipedia.org/wiki/TLS_termination_proxy) separate from the caching engine. This means, however, that the caching engine doesn't know that it was originally a TLS request. As a result, we set the `Fastly-SSL` header when fetching the content from your servers.

Because we set this header, you can check for its presence on your backend by doing something like:

```
if (req.http.Fastly-SSL) {
  set resp.http.X-Is-SSL = "yes";
}
```

and that should tell you if the request was a TLS request or not.

When using WordPress

If you're using Fastly TLS services with WordPress, you'll want to add a check for the `HTTP_FASTLY_SSL` header so that WordPress can build URLs to your CSS or JS assets correctly. Do this by placing a check in your `wp-config.php` file to override the SSL flag that is checked later:

```
if (!empty( $_SERVER['HTTP_FASTLY_SSL'])) {
  $_SERVER['HTTPS'] = 'on';
}
```

As usual, this must be placed anywhere before the `require_once` line with `wp-settings.php`.

Finding the original IP when using TLS termination

Because Fastly uses a TLS terminator (https://en.wikipedia.org/wiki/TLS_termination_proxy), separate from the caching engine for performance, the engine overwrites the original IP briefly due to the re-request to your origin servers once decrypted and causes anything that references the original IP to show up as 127.0.0.0/8 IPs. To find the original IP via VCL:

- use `req.http.Fastly-Client-IP` if you're using shielding

- use `client.ip` you're not using shielding or if you're building an ACL

Fastly also sends along the client IP to the origin in a HTTP header, `Fastly-Client-IP`, which can be used by server software to adjust as needed.

For more information about TLS-related issues, see our TLS guides (/guides/securing-communications/) or contact support@fastly.com (mailto:support@fastly.com) with questions.

• Guides (/guides/) > Advanced setup (/guides/advanced) > Purging (/guides/purging/)

§ Authenticating URL purge requests via API (/guides/purging/authenticating-api-purge-requests)

Fastly's URL purge (/guides/purging/single-purges.html#purging-a-url) feature allows you to purge individual URLs on your website. By default, authentication is not required to purge a URL with the Fastly API, but you can enable API key authentication in the Fastly application by adding a header or by using custom VCL.

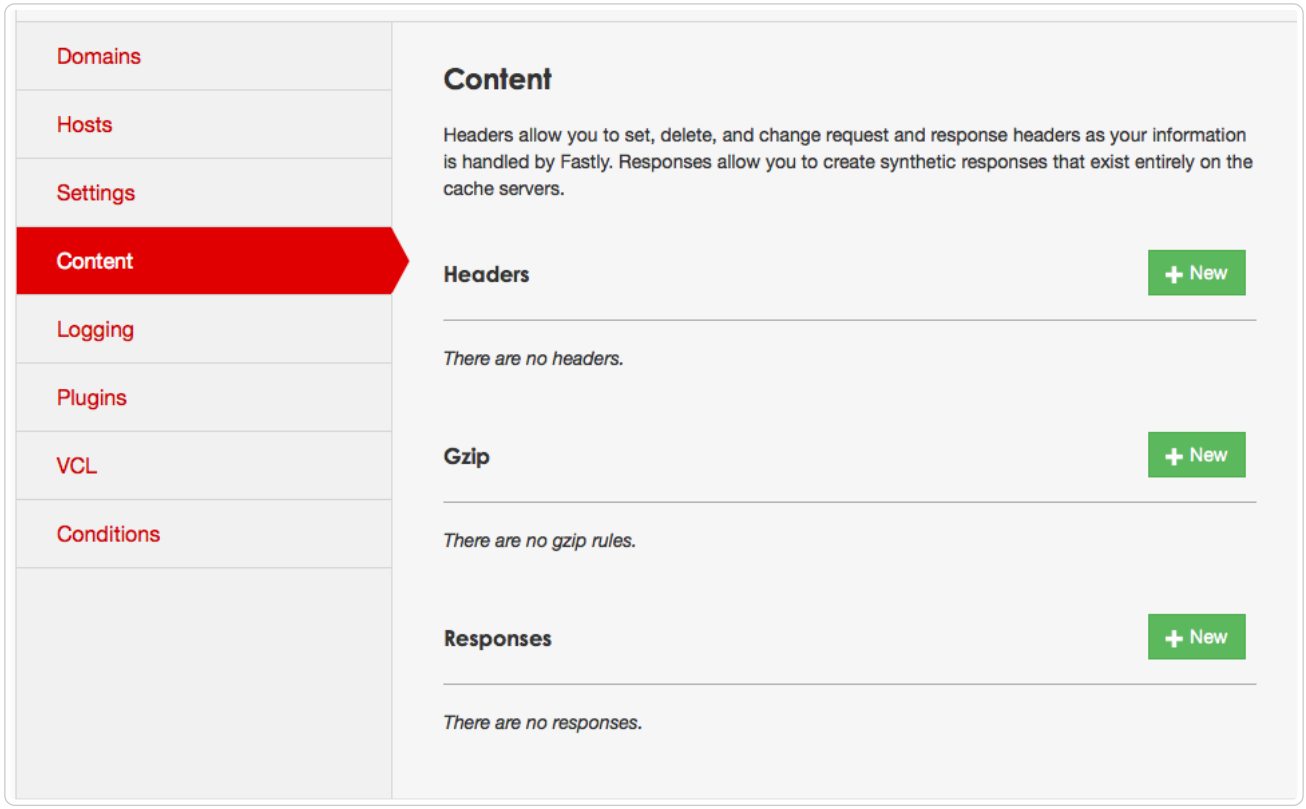
NOTE: All purge requests other than URL purges require authentication by default, as indicated in the API documentation (/api/purge#purge).

Enabling authentication in the Fastly Application

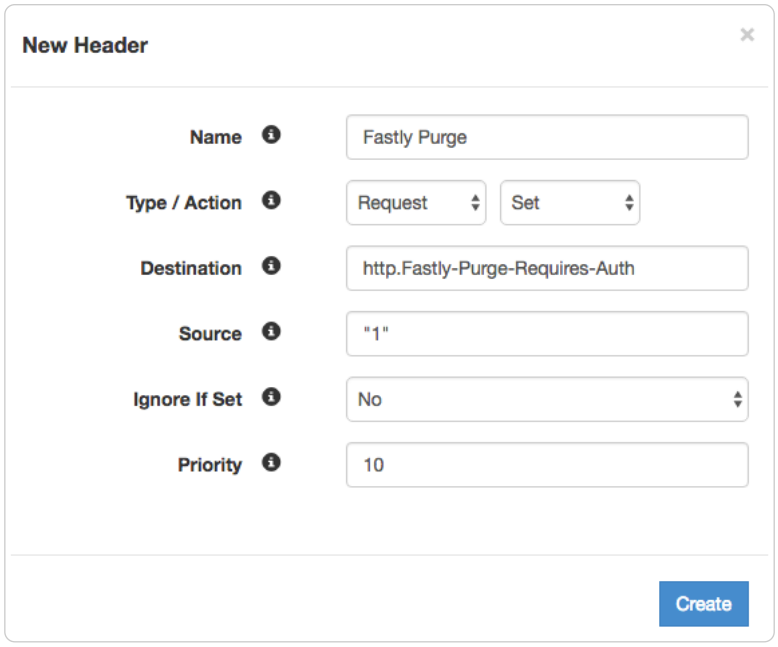
You can enable API key authentication for URL purge requests by adding a header and optionally attaching a condition in the Fastly application.

Adding the header

1. Log in to the Fastly application.
2. Click **configure** (the wrench icon at the top of the window).
3. From the **Services** menu, select the appropriate service.
4. Click the blue **Configure** button.
5. Click the **Content** pane from the list on the left.



6. In the **Headers** area, click **New**. The New Header window appears.

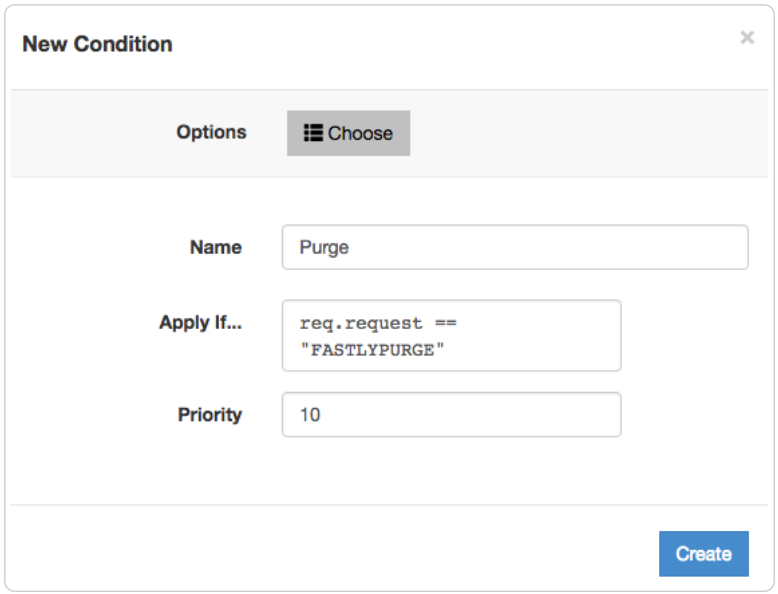


7. Fill out the **New Header** window as follows:
   - In the **Name** field, type the name of your header rule (for example, `Fastly Purge`).
   - From the **Type** menu, select **Request**, and from the **Action** menu, select **Set**.
   - In the **Destination** field, type `http.Fastly-Purge-Requires-Auth`.
   - In the **Source** field, type `"1"`.
   - From the **Ignore If Set** menu, select **No**.
   - In the **Priority** field, type `10`.
8. Click **Create**.

Attaching a condition

Attaching the following condition is optional. Without the condition, the header you just created will be added to all requests. With the condition, the header will be added to purge requests only.

1. In the **Headers** area, click the gear icon next to the new header you just created, and select **Request Conditions**.
2. Click the **New** button to create a new condition. The New Condition window appears.



3. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a descriptive name for the new condition (for example, `Purge`).

- In the **Apply If** field, type `req.request == "FASTLYPURGE"`.
- In the **Priority** field, type `10`.

4. Click **Create**.
5. Activate the new version of your service.

Enabling authentication with custom VCL

If you'd rather enable API key authentication for URL purge requests using custom VCL (/guides/vcl/uploading-custom-vcl), add the following to your VCL file:

```
if (req.request == "FASTLYPURGE") {
  set req.http.Fastly-Purge-Requires-Auth = "1";
}
```

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Purging URLs with an API key

After you've enabled API key authentication for URL purge requests, you'll need to provide your API key (/guides/account-management-and-security/finding-and-managing-your-account-info#finding-and-regenerating-your-api-key) in the URL purge API request (/api/purge#purge_3aa1d66ee81dbfed0b03deed0fa16a9a):

```
curl -X PURGE -H Fastly-Key:$FASTLY_KEY https://www.example.com/
```

which would return this response:

```
{"status": "ok", "id": "1234567890"}
```

WARNING: If your website is not configured to use HTTPS, you should use a POST request with Fastly's URL to perform the purge. The request will look like this: `curl -X POST -H Fastly-Key:$FASTLY_KEY https://api.fastly.com/purge/<your_url_here>`.

---

§ Getting started with surrogate keys (/guides/purging/getting-started-with-surrogate-keys)

Efficient cache invalidation is an essential part of keeping your website fast. Purging too much cache using purge all (/guides/purging/single-purges#purging-all-content) may increase your website's load time while the cache rebuilds. If you find yourself purging all cache on more than a weekly basis, consider using surrogate keys for more targeted purging.

Surrogate keys allow you to selectively purge related content. Using the `Surrogate-Key` header, you can tag a group of objects with a key and then use it to purge multiple pieces of content at once. This process can occur automatically within your application, making it easier to cache and purge content that changes rapidly and unpredictably.

This guide consists of four parts:

- Understanding surrogate keys
- Looking at a practical example
- Generating and setting surrogate keys
- Limitations

NOTE: This guide assumes you're already familiar with the way content delivery networks (CDNs) work (/guides/about-fastly-services/how-caching-and-cdns-work) in general, and the way Fastly's CDN works (/guides/about-fastly-services/how-fastlys-cdn-service-works) in particular.

Understanding surrogate keys

After you've signed up for Fastly and added one or more services (/guides/basic-setup/working-with-services#creating-a-new-service), you can start examining how your origin server responds to requests. When your origin server responds to an HTTP request for content, it's because Fastly hasn't yet cached that content or the cache has expired. Your server's response to the request will resemble the example shown below. (Note that you can use the curl command (/guides/debugging/curl-and-other-caching-verification-methods) to inspect any of your server's responses.)

```
HTTP/1.1 200 OK
Content-Type: text/html
Connection: keep-alive
...
```

To control how your content is served to users and cached by Fastly, you can add to or modify the headers (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses) that are included in your origin server's response. The `Surrogate-Key` header is one of the headers that you can add to the response. It allows you to "tag" an object, such as an image or a blog post, with one or more keys. When the object changes, you can reference the key in a purge request (/guides/purging/single-purges) to remove the object from the cache.

You can add space-delimited strings to the `Surrogate-Key` header, like this:

```
HTTP/1.1 200 OK
Surrogate-Key: key1 key2 key3
Content-Type: text/html
...
```

This response contains three surrogate keys: `key1`, `key2`, and `key3`. When Fastly receives a response like this, we use the surrogate keys to create a mapping from each key to the cached content, then we strip out the `Surrogate-Key` header so it's not included in the response to your readers.

Creating relationships between keys and objects

One of the major advantages of surrogate keys is that they allow for a many-to-many relationship between keys and objects. An origin server's response can associate multiple keys with the object, and the same key can be provided in different responses. Take a look at these two requests and responses:

```
GET /blog/ HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK Content-Type: text/html
Content-Length: 1234
Surrogate-Key: mainpage template-a
```

```
GET /blog/article/fastly-rocks HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 2345
Surrogate-Key: template-a article-fastly-rocks
```

In this example, there are two objects (`/blog` and `/blog/article/fastly-rocks`) with three keys (`mainpage`, `template-a`, and `article-fastly-rocks`). Two of the keys (`mainpage` and `article-fastly-rocks`) are associated with a single object, and a third key (`template-a`) is associated with both objects.

Purging objects with surrogate keys

By using the `Surrogate-Key` header to associate keys with one or more objects, you can precisely control which objects are removed from cache during a purge. Consider the example presented above. Purging the `mainpage` key would remove only the `/blog` object from the cache. On the other hand, purging the `template-a` key would remove both the `/blog` and `/blog/article/fastly-rocks` objects from the cache.

You can use the Fastly application to manually purge objects via key (/guides/purging/single-purges#purging-with-a-key), or you can use our Purge API (/api/purge) to purge a collection of objects with one API call. If you're using Fastly to cache your API, check out the guide on purging API cache with surrogate keys (/guides/api-caching/purging-api-cache-with-surrogate-keys) to learn how surrogate keys can help you purge API cache.

Looking at a practical example

Let's look at a practical example to learn how surrogate keys work. Imagine you're building a picture-hosting website and you're using Fastly to speed it up. You initially decide to cache picture pages by their URL (e.g., `http://www.example.com/pic/id`). When a picture's information changes, you'll remove the old version by sending a purge request to the Fastly API:

```
PURGE /pic/{id}
Host: example.com
Accept: */*
```

But there's a potential problem with this solution. You display a user's information next to their pictures, so you'll need to purge all of the user's picture pages if they change their information. You could send an individual purge for each picture, but that would take too long. As an alternative, you could cache the pages for a very short amount of time, but that would waste our server resources.

Surrogate keys solve this problem. By adding a surrogate key to all of a user's picture pages (e.g., `/user/542`, `/user/25`), you can purge all of the user's pictures by sending Fastly a purge for the user's surrogate key when they update their information. Now, instead of having to purge each picture individually, you can update them all with just one request:

```
PURGE /service/id/purge/user/542
...
```

Purging multiple sites at the same time

What if you wanted to build a mobile version of the picture-hosting website to complement the desktop version? You'll need a way to purge both the desktop version and the mobile version at the same time. Surrogate keys can help in this instance. You can tag the different versions of a picture page with the same surrogate key (e.g., `pic/76`, `pic/345`) and purge them all at once. All of the related content on our sites can now be purged with one request.

Tagging templates with surrogate keys

Surrogate keys come in really handy when making changes to templates. Imagine you have to make a change to the banner of the website. Since you're caching entire page, updating the header template isn't enough. You'll also need to purge all the pages that use the template. You could purge every page on the website, but there's no reason to purge content that doesn't use the header template.

You can make things easy by using surrogate keys. By adding surrogate keys for each template on a page (e.g., `/templates/pic/show`, `/templates/pic/header`, `/templates/pic/comment`), you can check which templates have changed and purge only pages with modified templates.

Generating and setting surrogate keys

There are two ways to set the `Surrogate-Key` header: by adding the header in the Fastly application, or by generating the keys with your own application. We describe how to use the Fastly application in our guide to generating Surrogate-Key headers based on URLs (/guides/purging/setting-surrogate-key-headers-based-on-a-url) (we have a separate guide for Amazon S3 origins (/guides/purging/setting-surrogate-key-headers-for-amazon-s3-origins)).

Automatically generating keys with your own application is described below using Fastly's Test Blog (https://www.fastly.com/blog/introducing-our-open-source-app-training) as an example. The test blog is a Ruby on Rails application that comes preloaded with example content that can be cached and purged using Fastly via the fastly-rails (https://github.com/fastly/fastly-rails) Ruby gem. (We also have other API clients (/api/clients) that support surrogate keys.)

NOTE: You can install the Fastly Test Blog and recreate this example yourself to practice generating surrogate keys. For the purposes of this example, we've made one deviation from the Fastly Test Blog instructions outlined on the GitHub page (https://github.com/fastly/fastly-test-blog). We've added a CNAME record (/guides/basic-setup/adding-cname-records) for our test blog to create two URLs — http://origin.example.com, which is the URL of the origin server not attached to our Fastly service, and http://fastly.example.com, which is the URL being cached by Fastly. Having the two URLs available will be useful when we examine the headers.

Configuring the API client

The fastly-rails (https://github.com/fastly/fastly-rails) gem provides a `set_surrogate_key_header` method (https://github.com/fastly/fastly-rails#headers) which the test blog uses to automatically generate surrogate keys for new articles. You can see how this works by examining the code in articles_controller.rb (https://github.com/fastly/fastly-test-blog/blob/master/app/controllers/articles_controller.rb). A slightly modified excerpt of the code from the controller is shown below.

```
class ArticlesController < ApplicationController
  # include this before_action in controller endpoints that you wish to edge cache
  # This can be used with any custom actions. Set these headers for GETs that you want to cache
  # e.g. before_action :set_cache_control_headers, only: [:index, :show, :my_custom_action]
  before_action :set_cache_control_headers, only: [:index, :show]

  # Returns all Article objects, and sets a table_key of 'articles',
  # and a record_key for each article object: "#{table_key}/#{article_id}"
  def index
    @articles = Article.all
    set_surrogate_key_header 'articles', @articles.map(&:record_key)
  end

  # Sets a surrogate key for the current article.
  #
  # Example:
  #
  #   Article[75]
  #   Surrogate-Key:articles/75
  def show
    set_surrogate_key_header @article.record_key
  end

  ...
end
```

The `before_action` method creates `Cache-Control` and `Surrogate-Control` HTTP headers with a default TTL of 30 days. This method must be added to any controller action that you want to edge cache. The `set_surrogate_key_header` method sets `Surrogate-Key` headers for objects that you want to be able to purge. In this case, surrogate keys are set for each article and the articles index.

Examining the headers

Now that you've looked at how the surrogate keys are generated by the test blog, inspect the headers on an article page on the origin server. Here's the partial output from `curl -svo /dev/null origin.example.com/articles/1`:

```
HTTP/1.1 200 OK
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Cache-Control: public, no-cache
Surrogate-Control: max-age=86400
Surrogate-Key: articles/1
```

Notice how the `Cache-Control`, `Surrogate-Control`, and `Surrogate-Key` headers are present in the response shown above. Thanks to the `set_surrogate_key_header` method, the test blog application automatically generates the unique `articles/1` surrogate key for this article.

Next, inspect the headers on the article index page on the origin server URL. Because this page lists all of the articles, you might expect it to contain the surrogate keys for every article displayed on the page, and that is indeed the case. Here's the partial output from `curl -svo /dev/null origin.example.com`:

```
HTTP/1.1 200 OK
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Cache-Control: public, no-cache
Surrogate-Control: max-age=86400
Surrogate-Key: articles articles/1 articles/2 articles/3 articles/4 articles/5 articles/6 articles/7 articles/8 articles/9 article
s/10 articles/11 articles/12 articles/15 articles/16 articles/17 articles/18 articles/19 articles/20 articles/27 articles/28 articl
es/29 articles/30 articles/31
```

If you purged the `articles/1` surrogate key, both http://origin.example.com/articles/1 and http://origin.example.com would be purged from the cache.

Finally, take a look at the URL that's piped through Fastly: http://fastly.example.com. The surrogate keys won't be visible in the headers, but Fastly knows what they are. Recall from understanding surrogate keys that Fastly strips out the `Surrogate-Header` and creates a mapping from each key to the cached content. Here's the partial output from `curl -svo /dev/null fastly.example.com`:

```
HTTP/1.1 200 OK
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Cache-Control: public, no-cache
Content-Type: text/html; charset=utf-8
Via: 1.1 varnish
Age: 78309
X-Served-By: cache-iad2120-IAD
X-Cache: HIT
X-Cache-Hits: 1
X-Timer: S1449255701.272992,VS0,VE5
```

With the `Surrogate-Key` header present on the index and article pages, you're now able to manually purge blog pages via key (/guides/purging/single-purges#purging-with-a-key), or purge a collection of pages with one API call (/api/purge).

Limitations

The surrogate keys sent by your origin server can be as simple or complex as you need, but there are a couple limitations:

- Surrogate keys are limited to a total length of 1,024 bytes each
- Any keys that exceed the limit will be dropped instead of truncated
- Surrogate key headers are limited to a total length of 16,384 bytes
- Any keys past the one that exceeds the limit will be dropped

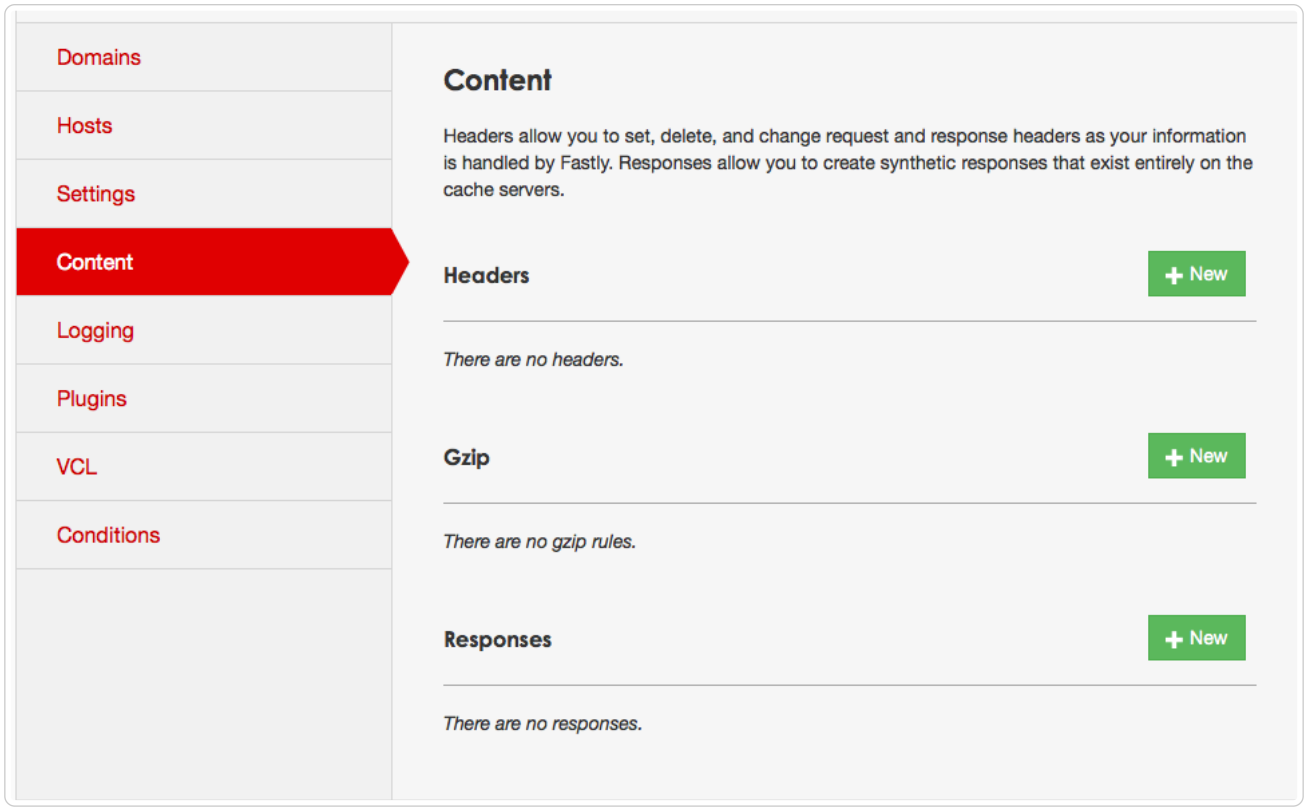
§ Setting Surrogate-Key headers based on a URL (/guides/purging/setting-surrogate-key-headers-based-on-a-url)

You can mark content with a surrogate key (/guides/purging/getting-started-with-surrogate-keys) and use it to purge groups of specific URLs (/guides/purging/single-purges.html#purging-with-a-key) at once without purging everything (/guides/purging/single-purges#purging-all-content), or purging each URL (/guides/purging/single-purges#purging-a-url) singularly.

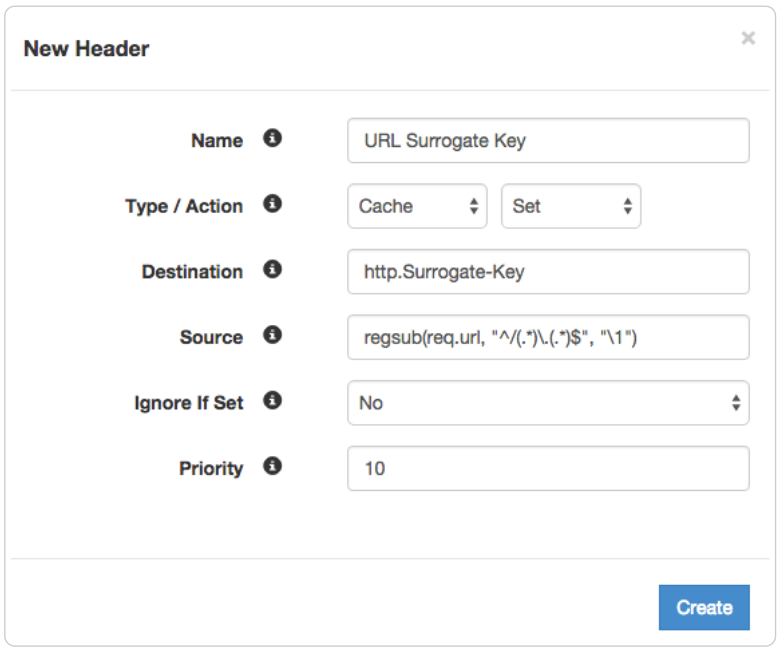
Follow these instructions to set Surrogate-Key headers based on a URL:

1. Log in to the Fastly application.
2. Click the **configure** tab (wrench icon).

3. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for the selected service appear.
4. Click **Content** from the section list on the left.



5. In the **Headers** area at the top of the page, click the **New** button. The New Header window appears.



6. Fill out the **New Header** fields as follows:
   - In the **Name** field, type a human-readable name for the header.
   - From the **Type** menu, select **Cache**. From the **Action** menu, select **Set**.
   - In the **Destination** field, type `http.Surrogate-Key`.
   - In the **Source** field, type `regsub(req.url, "^/(.*)\.(.*)$", "\1")`. This will accept a URL that looks like `/foo.html` and will create the Surrogate-Key `foo`.
   - From the **Ignore If Set** menu, select **No**.
   - In the **Priority** field, type `10`.
7. Click the **Create** button to create your header.

To publish the new header, deploy the version of the service you are editing.

NOTE: There are several limitations to surrogate keys. See the surrogate key limitations (/guides/purging/getting-started-with-surrogate-keys.html#limitations) section for more information.

§ Setting Surrogate-Key headers for Amazon S3 origins (/guides/purging/setting-surrogate-key-headers-for-amazon-s3-origins)

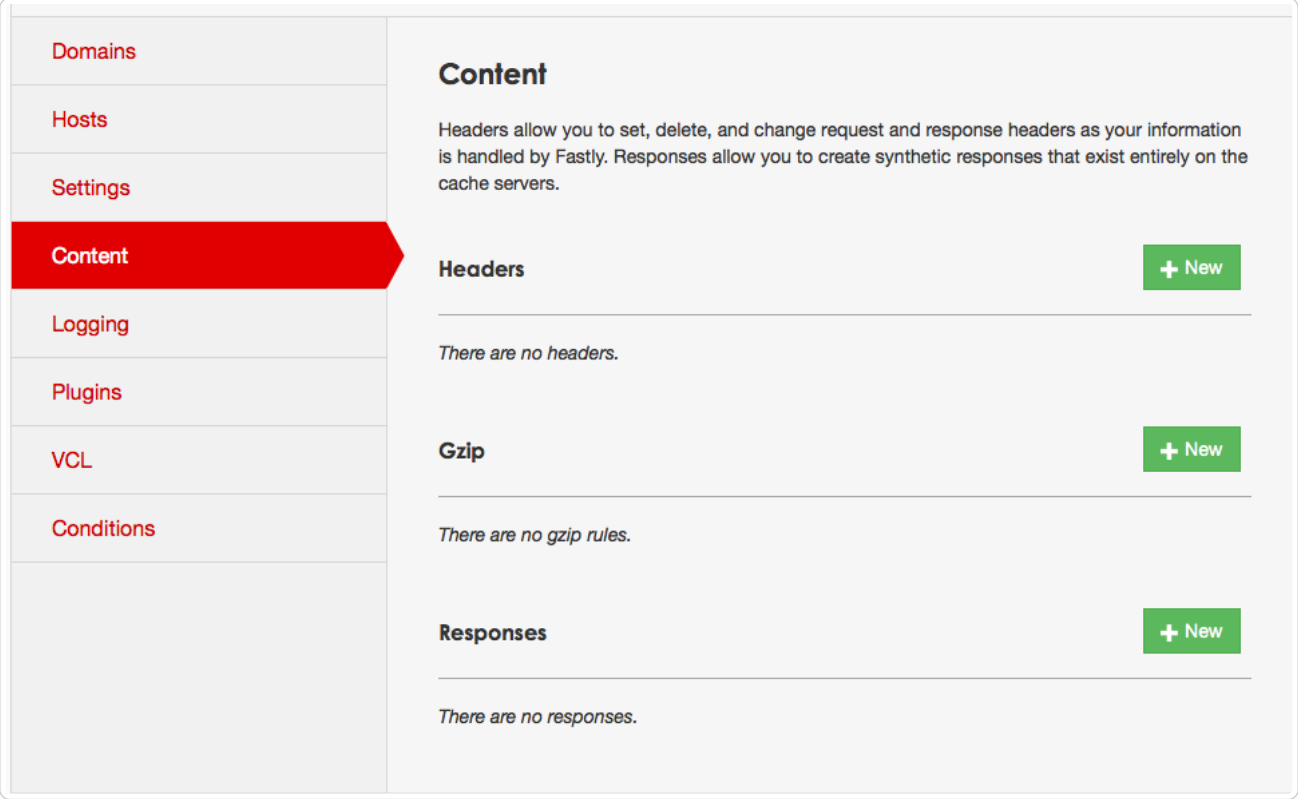
You can mark content with a surrogate key (/guides/purging/getting-started-with-surrogate-keys) and use it to purge groups of specific URLs (/guides/purging/single-purges.html#purging-with-a-key) at once without purging everything (/guides/purging/single-purges#purging-all-content), or purging each URL (/guides/purging/single-purges#purging-a-url) singularly. On the Amazon S3 side, you can use the `x-amz-meta-surrogate-key` header to mark your content as you see fit, and then on the Fastly side set up a Header configuration to translate the S3 information into the header we look for.

IMPORTANT: Pay close attention to the capitalization. Amazon S3 only accepts all lowercase header names.

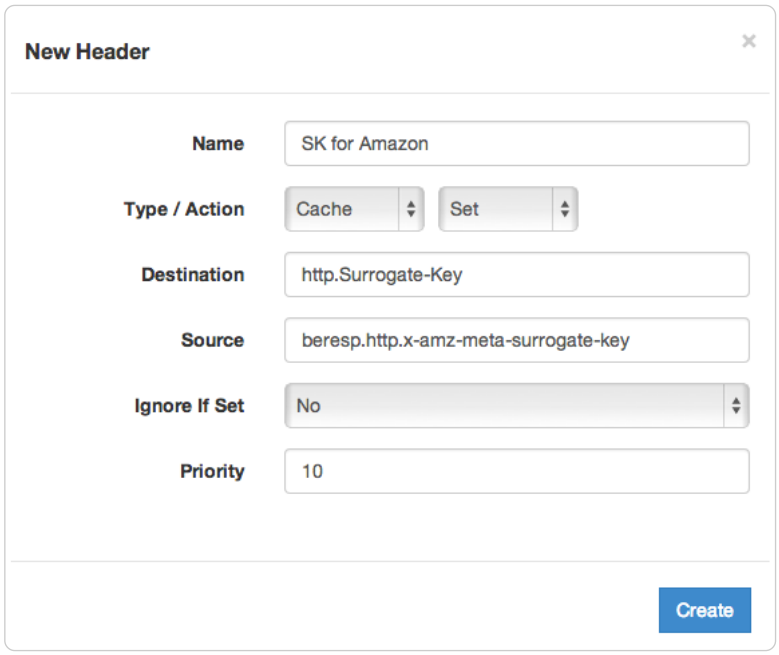
Follow these instructions to set Surrogate-Key headers for Amazon S3 origin servers:

1. Log in to the Fastly application.
2. Click the **configure** tab (wrench icon).

3. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for the selected service appear.
4. Click **Content** from the section list on the left.



5. In the **Headers** area at the top of the page, click the **New** button. The New Header window appears.



6. Fill out the **New Header** fields as follows:
   - In the **Name** field, type a human-readable name for the header.
   - From the **Type** menu, select **Cache**. From the **Action** menu, select **Set**.
   - In the **Destination** field, type `http.Surrogate-Key`.
   - In the **Source** field, type `beresp.http.x-amz-meta-surrogate-key`.
   - From the **Ignore If Set** menu, select **No**.
   - In the **Priority** field, type `10`.
7. Click the **Create** button to create your header.

To publish the new header, deploy the version of the service you are editing.

NOTE: There are several limitations to surrogate keys. See the surrogate key limitations (/guides/purging/getting-started-with-surrogate-keys.html#limitations) section for more information.

§ Single purges (/guides/purging/single-purges)

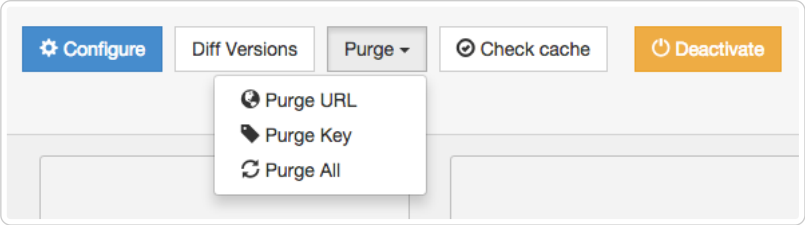
Fastly provides several levels of cache purging. You can purge something as small as a single URL via the "Purge URL" command or as large as all content under a service via the "Purge All" command. You can also selectively purge content via key-based purging using the "Purge Key" command. We also provide a purging feature called Soft Purge that allows you to mark content as outdated (stale) instead of permanently deleting it from Fastly's caches.

TIP: To mark content as outdated instead of permanently deleting it, check out our Soft Purge (/guides/purging/soft-purges) feature. You may also be interested in our wildcard purging (/guides/purging/wildcard-purges).

Purging via the user interface

To purge content using the Fastly application, follow the steps below.

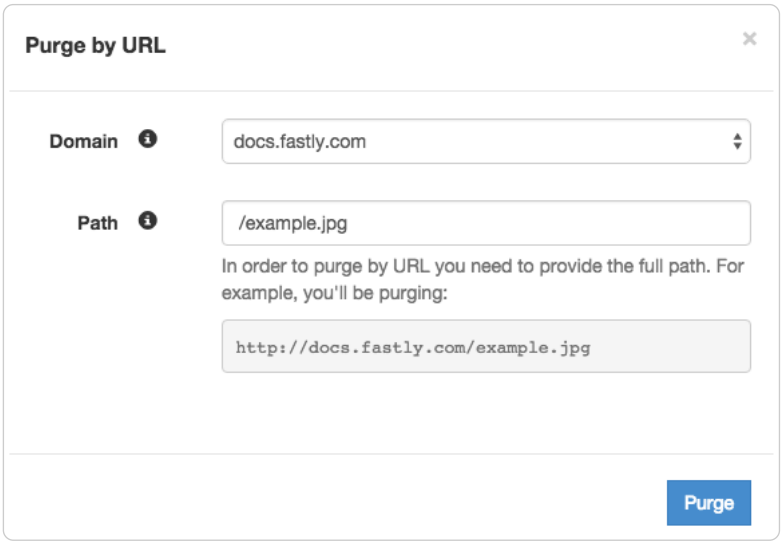
1. Log in to the Fastly application.
2. Click the **configure** button (the wrench icon at the top of the window).
3. From the **Service** menu, select the service you want to work on.
4. From the **Purge** menu button, select the appropriate purge type.



5. If asked to confirm your password, type your password in the confirmation window and click the **Confirm** button.
6. In the purging window that appears (see the examples below), provide the appropriate information to purge your content and then click the **Purge** button.

Purging a URL

To purge a single URL, select the domain from the **Domain** menu and type the path to the content in the **Path** field. For example, to purge `https://docs.fastly.com/example.jpg`, you would select `docs.fastly.com` from the **Domain** menu and type `/example.jpg` in the **Path** field.

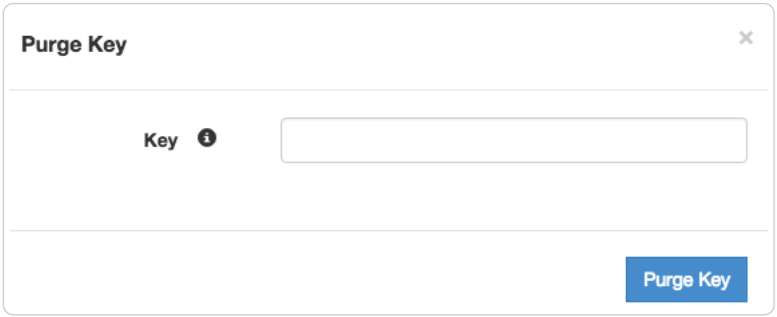


NOTE: By default, authentication is not required to purge a URL with the Fastly API, but you can enable API key authentication in the Fastly application by adding a header and attaching a condition, or by using custom VCL. See Authenticating URL purge requests via API (/guides/purging/authenticating-api-purge-requests) for more information.

Purging with a key

Key-based purging of cache information allows you to include surrogate key headers (/guides/purging/getting-started-with-surrogate-keys) with your content, which our caches strip out and index. You can then issue purge commands to selectively flush out anything with a specific surrogate key. For extra usefulness, items can have multiple keys that allow you to set up your own purging hierarchy.

To purge content tagged with a specific key, type that key in the **Key** field.



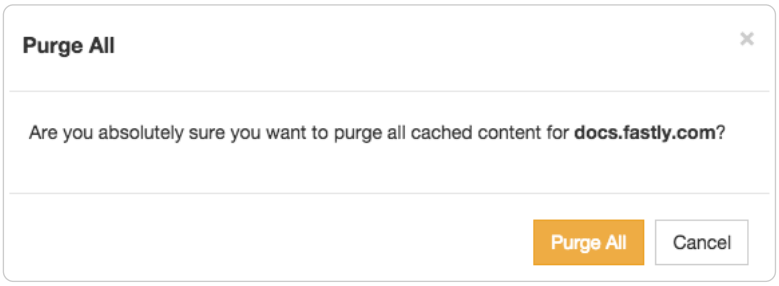
IMPORTANT: To enable key purging, your server must also send a special response header along with the content being cached. Have your server send the `Surrogate-Key` header with a list of keys for this URL. If you're using multiple keys, separate them with spaces.

For example, to tag a response with both the keys `foo` and `bar` you would send this header:

```
Surrogate-Key: foo bar
```

Purging all content

To instantly purge all content under your service, select the **Purge All** option. This clears all cached content for all domains on your service. Selecting this option displays a window asking you to confirm your decision.



⚠ **WARNING:** Do not purge all cached content if you are seeing 503 errors (/guides/debugging/common-503-errors). Purge all overrides stale-if-error (/guides/performance-tuning/serving-stale-content) and increases the requests to your origin server, which could result in additional 503 errors.

Purging via API

The syntax for purging a service through the API can be found in the Purging section (/api/purge) of the API (/api/) documentation.

§ Soft purges (/guides/purging/soft-purges)

Fastly provides a Soft Purge feature that allows you to mark content as outdated (stale) instead of permanently purging and thereby deleting it from Fastly's caches. Objects invalidated with Soft Purge will be treated as outdated (stale) while Fastly fetches a new version from origin. You can purge by URL or by surrogate key using Soft Purge.

Before using Soft Purge, we recommend you implement one of the following revalidation methods:

- Configure `stale_while_revalidate` to serve stale content (/guides/performance-tuning/serving-stale-content) and fetch the newest version of the object from origin in the background. If you choose this revalidation method, consider also configuring `stale_if_error` at the same time.
- Set up `ETag` or `Last-Modified` headers for relevant content on your origin servers.

To implement Soft Purge, add a `Fastly-Soft-Purge` request header (such as `Fastly-Soft-Purge: 1`) to any single URL or key-based purge. For example, to purge the URL `www.example.com` with Soft Purge, you would issue the following command:

```
curl -X PURGE -H "Fastly-Soft-Purge:1" http://www.example.com
```

§ Wildcard purges (/guides/purging/wildcard-purges)

Wildcard purging allows you to flush the cache of all pages under a directory branch or URL path; for example, you want to empty the cache of all pages under your "/service" path. Having to purge each URL (/guides/purging/single-purges#example:-purge-url) one by one using the Fastly API (/api/purge) or via the Fastly app is not very efficient.

Although Fastly does not have a specific wildcard purge function, you can implement the same behavior by making a small configuration change using surrogate keys (https://www.fastly.com/blog/surrogate-keys-part-1). Surrogate keys allow you to tag a group of objects with a keyword (key) and then purge multiple pieces of content at once with it through our Fastly application or API.

❗ **IMPORTANT:** Purging will only apply to new objects as they're being put into the cache after you set up configuration changes. It will not apply to objects already in the cache when this configuration is being applied.

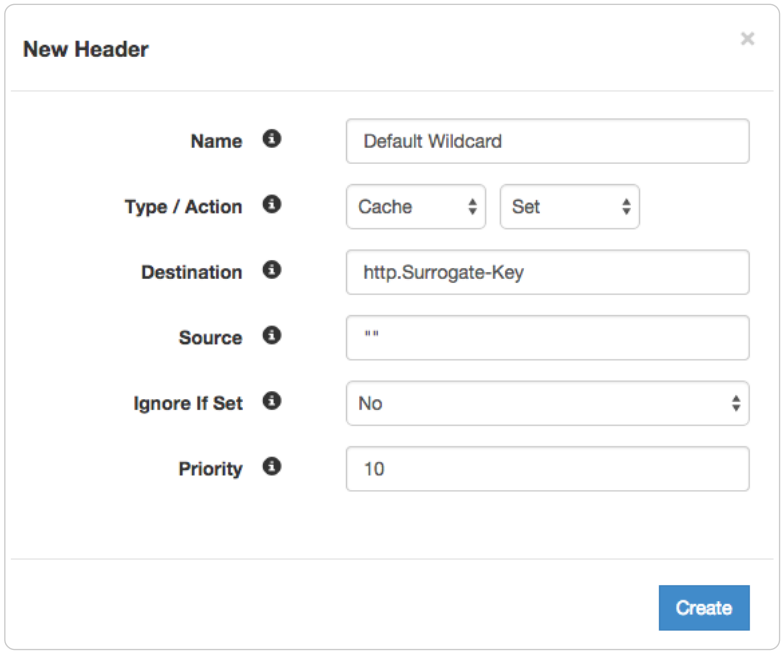
To purge content based on wildcard paths, follow the steps below.

Create a default wildcard header

We set a default wildcard so that we have the flexibility to append other surrogate keys to a URL path.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Content** section.
6. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.

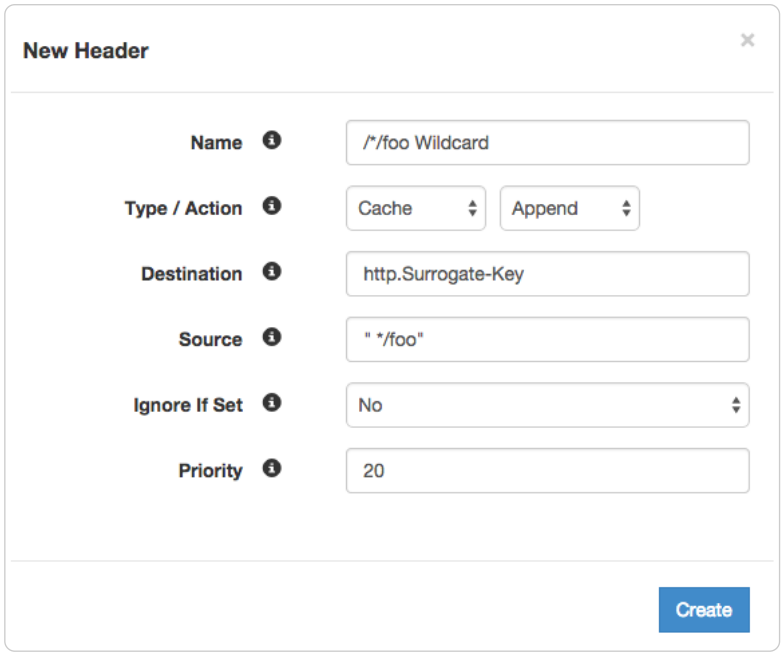


7. Fill out the **New Header** window as follows:
   - In the **Name** field, type `Default Wildcard`.
   - From the **Type/Action** menus, select **Cache** and **Set**.
   - In the **Destination** field type `http.Surrogate-Key`.
   - In the **Source** field type `""`.
   - From the **Ignore if Set** menu, select **No**.
   - In the **Priority** field, type `10`.
8. Click the **Create** button. A new header appears in the Headers area of the Content section.

Create headers for each wildcard path being purged

Next, create a header for each of the wildcard paths you need the ability to purge. For instance, you want to purge the wildcard path `/*/foo`.

1. Click the **New** button again to create another new header.



2. Fill out the next **New Header** window as follows:
   - In the **Name** field, type `/*/foo Wildcard`.
   - From the **Type/Action** menus, select **Cache** and **Append**.
   - In the **Destination** field type `http.Surrogate-Key`.
   - In the **Source** field type `" */foo"`.
   - From the **Ignore if Set** menu, select **No**.
   - In the **Priority** field, type `20`.
3. Click the **Create** button. A new header appears in the Headers area of the Content section.

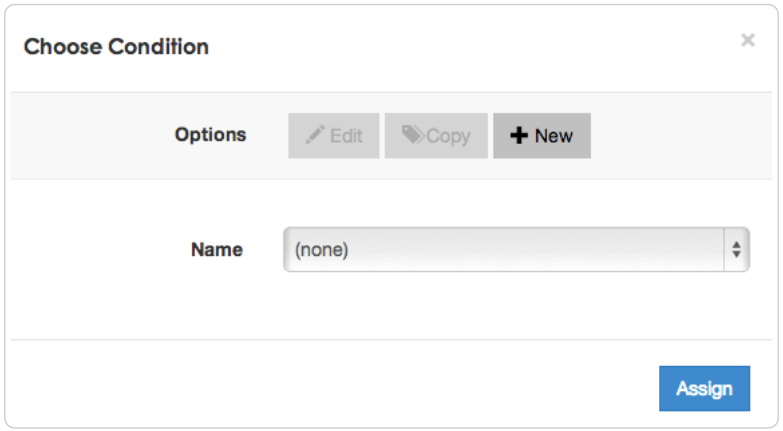
Notice the Action is set to Append to add to the default wildcard surrogate key. Also, there is a space before the asterisk in the Source field, which is important when appending multiple surrogate keys to a URL. Finally, the Priority is set to 20 so that the Default Wildcard header is executed first and then the wildcard path appends.

Create conditions for each wildcard path being purged

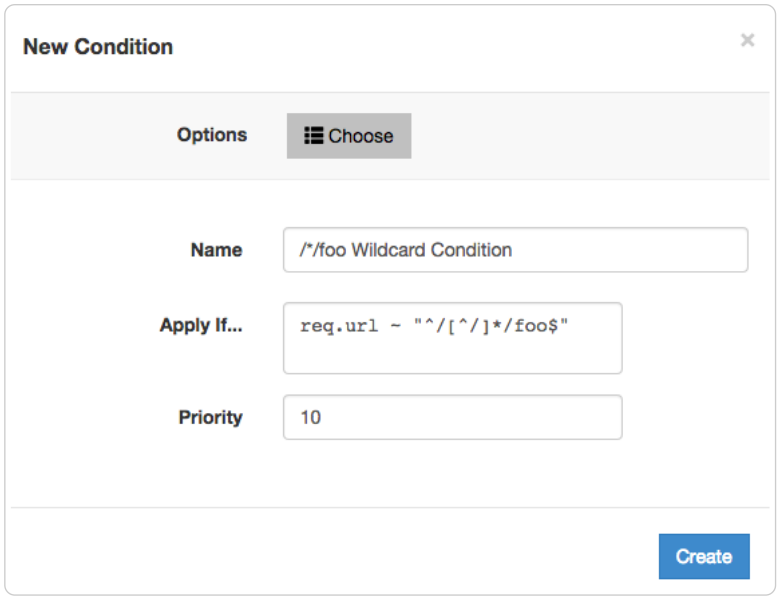
Finally, create a condition for each of the wildcard paths you need the ability to purge. Once you add the new wildcard path header, a gear icon appears to the right of its name in the Header area of the Content section.

This is where you'll create a new condition following the steps below.

1. Click the gear icon to the right of the wildcard path header name and select **Cache Conditions** from the menu. The Choose Condition window appears.



2. In the **Options** section, click the **New** button to add a new condition. The New Condition window appears.



3. Fill out the **New Condition** controls as follows:
   - In the **Name** field, type `/*/foo Wildcard Condition`.
   - In the **Apply If** field, type `req.url ~ "^/[^/]*/foo$"`.
   - In the **Priority** field, type `10`.
4. Click **Create** to create the new condition.

What does the condition mean? In the Apply If field above, the first `"^"` and `"$"` tells Fastly to look for the following pattern:

- Start from the first slash after the request host header.
- There should be one directory.
- It should be followed by the path `/foo` ending the URL.

Some examples would be `/a/foo`, `/bar/foo`, and `/c/foo`. You could also remove the first `"^"` and `">"$"` to allow the condition to be more general so that the pattern can occur in the middle of a URL path.

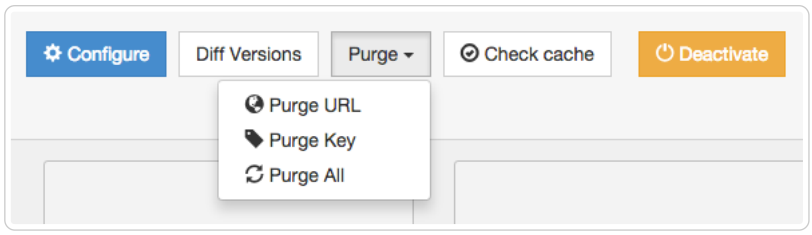
Some other examples for URL wildcard conditions:

Apply If field	Matched pattern
`req.url ~ "/[^/]*/foo"`	`/delta/wow/a/foo/neat/cool/img.gif`
`req.url ~ "^/.*/foo$"`	`/a/b/c/d/e/f/foo`

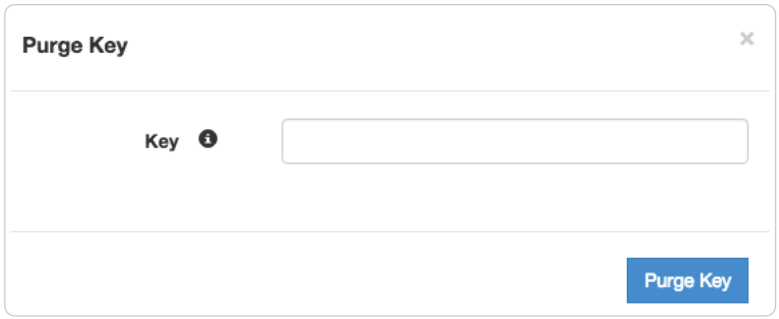
Purge the wildcard

Ready to purge that wildcard? You can do this through the UI using the steps below.

1. Select the appropriate service from the **Service** menu.
2. Click the **Purge** button to the right of the service name and then select **Purge Key** from the menu that appears.



   The Purge Key window appears.
3. In the **Key** field, type the Source name of the surrogate key you want to purge, without the quotations.



Continuing with our example, you would type `*/foo` without the quotes that were entered in the Source field of the New Header window above.

4. Click the **Purge Key** button.

You can also use our key-based purging via the API using an HTTP request like so:

```
POST /service/<Fastly Service ID>/purge/*/foo
Fastly-Key: <Fastly Key>
```

Both the UI version and the API version will purge any content that was associated with the `"*/foo"` surrogate key according to the setup in your header rules.

• Guides (/guides/) > Developer's tools (/guides/devtools) > Access Control Lists (/guides/access-control-lists/)

§ About Edge ACLs (/guides/access-control-lists/about-edge-acls)

Edge ACLs allow you to attach an access control list (ACL) to a service with versionless ACL entries that are stored separately from your VCL configuration. You can use the Fastly API to programmatically add, remove, and update ACLs and their entries.

How Edge ACLs work

Like Edge Dictionaries (/guides/edge-dictionaries/about-edge-dictionaries), Edge ACLs have two major components: The ACL itself (/guides/access-control-lists/creating-and-using-edge-acls), and the ACL entries (/guides/access-control-lists/creating-and-manipulating-edge-acl-entries) within it. ACLs are containers for ACL entries, which store IP addresses that can be used to whitelist or blacklist access to resources. Every ACL is attached to a version of a service, but ACL entries are versionless and any changes will become effective immediately.

When ACLs might be useful

- E-commerce companies preventing scraping from certain IP ranges
- Offices restricting access to their administrative portals
- Advertising technology companies blocking bad-actors at edge
- Mobile applications accepting only calls from specific proxies or IP ranges
- System administrators restricting access to groups of backends from an office IP address or subnet range

Getting started

To create an ACL and use it within your service, you'll need to perform the following steps:

1. Create an empty ACL (/guides/access-control-lists/creating-and-using-edge-acls) in a working version of a service that's unlocked and not yet activated.
2. Activate the new version of the service (/guides/basic-setup/working-with-services#service-versions-and-their-activation) you associated with the empty ACL.
3. Add ACL entries (/guides/access-control-lists/creating-and-manipulating-edge-acl-entries) to the newly created ACL.

Once the ACL is created, properly associated, and filled with ACL entries, it can be called in your service.

Putting Edge ACLs to work

After you've used the Fastly API to create an ACL and add ACL entries, the VCL for the ACLs and ACL entries will be automatically generated, as shown below. In this example, the name of the ACL is `office_ip_ranges`.

```
# This VCL is automatically generated when you add an Edge ACL and entries with the Fastly API
# In this example, the ACL name is "office_ip_ranges"
acl office_ip_ranges {
  "10.1.1.0"/16;                            # internal office
  "8.19.222.194";                           # remote VPN office
  "FE80:0000:0000:0000:0202:B3FF:FE1E:8329";   # ipv6 address remote
}
```

You can upload custom VCL (/guides/vcl/uploading-custom-vcl) to add logic to interact with ACLs. In this example, the `office_ip_ranges` ACL is used as a whitelist. All access to `/admin` is denied by default, but IP addresses in the `office_ip_ranges` ACL are allowed to access `/admin` without restriction.

```
sub vcl_recv {
  # block all requests to Admin pages from IP addresses not in office_ip_ranges
  if (req.url ~ "^/admin" && ! (client.ip ~ office_ip_ranges)) {
    error 403 "Forbidden";
  }
}
```

ACL entries have a boolean option for negation which allows you to specify whether or not the IP address is whitelisted or blacklisted. You can set the option to true (1) to blacklist (deny), or false (0) to whitelist (allow). We *do not* recommend mixing both negated and non-negated entries in the same ACL. Doing so might have the opposite effect depending on the condition you specify in the VCL.

Limitations

When creating Edge ACLs (/guides/access-control-lists/creating-and-using-edge-acls), keep the following limitations in mind as you develop your service configurations:

- **ACLs created with custom VCL cannot be manipulated using the API.** If you create an ACL using the API, you can use the API to make changes to it, and use custom VCL to interact with it. If you create an ACL using custom VCL (/guides/vcl/using-access-control-lists), that ACL must always be manipulated via custom VCL.
- **Custom VCL manipulations are always versioned.** This is true for both ACLs created using custom VCL and for any logic created to interact with ACLs.
- **ACLs are limited to 1000 ACL entries.** If you find your ACLs approaching this limit, you may want to review this with your account manager. We may be able to help you figure out an even more efficient way to do things with your ACLs.
- **Audit logs don't exist for ACL changes.** If you use the API to add, update, or remove an ACL entry, there will be no record of it. The only record of a change will exist when you compare service versions (https://www.fastly.com/blog/introducing-version-diff) to view the point at which the ACL was associated with the service version in the first place.
- **When you delete an ACL, you'll only delete it from the service version you're editing.** ACLs are tied to versions and can be cloned and reverted. When using ACLs, we want you to be able to do things like delete an ACL from a current version of your service in order to roll back your configuration to a previous version using as few steps as possible.
- **When you delete an ACL, we remove the ACL entries inside it.** When you delete an ACL we remove the entries that live inside that container, but they are not deleted from the ACL in earlier versions of your configuration. You can roll back to a previous version of a service to restore the ACL and the ACL entries.
- **ACL entry deletions are permanent.** If you delete an ACL entry, the entry is permanently removed from all service versions and cannot be recovered.

§ Creating and manipulating Edge ACL entries (/guides/access-control-lists/creating-and-manipulating-edge-acl-entries)

Edge ACL (/guides/access-control-lists/about-edge-acls) entries contain IP addresses that can be used to whitelist or blacklist access to resources. ACL entries are versionless, and any changes will become effective immediately, regardless of version.

Attributes

Edge ACL entries have the following attributes:

- **Service ID:** The ID of the Fastly service the ACL is associated with.
- **ACL ID:** The ID of the ACL.

- **ACL Entry ID:** The ID of the ACL entry.
- **IP Address:** The IP address contained within the ACL entry.
- **Subnet:** Optional. The range of IP addresses within a single ACL entry.
- **Boolean option for negation:** Defaults to false. A number indicating true (`1`) to blacklist (deny), or false (`0`) to whitelist (allow). Note that we *do not* recommend mixing both negated and non-negated entries in the same ACL.
- **Comment:** Optional. A descriptive comment indicating why you created the ACL entry.

Creating an ACL entry

To add an entry to an existing ACL, make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" -X POST https://api.fastly.com/service/<service_id>/acl/<acl_id>/entry -d 'ip=127.0.0.1&subnet=16&
negated=1&comment=do not allow'
```

The response will look like this:

```
{
    "acl_id": "<acl_id>",
    "comment": "do not allow",
    "created_at": "2016-04-22T19:14:02+00:00",
    "deleted_at": null,
    "id": "<acl_entry_id>",
    "ip": "127.0.0.1",
    "negated": "1",
    "service_id": "<service_id>",
    "subnet": 16,
    "updated_at": "2016-04-22T19:14:02+00:00"
}
```

Viewing ACL entries

To see information related to a single ACL entry, make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" -H 'Content-Type: application/vnd.api+json' https://api.fastly.com/service/<service_id>/acl/<acl_i
d>/entry/<acl_entry_id>
```

The response will look like this:

```
{
    "acl_id": "<acl_id>",
    "comment": "do not allow",
    "created_at": "2016-04-22T19:18:42+00:00",
    "deleted_at": null,
    "id": "<acl_entry_id>",
    "ip": "127.0.0.5",
    "negated": "1",
    "service_id": "<service_id>",
    "subnet": 16,
    "updated_at": "2016-04-22T19:18:42+00:00"
}
```

To view a list of all ACL entries attached to a particular ACL, make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" https://api.fastly.com/service/<service_id>/acl/<acl_id>/entries
```

The response will look like this:

```
[
    {
        "acl_id": "<acl_id>",
        "comment": "",
        "created_at": "2016-04-22T19:13:03+00:00",
        "deleted_at": null,
        "id": "<acl_entry_1_id>",
        "ip": "127.0.0.1",
        "negated": "0",
        "service_id": "<service_id>",
        "subnet": 16,
        "updated_at": "2016-04-22T19:13:03+00:00"
    },
    {
        "acl_id": "<acl_id>",
        "comment": "",
        "created_at": "2016-04-22T19:14:02+00:00",
        "deleted_at": null,
        "id": "<acl_entry_2_id>",
        "ip": "127.0.0.2",
        "negated": "1",
        "service_id": "<service_id>",
        "subnet": 16,
        "updated_at": "2016-04-22T19:14:02+00:00"
    }
]
```

Updating an ACL entry

To update an existing ACL entry, make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" -X PATCH https://api.fastly.com/service/<service_id>/acl/<acl_id>/entry/<acl_entry_id> -d 'ip=127.0.0.2&subnet=32&negated=0&comment=allow'
```

The response will look like this:

```
{
    "acl_id": "<acl_id>",
    "comment": "allow",
    "created_at": "2016-04-22T19:18:42+00:00",
    "deleted_at": null,
    "id": "<acl_entry_id>",
    "ip": "127.0.0.2",
    "negated": "0",
    "service_id": "<service_id>",
    "subnet": 32,
    "updated_at": "2016-04-22T19:18:42+00:00"
}
```

Deleting an ACL entry

⚠ **WARNING:** ACL entry deletions are permanent. If you delete an ACL entry, the entry is permanently removed from all service versions and cannot be recovered.

To permanently delete an ACL entry, make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" -X DELETE https://api.fastly.com/service/<service_id>/acl/<acl_id>/entry/<acl_entry_id>
```

The response will look like this:

```
{
  "status":"ok"
}
```

§ Creating and using Edge ACLs (/guides/access-control-lists/creating-and-using-edge-acls)

Edge ACLs (/guides/access-control-lists/about-edge-acls) are lists of permissions that Varnish uses to grant or restrict access to URLs within your services. You can use the Fastly API to programmatically add, remove, and update Edge ACLs and their entries. To create an Edge ACL and use it within your service, you'll need to perform the following steps:

1. Create an empty ACL (/guides/access-control-lists/creating-and-using-edge-acls) in a working version of a service that's unlocked and not yet activated.
2. Activate the new version of the service (/guides/basic-setup/working-with-services#service-versions-and-their-activation) you associated with the empty ACL.
3. Add ACL entries (/guides/access-control-lists/creating-and-manipulating-edge-acl-entries) to the newly created ACL.

Once the ACL is created, properly associated, and filled with ACL entries, it can be called in your service.

Attributes

Edge ACLs have the following attributes:

- **Service ID:** The ID of the Fastly service the ACL is associated with.
- **VCL Version Number:** The VCL version number the ACL is associated with. Note that the ACL will continue to reside within subsequently cloned counterparts.
- **ACL Name:** The name of the ACL.
- **ACL ID:** The unique identifier of the ACL.

Creating an ACL

To start using an ACL, you'll need to create an empty one within a version of a service that's unlocked and not yet activated. Make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" -X POST https://api.fastly.com/service/<service_id>/version/<vcl_version_number>/acl -d name=my_acl
```

The response will look like this:

```
{
    "id": "<vcl_version_number>",
    "name": "my_acl",
    "service_id": "<service_id>",
    "version": "1",
    "created_at": "2016-04-14 21:23:21",
    "updated_at": "2016-04-14 21:23:21"
}
```

Be sure to activate the new version of the service you associated with the empty ACL.

Viewing ACLs

To see information related to a single ACL (in this example, `my_acl`) attached to a particular version of a service, make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" https://api.fastly.com/service/<service_id>/version/<vcl_version_number>/acl/my_acl
```

The response will look like this:

```
{
    "id": "<acl_id>",
    "name": "my_acl",
    "service_id": "<service_id>",
    "version": "<vcl_version_number>",
    "created_at": "2016-04-14 21:23:21",
    "updated_at": "2016-04-14 21:23:21"
}
```

To view a list of all ACLs attached to a particular version of a service, make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" https://api.fastly.com/service/<service_id>/version/<vcl_version_number>/acl
```

The response will look like this:

```
[
    {
        "id": "<acl_1_id>",
        "name": "my_new_acl",
        "service_id": "<service_id>",
        "version": "<vcl_version_number>",
        "created_at": "2016-04-14 21:23:21",
        "updated_at": "2016-04-15 17:23:09"
},
    {
        "id": "<acl_2_id>",
        "name": "my_other_acl",
        "service_id": "<service_id>",
        "version": "<vcl_version_number>",
        "created_at": "2016-04-14 21:23:21",
        "updated_at": "2016-04-15 17:23:09"
    }
]
```

Deleting an ACL

Deleting an ACL deletes the ACL and all of its associated entries. To delete an ACL (in this example, `my_new_acl`), make the following API call in a terminal application:

```
curl -H "Fastly-Key: FASTLY_KEY" -X DELETE https://api.fastly.com/service/<service_id>/version/<vcl_version_number>/acl/my_new_acl
```

The response will look like this:

```
{
  "status":"ok"
}
```

• Guides (/guides/) > Developer's tools (/guides/devtools) > API Caching (/guides/api-caching/)

§ Enabling API caching (/guides/api-caching/enabling-api-caching)

Application Programming Interfaces (APIs) allow you to retrieve data from a variety of web services. Fastly makes it possible for you to cache your API so you can accelerate the performance of your service-oriented architecture. It optimizes your API's performance by efficiently handling traffic bursts and reducing latency.

This guide consists of three parts:

- Determining which API endpoints to cache
- Configuring PURGE requests
- Setting up Fastly

An example

Let's look at an example to learn how API caching works. Imagine we're an online magazine with articles on which users can make comments. Each article can have many comments, and each comment is authored by exactly one user.

We'll design a RESTful API specification and use it to manipulate and retrieve comments:

- `GET /comment` - Returns a list of all comments
- `GET /comment/:id` - Returns a comment with the given ID

- `POST /comment` - Creates a new comment
- `PUT /comment/:id` - Updates a comment with the given ID
- `DELETE /comment/:id` - Deletes a comment with the given ID

The create, read, update, and delete (CRUD) methods ensure the API can perform its basic operations, but they don't expose the relational aspect of the data. To do so, you would add a couple of relational endpoints:

- `GET /articles/:article_id/comments` - Get a list of comments for a given article
- `GET /user/:user_id/comments` - Get all comments for a given user

Endpoints like these allow programmers to get the information they need to do things like render the HTML page for an article, or display comments on a user's profile page. While there are many other possible endpoints we could construct, this set should suffice for the purposes of this guide. Let's assume that the API has been programmed to use an Object-Relational Mapper (ORM), such as ActiveRecord, when interacting with the database.

Determining which API endpoints to cache

Start by identifying the URLs you want to cache. We recommend splitting the specification endpoints into two groups.

The first group, called "accessors," retrieves or accesses the comment data. These are the endpoints you want to cache using Fastly. Using the example, four endpoints match this description:

- `GET /comment`
- `GET /comment/:id`
- `GET /article/:article_id/comments`
- `GET /user/:user_id/comments`

The second group, called "mutators," changes or mutates the comment data. These endpoints are always dynamic, and are therefore uncacheable. Using the example, three endpoints match this description:

- `POST /comment`
- `PUT /comment/:id`
- `DELETE /comment/:id`

You should see a pattern emerging. Because the example API is RESTful, we can use a simple rule to identify the accessor and mutator endpoints: GET endpoints can be cached, but PUT, POST, and DELETE endpoints cannot.

Once you've gathered this information, you're ready to program the API to configure PURGE requests.

Configuring PURGE requests

Don't be tempted to point at the PUT, POST, and DELETE endpoints as the place where data is modified. In most modern APIs, these endpoints represent an interface to the actual model code responsible for handling the database modifications.

In the example, we assumed that we'd be using an ORM to perform the actual database work. Most ORMs allow programmers to set special "callbacks" on models that will fire when certain actions have been performed (e.g., before or after validation, or after creating a new record).

For purging, we are interested in whether a model has saved information to the database — whether it's a new record, an update to an existing record, or the deleting of a record. At this point, we'd add a callback that tells the API to send a PURGE request to Fastly for each of the cacheable endpoints.

For an ActiveRecord comments model, you could do something like this:

```
require 'fastly'

class Comment < ActiveRecord::Base
  fastly = Fastly.new(api_key: 'MY_API_KEY')

  after_save do
    fastly.purge "/comment"
    fastly.purge "/comment/#{self.id}"
    fastly.purge "/article/#{self.article_id}/comments"
    fastly.purge "/user/#{self.user_id}/comments"
  end
end
```

Keep two things in mind when creating the callback:

- The purge code should be triggered *after* the information has been saved to the database, otherwise a race condition could be created where Fastly fetches the data from the origin server before the data has been saved to the database. This would cache the old data instead of the

new data.

- These URLs are being purged because they have content that changes when a comment is changed.

With the model code in place, the API is now ready to be cached.

Setting up Fastly

The final step is setting up Fastly. You'll need to perform the following steps:

- Create a new service
- Add the domain for the API
- Add the origin server that powers the API

In addition, you can optionally create rules that tell Fastly how to work with the specific elements that are exclusive to your API.

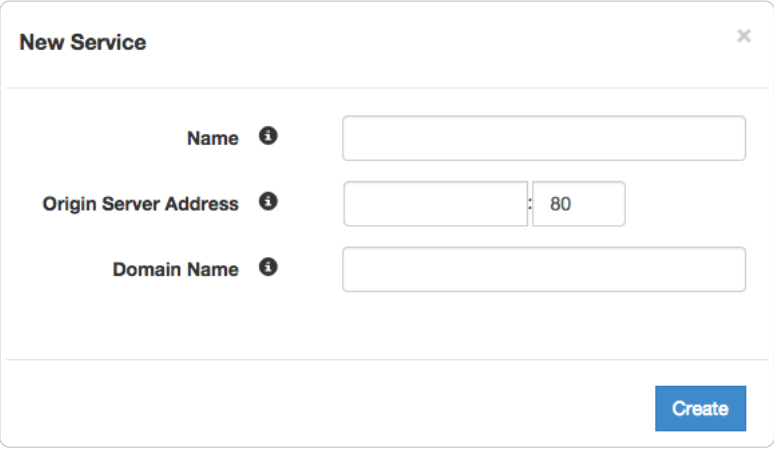
> **NOTE:** By default, Fastly will not cache PUT, POST, and DELETE requests. For more information, see our article on default caching behavior of HTTP methods (/guides/debugging/using-get-instead-of-head-for-command-line-caching-tests).

Creating a new service

Follow these instructions to add a new service to your Fastly account:

1. Log in to the Fastly application.
2. Click the **configure** tab (wrench icon).

3. Click the green **New Service** button. The New Service window appears.



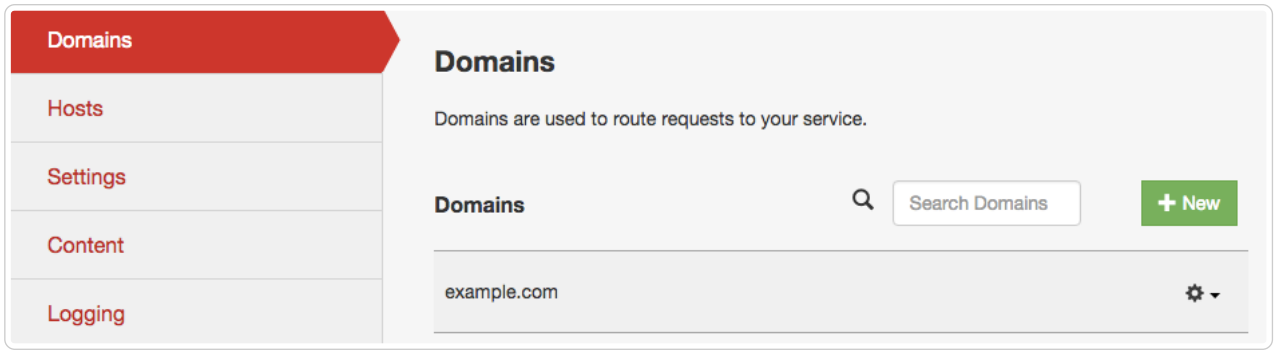
4. Fill out the **New Service** fields as follows:
   - In the **Name** field, type a name for this service (e.g., `My API Service`).
   - In the **Origin Server Address** field, type the IP address of your API server.
   - In the **Domain Name** field, type the domain name associated with your API (e.g., `api.example.com`).
5. Click **Create**.

Your service now appears in the Service menu.

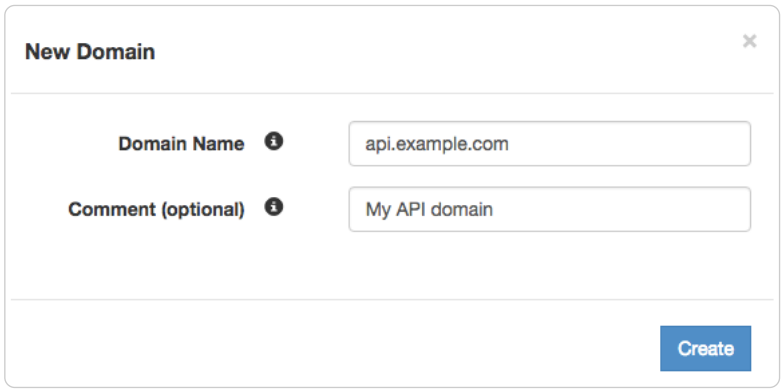
Adding the domain

Follow these instructions to add the API's domain name to your Fastly service:

1. From the **Service** menu, select the service you just created and then click the blue **Configure** button. The main controls for your selected service appear.
2. Click **Domains** from the section list on the left. The Domains page appears.



3. Click **New**. The New Domain window appears.

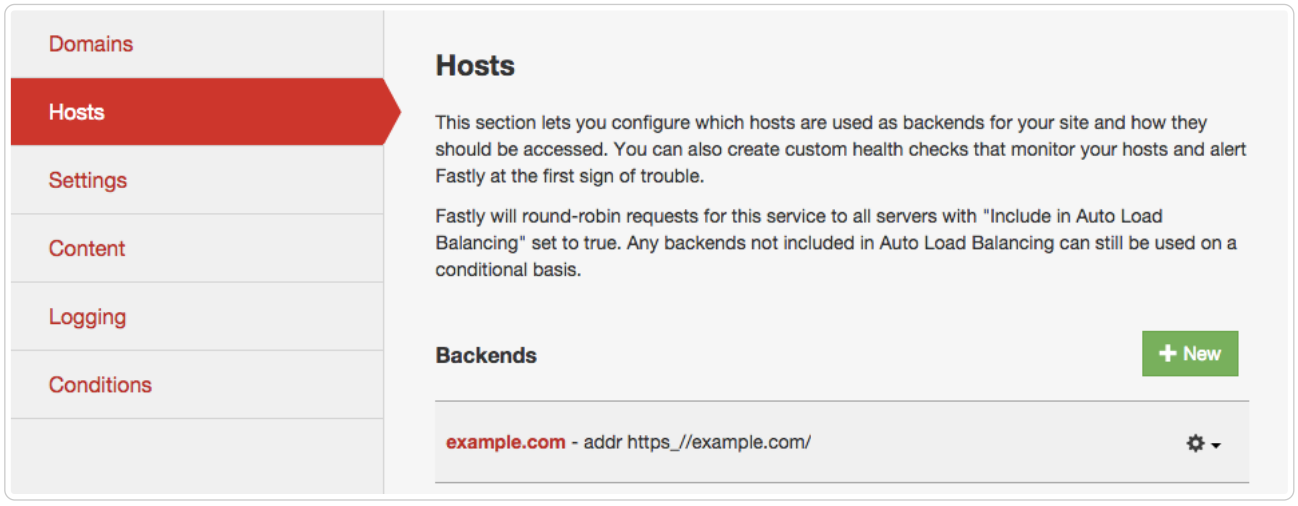


4. Fill out the **New Domain** window as follows:
   - In the **Domain Name** field, type the domain name for the API.
   - In the **Comment** field, optionally type a comment.
5. Click **Create**. Your API's domain name appears in the list of domains.

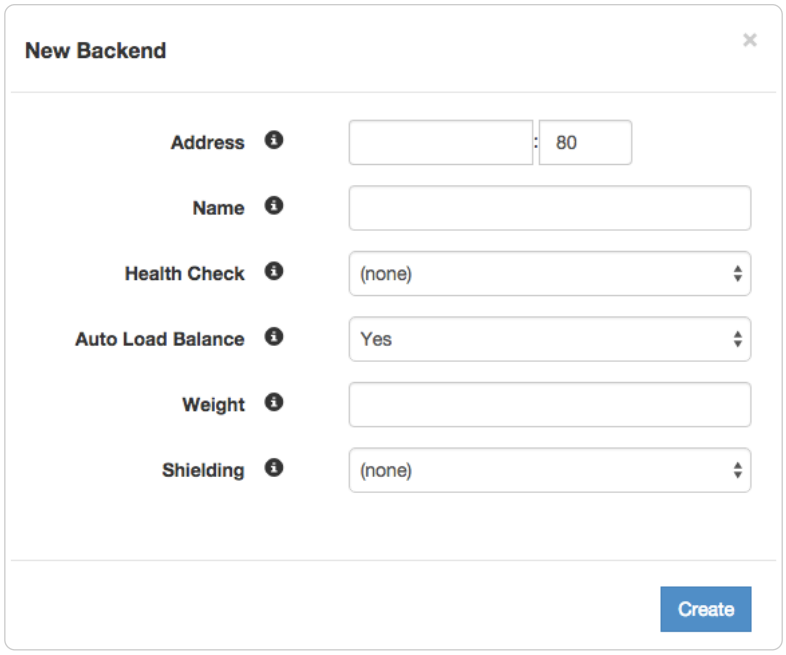
Adding the origin server

Follow these instructions to add the origin server that runs the API software to your Fastly service:

1. Click **Hosts** from the section list on the left. The Hosts page appears.



2. In the **Backends** area, click the **New** button. The New Backend window appears.



3. Fill out the **New Backend** fields as follows:
   - In the **Address** field, type the IP address or hostname and port number of the origin server for your API.
   - In the **Name** field, type a name for this origin server (e.g., `My API Server`).
   - From the **Health Check** menu, optionally select a health check for this origin server. For more information, see our health checks tutorial (/guides/basic-configuration/health-checks-tutorial).
   - From the **Auto Load Balance** menu, select an option to enable or disable load balancing for this origin server. For more information, see our guide on load balancing (/guides/performance-tuning/load-balancing-configuration).
   - If you enabled load balancing, type a weight in the **Weight** field.
   - If you'd like to use the shielding feature (/guides/performance-tuning/shielding), select a POP from the **Shielding** menu.
4. Click **Create**. The origin server appears in the list of origin servers.

§ Implementing API cache control (/guides/api-caching/implementing-api-cache-control)

This guide explains how to implement API cache control. Once you've enabled API caching (/guides/api-caching/enabling-api-caching), and ensured purging works properly with your cached data, you can set up specific headers like cache-control and surrogate-control to change when data is cached.

This guide consists of three parts:

- Understanding cache control headers
- Updating the example to use cache-control
- Implementing cache-control

Understanding cache control headers

In general, we assume that GET requests are cached and PUT, POST, and DELETE requests are not. For an ideal REST API, this rule works well. Unfortunately, most APIs are far from ideal and require additional caching rules for some requests.

For these reasons, it's a good idea to set cache-control headers when migrating APIs to Fastly. Cache-control, as defined by RFC 7234 (https://tools.ietf.org/html/rfc7234#section-5.2) (the HTTP specification), includes many different options for appropriate handling of cached data. Specifically, cache-control headers tell user agents (e.g., web browsers) how to handle the caching of server responses. For example:

- `Cache-Control: private`
- `Cache-Control: max-age=86400`

In the first example, `private` tells the user agent the information is specific to a single user and should not be cached for other users. In the second example, `max-age=86400` tells the user agent the response can be cached, but that it expires in exactly 86,400 seconds (one day).

Fastly respects cache-control headers by default, but you can also use another proxy-specific header: surrogate-control. Surrogate-control headers are similar to cache-control headers, but provide instructions to reverse proxy caches like Fastly. You can use cache-control and surrogate-control headers together. For more information about cache-control and surrogate-control headers, see our cache control tutorial (/guides/tutorials/cache-control-tutorial).

An updated example

Let's take a look at how the cache-control headers could be used in our original example, the comments API (/guides/api-caching/enabling-api-caching#an-example). Recall the API endpoint that provided a list of comments for a given article:

`GET /article/:article_id/comments`

When a user submits a comment for a given article, the response from this endpoint will be purged from the Fastly cache by the comment model (/guides/api-caching/enabling-api-caching#configuring-purge-requests). It's hard to predict when content will change. Therefore, we'd like to ensure the following:

1. If the content doesn't change, it should stay cached in Fastly for a reasonable amount of time.
2. If the content does change, it should not be cached by the client longer than it needs to be.

The goal is to ensure that API responses will reach clients in a timely manner, but we also want to ensure that clients always have the most up-to-date information. The first constraint can be solved by using the surrogate-control header, and the second constraint can be solved by using the cache-control header:

```
Surrogate-Control: max-age=86400
Cache-Control: max-age=60
```

These headers tell Fastly that it is allowed to cache the content for up to one day. In addition, the headers tell the client that it is allowed to cache the content for 60 seconds, and that it should go back to its source of truth (in this case, the Fastly cache) after 60 seconds.

Implementing cache control

Migrating APIs isn't easy, even for experienced teams. When migrating an API to Fastly, we recommend separating the task into three strategic endpoint migrations to make the process more manageable while still maintaining the validity of the API as a whole.

Preparing the API

To ensure that the API bypasses the cache during the piecewise migration, we must have every API endpoint return a specific control header:

`Cache-Control: private`

This header tells Fastly that a request to any endpoint on the API should bypass the cache and be sent directly to the origin. This will allow us to serve the API via Fastly and have it work as expected.

> **NOTE:** Modern web frameworks allow for blanket rules to be overridden by specific endpoints (for example, by the use of middlewares). Depending on how the API has been implemented, this step might be as simple as adding a single line of code.

Serving traffic with Fastly

The next step is configuring a Fastly service (/guides/api-caching/enabling-api-caching#setting-up-the-fastly-configuration) to serve the API's traffic. After you save the configuration, there will be an immediate speed improvement. This happens because Fastly's cache servers keep long-lasting connections to the API's origin servers, which reduces the latency overhead of establishing multiple TCP connections.

Migrating endpoints

Now we can implement instant purge caching for each cacheable API endpoint, one at a time. The order in which this is done depends on the API, but by targeting the slowest endpoints first, you can achieve dramatic improvements for endpoints that need them the most. Because each endpoint can be worked on independently, the engineering process is easier to manage.

Excluding endpoints

The last step is deciding which API endpoints you don't want Fastly to cache. To disable caching for endpoints, you'll need to add new conditions for the endpoints (/guides/performance-tuning/controlling-caching#conditionally-preventing-pages-from-caching). As you learned in Preparing the API, using the `Cache-Control: private` header is another option for disabling caching.

---

§ Purging API cache with surrogate keys (/guides/api-caching/purging-api-cache-with-surrogate-keys)

Fastly makes it possible for you to cache your API so you can accelerate the performance of your service-oriented architecture. Of course, caching your API is one thing - efficiently invalidating the API cache is another matter entirely. If you've already enabled API caching (/guides/api-caching/enabling-api-caching) and implemented API cache control (/guides/api-caching/implementing-api-cache-control), you've probably run into this problem, which was aptly described by Phil Karlton (http://martinfowler.com/bliki/TwoHardThings.html):

> *There are only two hard things in computer science: cache invalidation and naming things.*

This guide explains how to use the Fastly API to purge your API cache with surrogate keys (/guides/purging/getting-started-with-surrogate-keys). Surrogate keys allow you to reduce the complexity of caching an API by combining multiple cache purges into a single key-based purge.

What's a surrogate key?

Surrogate keys allow you to selectively purge related content. Using the `Surrogate-Key` header, you can "tag" an object, such as an image or a blog post, with one or more keys. When Fastly fetches an object from your origin server, we check to see if you've specified a `Surrogate-Key` header. If you have, we add the response to a list we've set aside for each of the keys.

When you want to purge all of the responses associated with a key, issue a key purge (/guides/purging/single-purges#purging-with-a-key) request and all of the objects associated with that key will be purged. This makes it possible to combine many purges into a single request. Ultimately, it makes it easier to manage categorically related data.

To learn more about surrogate keys and to see how you can integrate them into your application, see our guide on getting started with surrogate keys (/guides/purging/getting-started-with-surrogate-keys).

Example: Purging categories

To see how surrogate keys work in conjunction with an API endpoint, imagine you have an online store and an API endpoint that returns the details of a product. When a user wants to get information about a specific product, like a keyboard, the request might look like this:

```
GET /product/12345
```

If your API is using Fastly and the response is not already cached, Fastly will make a request to your API's origin server and receive a response like this:

```
HTTP/1.1 200 OK
Content-Type: text/json
Cache-Control: private
Surrogate-Control: max-age=86400
Surrogate-Key: peripherals keyboards

{id: 12345, name: "Uber Keyboard", price: "$124.99"}
```

You knew that entire product categories would occasionally need to be purged, so you thoughtfully included the `peripherals` and `keyboards` product categories as keys in the `Surrogate-Key` header. When Fastly receives a response like this, we add it to an internal map, strip out the `Surrogate-Key` header, cache the response, and then deliver it to the end user.

Now imagine that your company decides to apply a 10% discount to all peripherals. You could issue the following key purge to invalidate all objects tagged with the `peripherals` surrogate key:

```
PURGE /service/:service_id/peripherals
```

When Fastly receives this request, we reference the list of content associated with the `peripherals` surrogate key and systematically purge every piece of content in the list.

Relational dependencies

Your API can use surrogate keys to group large numbers of items that may eventually need to be purged at the same time. Consider the example presented above. The API for your online store could have surrogate keys for product types, specific sales, or manufacturers.

From this perspective, the `Surrogate-Key` header provides Fastly with information about relations and possible dependencies between different API endpoints. Wherever there's a relation between two different types of resources in an API, there might be a good reason to keep them categorized by using a surrogate key.

Example: Purging product reviews and action shots

To learn how surrogate keys can help with relational dependencies, imagine that your online store wants to allow buyers to post product reviews and "action shots" depicting the products in use. To support these new features, you'll need to change your API. First, you'll need to create a new `review` endpoint:

```
GET /review/:id
POST /review
```

Next, you'll need to create a new `action_shot` endpoint:

```
POST /product/:id/action_shot
GET /product/:id/action_shot/:shot_id
```

Since both of the new endpoints refer to specific products, they'll need to be purged when relevant product information changes. Surrogate keys are a perfect fit for this use case. You can implement them by modifying the `review` and `action_shot` to return the following header:

```
Surrogate-Key: product/:id
```

This relates each of the endpoints to a specific product in the cache (where `:id` the product's unique identifier). When the product information changes, your API issues the following a key purge:

```
PURGE /service/:service_id/product/:id
```

When Fastly receives this request, we purge each of the related endpoints at the same time.

Variations on a theme

You'll also want to consider using surrogate keys if your API has many different endpoints that all derive from a single source. Any time the source data changes, each of the endpoints associated with it will need to be purged from the cache. By associating each of the endpoints with a surrogate key, a single purge can be issued to purge them from the cache when the source changes.

Example: Purging product images

To understand how this works, imagine that your online store has an API endpoint for retrieving product images in various sizes:

```
GET /product/:id/image/:size
```

This endpoint returns an image of the appropriate `:size` (e.g., `small`, `medium`, `large`) for the product of the given `:id`. To save disk space, you opt to have the API generate each specifically sized image from a single source image using an imaging library like ImageMagick (http://www.imagemagick.org/). Since the sales and marketing team uses the API to upload new product images, you set up the endpoint to include a surrogate key:

```
Surrogate-Key: product/:id/image
```

When the PUT endpoint for uploading a product image is called, the API sends the following purge request:

```
PURGE /service/:service_id/product/:id/image
```

When Fastly receives this request, we purge all size variations of the product image.

• Guides (/guides/) > Developer's tools (/guides/devtools) > Conditions (/guides/conditions/)

§ About conditions (/guides/conditions/about-conditions)

Use conditions to specify how to edit and deploy configurations. This allows you to add logic to any basic configuration object in a service. When you assign a condition to an object in your service, you are wrapping the object in a VCL if-statement. If the condition is met, then the object is used when processing an end-user request.

Before you begin

Because of their complexity and power, be sure you're comfortable with the following before you begin using conditions (/guides/conditions/using-conditions):

- **Know how to edit and deploy configurations using the Fastly application.** If you are not familiar with basic editing using the application, see our documentation (/guides/basic-setup/working-with-services) to learn more before moving forward.
- **Understand basic C-style logical expression syntax** (e.g., basic logic, operators such as &&, and precedence). If you do not have experience with these types of expressions, refer to a basic programming guide (such as the Tizag Perl tutorial (http://www.tizag.com/perlT/)) that deals with "IF" style expressions in either C or Perl. (These languages are not directly applicable to the tutorial (/guides/conditions/using-

conditions#an-example-of-adding-conditions) because VCL is neither C nor Perl. However, the syntax is similar.)

Condition basics

Conditions are saved in your service like any other configuration object, but they can only be accessed, assigned, and manipulated (/guides/conditions/using-conditions#an-example-of-adding-conditions) through a selection in the gear-shaped menu to the right of an existing configuration object (e.g., origin servers and headers). This means that when you create a condition, you can apply it in multiple places across your configuration.

Properly configured conditions require only three parameters, making them appear deceptively simple:

- a **Name** - A human-readable identifier for the condition.
- an **Apply If** statement - The logical expression to execute in VCL to determine if condition resolves as True or False.
- the **Priority** - A number used to determine the order in which multiple conditions execute. Lower numbers execute first.

Most problems occur in the Apply If parameter because it uses actual VCL code to specify when a condition should be applied.

Working with conditions

A basic condition used by Fastly might look like this:

```
req.url ~ "^/api/"
```

This condition tells the cache server to pay attention if a request it receives begins with the string `"/api/"`. The `~` operator performs a Perl-style regular expression match with the `"^/api/"` regex and returns true if `req.url` matches. You might use this condition when you want to specifically route certain requests for your service. In this example, you might be routing requests to an API backend that exists on your website.

Matching conditions on logical expressions

Conditions can perform matches on more complicated logical expressions. For example:

- `client.ip == "127.0.0.1"`: The client requesting a resource on your service has the IP `"127.0.0.1"`
- `req.http.host ~ "example.com"`: The host header of the incoming request is `"example.com"`
- `req.request ~ "POST|PUT" && req.url ~ "^/api/articles/"`: The request is a POST or a PUT and the URL begins with `"/api/articles/"`

The `client.ip`, `req.http.host`, `req.request`, and `req.url` conditions shown above all represent configuration variables in VCL.

Operators for complex conditions

You could also get creative and create a more complex condition used by Fastly that might look like this:

```
req.http.host = "www.example.com" && (req.url !~ "^/foo" || req.url !~ "^/bar/" || req.url !~ "^/baz/")
```

This condition tells the cache server that the URL should equal `www.example.com` and the url cannot match `www.example.com/foo`, or `www.example.com/bar`, or `www.example.com/baz`. You might use this type of condition when you have multiple variables or options and want to fine-tune your results. In this example, you are indicating that you don't want URLs that contain `foo`, `bar`, or `baz` by using the following operators:

- ( ) groups your expressions and restricts alteration to part of the regex
- | | performs an alternation where each variable is checked until it finds a variable that is true
- !~ excludes any urls that include the specified variables

Where to go for more information

The Varnish Cache documentation (https://www.varnish-cache.org/docs/2.1/reference/vcl.html#variables) on variables provides a complete list of variables you can use to craft conditions (/guides/conditions/using-conditions). Keep in mind, however, some of the variables Varnish allows may not be available or may have no meaning in the context of Fastly services. For more information, see our Guide to VCL (/guides/vcl/guide-to-vcl).

---

§ Troubleshooting conditions (/guides/conditions/troubleshooting-conditions)

If you are having problems using conditions, here are some common mistakes to look for:

- **Don't put the `if ()` statement in the Apply If field of the condition window.** The actual if statement is implied. You only need to type an evaluated expression. For example: `req.url ~ "^/special/"`.
- **Don't use the `!~` (inverse regex match) to build regular expressions that exclude particular URLs.** For example, if you want to apply something to all URLs except those that start with `/admin`, the condition for this is `req.url !~ "^/admin"` and is entered in the Apply If field.

An alternative to this expression would be `!(req.url !~ "^/admin")`.

Editing conditions

The Conditions Overview on the Configuration Controls provides an summary of how conditions are used and mapped in your service. In order to edit those conditions, however, you need to access the location where those conditions are applied.

For example, if you had a condition that was being applied to a header, you wouldn't edit it via the Conditions Overview, you would edit that condition in your service's configuration as follows:

1. Click the **Content** pane on the left side and click the gear box to the right of the name of the appropriate header.
2. Select **Conditions** from the menu that appears.
3. Click **Edit** from the **Choose Condition** window that appears and begin editing the selected condition.

Helpful links

The following links provide additional information useful when working with and troubleshooting conditions:

- Fastly Help Guides (/guides/) provides basic editing information when using the application.
- Tizag Perl tutorial (http://www.tizag.com/perlT/perlif.php) explains basic C-style logical expression syntax (e.g., basic logic, operators such as &&, precedence) that users need to be familiar with when creating conditions.
- Varnish-Cache.org documentation (https://www.varnish-cache.org/docs/2.1/reference/vcl.html#variables) explains different set of VCL variables that a condition can use to determine if an object applies.
- Varnish Cache documentation on variables (https://www.varnish-cache.org/docs/2.1/reference/vcl.html#variables) describes variables that are used use to craft conditions.

---

§ Using conditions (/guides/conditions/using-conditions)

Conditions use the Varnish Control Language (VCL) (/guides/vcl/guide-to-vcl#about-varnish-and-why-fastly-uses-it) to define when an object should be applied while processing requests to a cache server. Once you understand some basics about conditions (/guides/conditions/about-conditions), use this guide to learn about when to use conditions and how to create them using the Fastly application.

About request, response, and cache conditions

Conditions can occur in various stages of caching: when a request is being processed, when a response to that request is being processed, and when a response from your origin serve arrives but before that information is (potentially) cached. Each stage of caching has access to a different set of VCL variables (as described in the Varnish-Cache.org documentation (https://www.varnish-cache.org/docs/2.1/reference/vcl.html#variables)) that can be used to create conditions.

To help distinguish conditions during these stages we group them into three types: request, response, and cache. Configuration objects also only apply in various stages of the pipeline and can have different types of conditions associated with them.

The following list of the condition types describes the objects to which conditions can apply and the VCL variable types that are available to them:

- **Request**: Conditions applied to objects while processing a request.
  - Objects: Request Settings, Backends, Headers, and Responses
  - VCL Variables: `client.*`, `server.*`, `req.*`
  - Example: `req.url ~ "^/foo/bar$"`
- **Response**: Conditions applied to objects while processing a response to a request.
  - Objects: Headers, Syslog
  - VCL Variables: `client.*`, `server.*`, `req.*`, `resp.*`
  - Example: `resp.status == 404`
- **Cache**: Conditions applied when Fastly receives a response from your origin, just before that response is (potentially) cached.
  - Objects: Cache Conditions, Headers, Gzip, Responses
  - VCL Variables: `client.*`, `server.*`, `req.*`, `beresp.*`
  - Example: `!beresp.cachable && req.url ~ "^/articles/"`

These conditions allow you to control your configuration with minimal programming.

An example of adding conditions

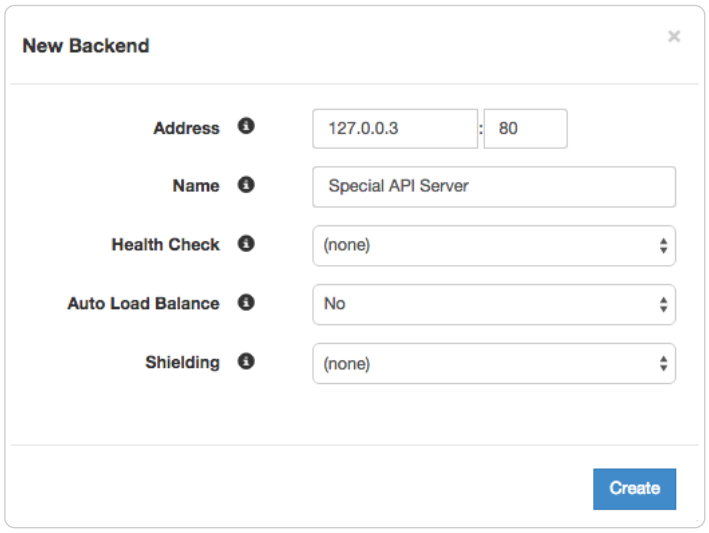
The scenario: You want to add a new origin server that handles a specific portion of your API requests. Some requests to this API must be cached differently than other requests to your API, so want to set special headers for specific types of requests. Specifically, you don't want your new origin server to cache PUT, POST, or DELETE requests because they're special for this particular API and send back extra, time dependent, meta-information in each response. And finally, you want to track the effectiveness of doing this. To accomplish all of this using conditions via the Fastly application, you would:

1. Create a new origin server to handle the special API traffic.
2. Create a new condition that tells the cache how to route requests to that origin server.
3. Create a new cache setting object and then a new condition that applies that object at the appropriate time.
4. Create a new header to track the specific type of API requests.
5. Create a new cache condition to make sure that the header is only set on specific type of request.
6. Check your work.

Create a new origin server

To create a new origin server that will handle the special API traffic, follow the steps below.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.
3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Select the **Host** pane from the list on the left.
6. In the **Backends** area click the **New** button to create a new backend for your special API traffic. The New Backend window appears.

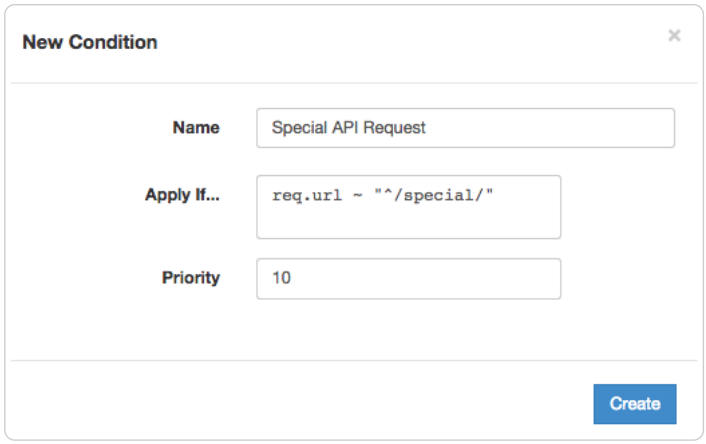


7. Fill out the **New Backend** window as follows:
   - In the **Address** field, type the address of your API server (for example, `127.0.0.3`).
   - In the **Port** field, type `80`.
   - In the **Name** field, type the name of your API server (for example, `Special API Server`).
   - From the **Auto Load Balance** menu, select **No** to exclude this server from the normal, automatic load balancing pool.
   - Leave the **Shielding** control set to its default.
8. Click **Create**. The new origin server appears in the Backends area.

Create a request condition

Once you've created a new origin serve to handle the special API traffic, tell the cache how to route requests to this origin server by creating a request condition.

1. In the **Backends** area, click the gear icon next to the name of the origin server you just created, then select **Conditions**. The Choose Conditions window appears.
2. Click the **New** button to create a new condition. The New Condition window appears.

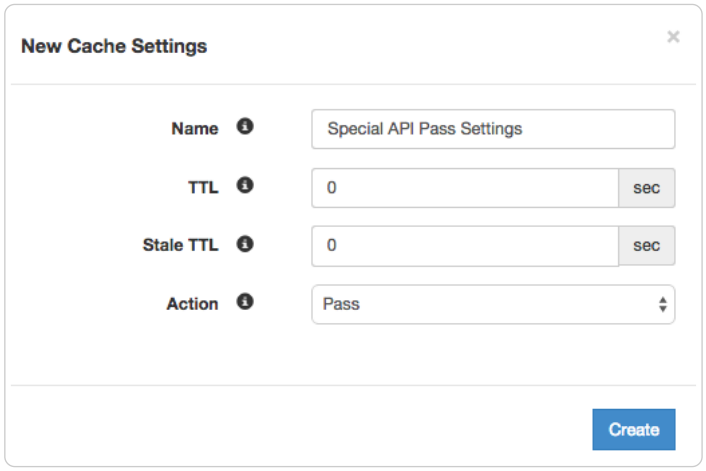


3. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a descriptive name for the new condition (for example `Special API Request`).
   - In the **Apply if** field, type the appropriate request condition that will be applied (for example, `req.url ~ "^/special/"` to address all requests related to the special API server).
   - Leave the **Priority** field set to its default value.
4. Click **Create**.

Create a cache settings object and its condition

The requests now are being properly routed to the new origin, you need to make sure that it doesn't cache any responses from PUT, POST, or DELETE requests because they are special for this particular API and send back extra, time dependent, meta-information in each response. To do this, create a cache settings object in the configuration.

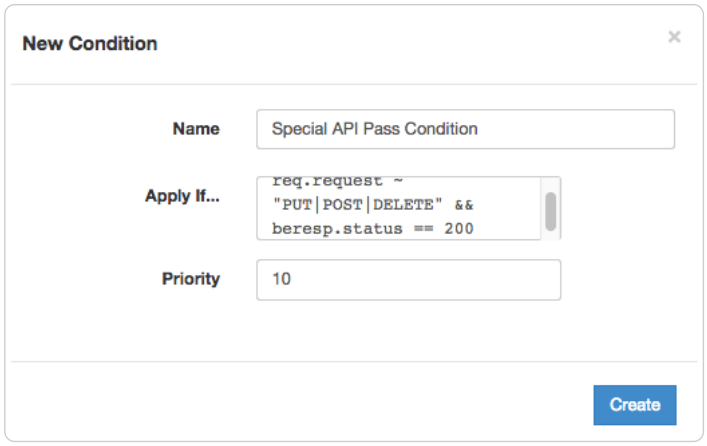
1. Click the **Settings** pane. The Settings appear.
2. In the **Cache Settings** area, click the **New** button to create a cache settings object. The New Cache Settings window appears.



3. Fill out the **New Cache Settings** window as follow:
   - In the **Name** field, type a descriptive name for the new cache settings (for example, `Special API Pass Settings`).
   - Leave the **TTL** and **Stale TTL** fields set to their default values.
   - From the **Action** menu, select **Pass**.
4. Click **Create** to create the new cache setting.

Then, create a new condition that specifies when the cache settings object should be applied.

1. In the **Cache Settings** area, click the gear icon next to the name of the cache setting you just created, then select **Conditions**. The Choose Conditions window appears.
2. Click the **New** button to create a new condition. The New Condition window appears.



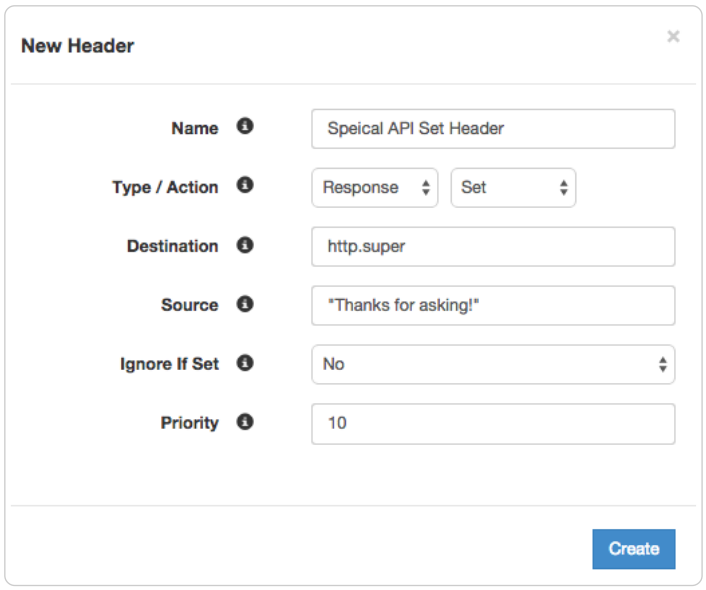
3. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a descriptive name for the new condition (for example, `Special API Pass Condition`).
   - In the **Apply if** field, type the appropriate request condition that will be applied (for example, `req.request ~ "PUT|POST|DELETE" && beresp.status == 200`).
   - Leave the **Priority** field set to its default value.
4. Click **Create** to create the new condition for the cache setting.

Create a new header and its condition

To make sure you can track the effectiveness the new API, create a new header so you can use it to gather information about the special API requests as they happen.

To set the condition:

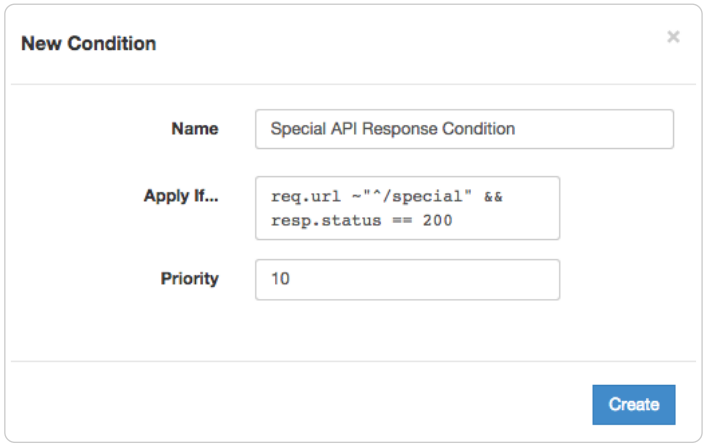
1. Click the **Content** pane. The Content controls appear.
2. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.



3. Fill out the **New Header** window as follows:
   - In the **Name** field, type a descriptive name for the new header (for example, `Special API Set Header`).
   - From the **Type/Action** menus select **Response** and **Set**.
   - In the **Destination** field, type the name of the header that will be affected by the action (for example, `http.super`).
   - In the **Source** field, type a description of the source where the content for this header comes from (for example, "Thanks for asking!").
   - Leave the **Ignore If Set** and **Priority** fields set to their default settings.
4. Click **Create** to create the new header.

Once the header is created, create an associated condition to ensure that this header is only set on that special type of request.

1. In the **Headers** area, click the gear icon next to the name of the new header you just created, then select **Conditions**. The Choose Conditions window appears.
2. Click the **New** button to create a new condition. The New Condition window appears.



3. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a descriptive name for the new condition (for example, `Special API Response Condition`).
   - In the **Apply if** field, type the appropriate request condition that will be applied (for example, `req.url ~ "^/special" && resp.status == 200`).
   - Leave the **Priority** field set to its default value.
4. Click **Create** to create the new condition for the header.

Check your work

Before deploying the application, review the generated VCL (/guides/vcl/previewing-and-testing-vcl-before-activating-it) to see how Fastly converted the objects and conditions into actual VCL. For the example show above, the VCL for the request condition appears as:

```
# Condition: Special API Request Prio: 10
if (req.url ~ "^/special/") {
  set req.backend = F_Special_API_Server;
}
#end condition
```

The cache settings and condition VCL appears as:

```
if (req.request ~ "PUT|POST|DELETE" && beresp.status == 200) {
  set beresp.ttl = 0s;
  set beresp.grace = 0s;
  return(pass);
}
```

And the new header response condition VCL appears as:

```
# Condition Special API Response Condition Prio: 10
if (req.url ~ "^/special" && resp.status == 200) {

  # Header rewrite Special API Set Header: 10
  set resp.http.super = "Thanks for asking!";
}
```

As you become more familiar with the VCL syntax and programming, look at the generated VCL to see if the configuration is doing what you think it is doing (most VCL is pretty simple once you know what the variables are referring to).

• Guides (/guides/) > Developer's tools (/guides/devtools) > Edge Dictionaries (/guides/edge-dictionaries/)

§ About Edge Dictionaries (/guides/edge-dictionaries/about-edge-dictionaries)

Fastly offers instantly updatable, global, Edge Dictionaries for use with your services (/guides/basic-setup/working-with-services).

Edge Dictionaries simplify services

Edge Dictionaries are made up of dictionary containers and the dictionary items within them. In combination, containers and items allow you to store data as key-value pairs and turn frequently repeated statements like this:

```
if (something == "value1") {
  set other = "result1";
} else if (something == "value2") {
  set other = "result2";
}
```

into a single function that acts as constant:

```
table <ID> {
  "KEY_STRING": "VALUE_STRING",
  "KEY_STRING2": "VALUE_STRING2",
  ...
}
```

Once you attach a dictionary container to a version of your service with the API and that service is activated, the data in it becomes "versionless." This means you can add to and update (/guides/edge-dictionaries/creating-and-manipulating-dictionary-items) the data an Edge Dictionary contains using a single API call at any time after it is created, without ever incrementing a service's version. You can perform lookups on the dictionary items in an Edge Dictionary using functions like `table.lookup(<ID>, "KEY_STRING")` or `table.lookup(<ID>, "KEY_STRING", "DEFAULT_VALUE_STRING")` in your configuration.

When Edge Dictionaries might be useful

- Content sharing and social media outlets updating large referrer blacklists
- Mobile advertisers validating a key to prevent cache-bust guessing
- Customers authenticating valid user keys at the edge
- Global publishers redirecting users to a specific country site based on geo-location
- Image providers performing token checks for certain objects
- Advertising technology companies blocking bad actors at edge
- Customers deploying user interface versions with simple value change via API

Limitations and considerations

When creating Edge Dictionaries (/guides/edge-dictionaries/creating-and-using-dictionaries), keep the following things in mind as you develop your service configurations:

- **Edge Dictionaries created with custom VCL cannot be manipulated using the API.** If you create a dictionary container using the API (/api/config#dictionary), you can use the API to make changes to it, and use custom VCL to interact with it. If you create a dictionary container using custom VCL, that dictionary must always be manipulated via custom VCL.
- **Dictionary containers are limited to 1000 dictionary items.** If you find your dictionary containers approaching this item limit, contact us (mailto:support@fastly.com). We may be able to help you figure out an even more efficient way to do things with your Edge Dictionaries.
- **Dictionary item keys are limited 256 characters and values are limited to 8000 characters.** Dictionary items are made up of key-value pairs with character limits. Be sure to take this into account when designing your Edge Dictionaries.
- **Dictionary item keys are case sensitive.** Dictionary item names are case sensitive. Be sure to take this into account when designing your Edge Dictionaries.
- **Audit logs don't exist for Edge Dictionary changes.** If you add, update, or remove a dictionary item, there will be no record of it. The only record of a change will exist when you compare service versions (https://www.fastly.com/blog/introducing-version-diff) to view the point at which the dictionary container was associated with the service version in the first place.
- **When you delete a dictionary container, you'll only delete it from the service version you're editing.** Dictionary containers are tied to versions and can be cloned and reverted. When using Edge Dictionaries, we want you to be able to do things like delete a dictionary container from a current version of your service in order to roll back your configuration to a previous version using a few steps as possible.
- **When you delete a dictionary container, we don't delete the dictionary items inside it.** The dictionary items in a dictionary container are versionless. When you change service versions, we want you to still be able to access the data.
- **Dictionary item deletions are permanent.** Because we don't store data, if you delete a dictionary item, the entry is gone forever from all service versions.

§ Creating and manipulating dictionary items (/guides/edge-dictionaries/creating-and-manipulating-dictionary-items)

A dictionary item is a key-value pair that makes up an entry in a dictionary container in an Edge Dictionary. Once you create an Edge Dictionary (/guides/edge-dictionaries/creating-and-using-dictionaries) and associate the dictionary container with a service, any dictionary items created will appear in your generated VCL.

For example, if you were using Edge Dictionaries to control geoIP redirects, the table would appear similar to this:

```
table geoip_redirect {
  "GB"  :  "www.example.co.uk",
  "IE"  :  "www.example.co.uk",
  "IT"  :  "www.example.com.it",
  "AU"  :  "www.example.com.au",
}
```

If you already have a dictionary container associated with an active version of your service, you can easily add, update, or delete the items in it as long as you know the `dictionary_id`.

In our geoIP example, you would find your `dictionary_id` using the following API call:

```
curl -H 'Fastly-Key: <API_key>'
https://api.fastly.com/service/<service_id>/version/<version_number>/dictionary/geoip_redirect
```

which would return this response:

```
{
  "version": <version_number>,
  "name": "geoip_redirect",
  "id": "<dictionary_id>",
  "service_id": "<service_id>"
}
```

Adding new items to dictionary

You can add new dictionary items without having to increment your service version number. For example, this API call to a geoIP table to add a new dictionary item:

```
curl -X POST -H 'Fastly-Key: <API_key>' -d 'item_key=NZ&item_value=www.example.com.au' "https://api.fastly.com/service/<service_id
>/dictionary/<dictionary_id>/item"
```

returns this response:

```
{
  "dictionary_id": "<dictionary_id>",
  "service_id": "<service_id>",
  "item_key": "NZ",
  "item_value": "www.example.com.au"
}
```

The table in the generated VCL would then be updated with the new dictionary item and look like this:

```
table geoip_redirect {
  "GB"  :  "www.example.co.uk",
  "IE"  :  "www.example.co.uk",
  "IT"  :  "www.example.com.it",
  "AU"  :  "www.example.com.au",
  "NZ"  :  "www.example.com.au",
}
```

Upserting dictionary items

You can create and update dictionary items regardless of whether or not they exist. For example, the following API call to the geoIP table to update an existing dictionary item or create it if it doesn't exist:

```
curl -X PUT -H 'Fastly-Key: <API_key>' -d 'item_value=www.example.co.aq' "https://api.fastly.com/service/<service_id>/dictionary/<d
ictionary_id>/item/AQ"
```

returns this response:

```
{
  "dictionary_id": "<dictionary_id>",
  "item_key": "AQ",
  "item_value": "www.example.co.aq",
  "service_id": "<service_id>"
}
```

The table in the generated VCL would then be updated with the new dictionary item and look like this:

```
table geoip_redirect {
  "GB"  :  "www.example.co.uk",
  "IE"  :  "www.example.co.uk",
  "IT"  :  "www.example.com.it",
  "AU"  :  "www.example.com.au",
  "NZ"  :  "www.example.com.au",
  "AQ"  :  "www.example.co.aq",
}
```

Updating dictionary items

You can also update any dictionary item without having to increment your service version number. For example, the following API call to the geoIP table to update an existing dictionary item:

```
curl -X PATCH -H 'Fastly-Key: <API_key>' -d 'item_value=www.example.co.uk' "https://api.fastly.com/service/<service_id>/dictionary/
<dictionary_id>/item/NZ"
```

returns this response:

```
{
  "dictionary_id": "<dictionary_id>",
  "item_key": "NZ",
  "item_value": "www.example.co.uk",
  "service_id": "<service_id>"
}
```

The table in the generated VCL would then be updated with the new dictionary item and look like this:

```
table geoip_redirect {
  "GB"  :  "www.example.co.uk",
  "IE"  :  "www.example.co.uk",
  "IT"  :  "www.example.com.it",
  "AU"  :  "www.example.com.au",
  "NZ"  :  "www.example.co.uk",
  "AQ"  :  "www.example.co.aq",
}
```

Batch updating dictionary items

You can update up to 1,000 dictionary items with a single API call. The following actions are available within a batch update:

- **Upsert** - Creates an item if it doesn't exist, otherwise modifies the existing one.
- **Create** - Creates a new item, but will not update an existing one.
- **Update** - Updates an existing item, but will not create a new one if it doesn't exist.
- **Delete** - Permanently deletes the item from the dictionary.

For example, to batch update existing dictionary items in the geoIP table, create a new file called `batch.json` that contains the following JSON-encoded data:

```
{
  "items": [
    {
      "op": "create",
      "item_key": "JP",
      "item_value": "www.example.co.jp"
    },
    {
      "op": "update",
      "item_key": "GB",
      "item_value": "www.example.co.uk"
    },
    {
      "op": "delete",
      "item_key": "IT"
    }
  ]
}
```

Now you can make the following API call:

```
curl -X PATCH -H 'Content-Type: application/json' -H 'Fastly-Key: <API_key>' -d @batch.json "https://api.fastly.com/service/<service_id>/dictionary/<dictionary_id>/items"
```

See the API documentation (/api/config#dictionary_item_dc826ce1255a7c42bc48eb204eed8f7f) for more information.

Deleting a dictionary item

⚠ **WARNING:** Dictionary item deletions are permanent. Fastly does not store data. If you delete a dictionary item, the entry is gone forever from all versions of your service.

To remove an item from your table, use this API call:

```
curl -X DELETE -H 'Fastly-Key: <API_key>' https://api.fastly.com/service/<service_id>/dictionary/<dictionary_id>/item/NZ
```

Unlike creation and update of dictionary items, the API call returns no response.

---

§ Creating and using Edge Dictionaries (/guides/edge-dictionaries/creating-and-using-dictionaries)

Edge Dictionaries (/guides/edge-dictionaries/about-edge-dictionaries) allow you to create logic that doesn't need to be attached to a configuration service version. Edge Dictionaries are made up of dictionary containers and dictionary items. You can use dictionary items to create and store key-value pairs. Attaching dictionary containers to a service version allows you to turn frequently repeated statements into single function statements that act as a constant.

To create an Edge Dictionary and use it within your service you need to:

1. Create an empty dictionary container in a working version of a service that's unlocked and not yet activated.
2. Activate the new version of the service you associated with the empty dictionary container.
3. Add dictionary items to the newly created dictionary container.

Once the dictionary container is created, properly associated, and filled with dictionary items, it can be called in your service.

For example, say you have a referer blacklist that changes frequently and you want to associate it with a service. Any time that service's configuration changes, especially if the configuration rolls back to a previous version, you would want the blacklisted referer domains to continue to remain with the service configuration instead of being removed. Edge Dictionaries would help you do this.

Create an empty dictionary container within a service

In order to use a dictionary container, start by creating an empty one within an unlocked version of a service.

Before an Edge Dictionary can be manipulated, its dictionary container must be associated with at least one service version that is not locked and not active so that the service becomes aware of the dictionary's existence.

For example, if you were creating a referer blacklist via the API, you would make an API call by running this command:

```
curl -X POST -H 'Fastly-Key: <API_key>' -d 'name=referer_blacklist' https://api.fastly.com/service/<service_id>/version/<version_number>/dictionary
```

which would return:

```
{
  "name": "referer_blacklist",
  "service_id": "<service_id>",
  "version": <version_number>,
  "id": "<dictionary_id>"
}
```

Activate the service associated with the dictionary container

In order for an Edge Dictionary to appear in generated VCL so it can be referred to later, the version associated with the dictionary container must be activated.

In our referer blacklist example, you would make this API call to activate service version associated with the empty dictionary container you created:

```
curl -X PUT -H 'Fastly-Key: <API_key>' https://api.fastly.com/service/<service_id>/version/<version_number>/activate
```

The response would be this:

```
{
  "number": <version_number>,
  "active": true,
  "service_id": "<service_id>"
}
```

Add dictionary items

Once the dictionary container becomes associated with the configuration of a service, you can begin populating it with dictionary items.

For example, you would use the following API call for each URL you want to add a URLs to your referer blacklist:

```
curl -X POST -H 'Fastly-Key: <API_key>' -d 'item_key=example-referer.org&item_value=true' "https://api.fastly.com/service/<service_id>/dictionary/<dictionary_id>/item"
```

The response for each URL added would look similar to this:

```
{
  "dictionary_id": "<dictionary_id>",
  "service_id": "<service_id>",
  "item_key": "example-referer.org",
  "item_value": "true"
}
```

Once the blacklisted URLs are added as items in your dictionary container, you can find them in your generated VCL by looking for a table similar to this:

```
table referer_blacklist {
  "example-referer.org": "true",
  "another-referer.net": "true",
  "sample-referer.com": "true",
}
```

Using a service to call Edge Dictionaries

When you create Edge Dictionaries via API calls, the dictionary contents aren't tied to any single version of your service.

The logic needed to *interact* with the table of information the Edge Dictionary creates, however, is always tied to a service version.

For example, adding a new referer to your blacklist requires that you specifically interact with the Edge Dictionary at some point after you create it. You could do this via API calls because its data would not require a service version deploy. The dictionary was created via API calls not via custom VCL.

Specifically, you would set the host of the referer to a header by including custom VCL like this:

```
// versioned vcl
sub vcl_recv {

  # capture host of referer into a header
  set req.http.Referer-Host = regsub(req.http.Referer, "^https?://?([^:/\s]+).*$", "\1");

  # check if referrer host is in blacklisted table
  if (table.lookup(referer_blacklist, req.http.Referer-Host)) {
    # ResponseObject: forbidden-referrer
    error 900 "Fastly Internal";
  }
  #end condition
}

sub vcl_error {

  if (obj.status == 900) {
    set obj.http.Content-Type = "";
    synthetic {""};
    return(deliver);
  }
}
```

• Guides (/guides/) > Developer's tools (/guides/devtools) > VCL (/guides/vcl/)

§ Accept-Language header VCL features (/guides/vcl/accept-language%20header-vcl-features)

Fastly provides functions in VCL to parse and normalize the `Accept-Language` header.

Language lookup

An implementation of the Lookup functionality as defined by RFC 4647, section 3.4 (http://tools.ietf.org/html/rfc4647#section-3.4).

Syntax

`accept.language_lookup(<available languages>, <default>, <priority list>)`

Argument	Explanation
`available languages`	A colon-separated list of languages to choose from. Typically the languages that the origin can provide. For example: `en:de:fr:pt:es:zh-CN`
`default`	The default language to return if none from the `priority list` match. For example: `en`
`priority list`	The `Accept-Language` header. A comma-separated list of languages, optionally accompanied by weights (q-values). For example: `pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4`

Return values

The best matching language (as per the RFC) is returned. If none are found, the default language is returned, unless a weight of zero (`q=0`) was indicated by the priority list, in which case `NULL` is returned.

Examples

```
set req.http.Normalized-Language =
  accept.language_lookup("en:de:fr:pt:es:zh-CN", "en", req.http.Accept-Language);
```

The above would result in `Normalized-Language: pt` given an `Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4` header.

`accept.language_lookup("en", "nl", "en-GB")` results in `en`, as each subtag is removed and the match retried when a tag does not match.

`accept.language_lookup("en:nl", "nl", "en-GB,nl;q=0.5")` results in `en` still, even if `nl` is a more exact match, because the q-value of nl is lower, and that has precedence. Exactness just does not come into the equation.

`accept.language_lookup("en-US:nl", "nl", "en-GB,nl;q=0.5")` results in `nl`, because subtags are not removed from the available languages, only from language tags on the priority list.

`accept.language_lookup("en-US:nl", "nl", "en-us,nl;q=0.5")` results in `en-US`, as the lookup is case insensitive.

`accept.language_lookup("en-US:nl", "nl", "en-GB,nl;q=0")` results in `NULL` (the value, not a string) since `en-GB` and `en` do not match, and `nl` (the default) is listed as unacceptable.

If `q=0` for the default language is to be ignored, the following VCL can be used:

```
set req.http.Normalized-Language =
  accept.language_lookup("en-US:nl", "nl", req.http.Accept-Language);
if (!req.http.Normalized-Language) {
  # User will get Dutch even if he doesn't want it!
  # (Because none of our languages were acceptable)
  set req.http.Normalized-Language = "nl";
}
```

Language filter (Basic)

An implementation of the Basic Filtering functionality as defined by RFC 4647, section 3.3.1 (http://tools.ietf.org/html/rfc4647#section-3.3.1).

The implementation is not exact when the wildcard tag (`*`) is used. If a wildcard is encountered and no matches have been found yet, the default is returned. If there are matches, those are returned and the remainder of the priority list is ignored.

(There is no implementation of Extended Filtering, but if you are in need you could always file a feature request with Support. :))

Syntax

`accept.language_filter_basic(<available languages>, <default>, <priority list>, <limit>)`

Argument	Explanation
`available languages`	A colon-separated list of languages choose from. Typically the languages that the origin can provide. For example: `en:de:fr:pt:es:zh-CN`
`default`	The default language to return if none from the `priority list` match. For example: `en`
`priority list`	The `Accept-Language` header. A comma-separated list of languages, optionally accompanied by weights (q-values). For example: `pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4`
`limit`	The maximum amount of languages returned.

Return values

The best matching language (as per the RFC) is returned. If none are found, the default language is returned, unless a weight of zero (`q=0`) was indicated by the priority list, in which case `NULL` is returned.

Examples

```
set req.http.Filtered-Language =
  accept.language_filter_basic("en:de:fr:pt:es:zh-CN", "en", req.http.Accept-Language, 2);
```

The above would result in `Filtered-Language: pt,en` given an `Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4` header.

`accept.language_filter_basic("en", "nl", "en-GB", 2)` results in `en`, as each subtag is removed and the match retried when a tag does not match.

`accept.language_filter_basic("en:nl", "nl", "en-GB,nl;q=0.5", 2)` results in `en,nl`, even if `nl` is a more exact match, because the q-value of nl is lower, and that has precedence. Exactness just does not come into the equation.

`accept.language_filter_basic("en-US:nl", "nl", "en-GB,nl;q=0.5", 2)` results in `nl`, because subtags are not removed from the available languages during the search.

`accept.language_filter_basic("en-US:nl", "nl", "en-us,nl;q=0.5", 2)` results in `en-US,nl`, as the lookup is case insensitive.

`accept.language_filter_basic("en-US:nl", "nl", "en-GB,nl;q=0", 2)` results in `NULL` (the value, not a string) since `en-GB` and `en` do not match, and `nl` (the default) is listed as unacceptable.

If `q=0` for the default language is to be ignored, the following VCL can be used:

```
set req.http.Filtered-Language =
  accept.language_filter_basic("en-US:nl", "nl", req.http.Accept-Language, 2);
if (!req.http.Filtered-Language) {
  # User will get Dutch even if he doesn't want it!
  # (Because none of our languages were acceptable)
  set req.http.Filtered-Language = "nl";
}
```

`accept.language_filter_basic("en:nl:de:fr", "nl", "en-GB,*;q=0.5", 2)` results in `en` and `accept.language_filter_basic("en:nl:de:fr", "nl", "*", 2)` results in `nl`.

§ Authenticating before returning a request (/guides/vcl/authenticating-before-returning-a-request)

Performing authentication before returning a request is possible if your authentication is completely header-based and you do something like the following using custom VCL (/guides/vcl/uploading-custom-vcl):

```
sub vcl_recv {

  /* unset state tracking header to avoid client sending it */
  if (req.restarts == 0) {
    unset req.http.X-Authed;
  }

  if (!req.http.X-Authed) {
    /* stash the original URL and Host for later */
    set req.http.X-Orig-URL = req.url;

    /* set the URL to what the auth backend expects */
    set req.url = "/authenticate";

    /* Auth requests wont be cached, so pass */
    return(pass);
  }

  if (req.http.X-Authed == "true") {
    /* were authed, so proceed with the request */
    /* reset the URL */
    set req.url = req.http.X-Orig-URL;

  } else {
    /* the auth backend refused the request, so 403 the client */
    error 403;
  }

#FASTLY recv

  ...etc...
}

sub vcl_deliver {

  /* if we are in the auth phase */
  if (!req.http.X-Authed) {

    /* if we got a 5XX from the auth backend, we should fail open */
    if (resp.status >= 500 && resp.status < 600) {
      set req.http.X-Authed = "true";
    }

    if (resp.status == 200) {

      /* the auth backend responded with 200, allow the request and restart */
      set req.http.X-Authed = "true";
    } else if (resp.status == 401) {

      return(deliver);

    } else {

      /* the auth backend responded with non-200, deny the request and restart */
      set req.http.X-Authed = "false";
    }

    restart;
  }

#FASTLY deliver

  ...etc...
}
```

REMEMBER: Change `"/authenticate"` to whatever your authentication end point is.

If you feel like you can cache the authentication, then add the appropriate headers to the hash in `vcl_hash` and `return(lookup)` instead of `(pass)`.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

§ Conditionally changing a URL (/guides/vcl/conditionally-changing-a-url)

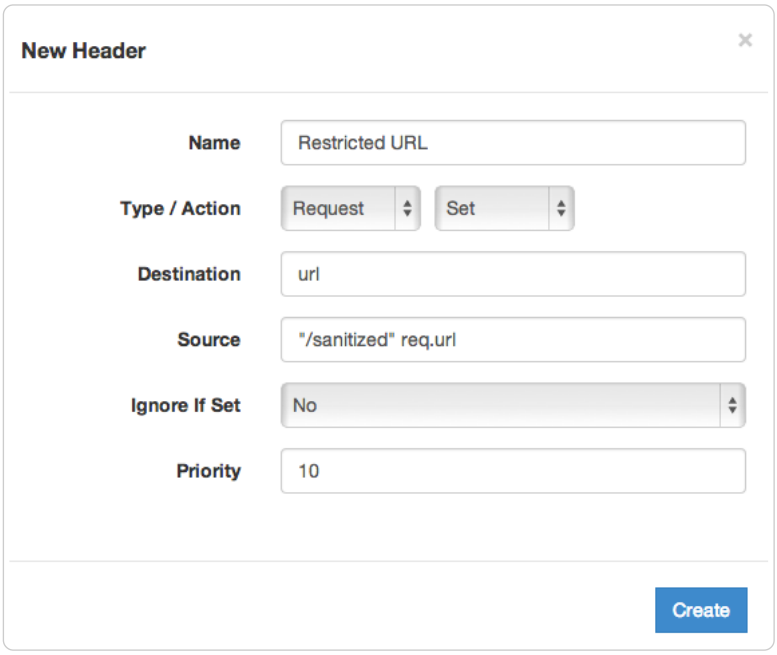
To conditionally change a URL based on the domain, include VCL that looks something like this:

```
if (req.http.host ~ "^restricted") {
  set req.url = "/sanitized" req.url;
}
```

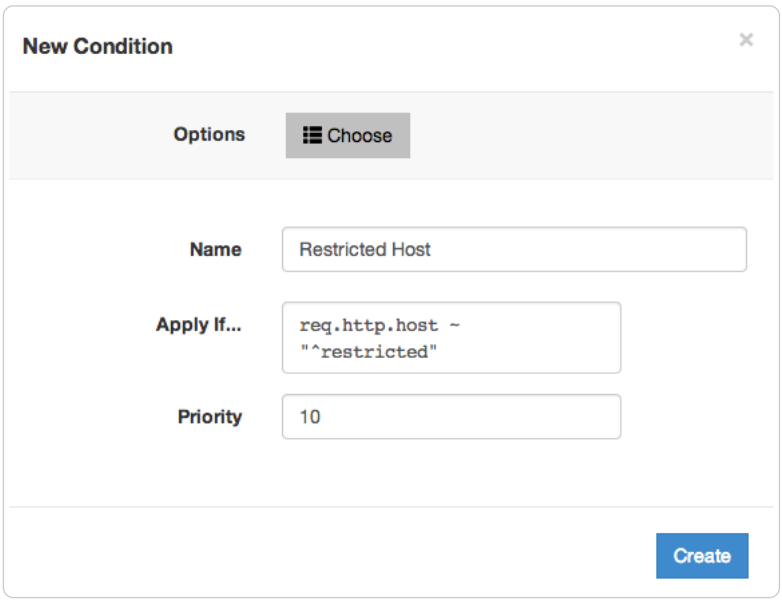
If you have shielding enabled, however, add the following code instead to avoid rewriting the URL twice:

```
if (req.http.host ~ "^restricted" && req.url !~ "^/sanitized") {
  set req.url = "/sanitized" req.url;
}
```

In Fastly's user interface, this VCL would be the equivalent of creating a new Header (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses) in the Content pane of your control panel:



and then creating a request condition that restricts connections to that host:



§ Creating location-based tagging (/guides/vcl/creating-location-

based-tagging)

You can set custom HTTP headers in your varnish configuration (VCL) based on the variables we expose. Use the GeoIP features we have built into Varnish to create location-based tagging. We provide a list of geographic information based on a client's IP address. For a complete list of available GeoIP variables, read about which GeoIP features are accessible to VCL (/guides/vcl/geoip-related-vcl-features).

In the example below, an HTTP header Fastly-GeoIP-CountryCode is created with the two letter country code of the client's IP address.

```
sub vcl_recv {
  ...
  set req.http.fastly-GeoIP-CountryCode = geoip.country_code;
  ...
}
```

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

§ Cryptographic- and hashing-related VCL functions (/guides/vcl/cryptographic-and-hashing-related-vcl-functions)

Fastly provides several functions in VCL (/guides/vcl/) for cryptographic- and hashing-related purposes. It is based very heavily on Kristian Lyngstøl's digest vmod (https://github.com/varnish/libvmod-digest) for Varnish 3 (which means you can also refer to that documentation for more detail).

Functions

Function	Description
`digest.hmac_md5(<key>,<message>)`	Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using MD5. Returns a hex-encoded string prepended with 0x.
`digest.hmac_md5_base64(<key>, <message>)`	Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using MD5. Returns a base64-encoded (https://en.wikipedia.org/wiki/Base64) string.
`digest.hmac_sha1(<key>, <message>)`	Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a hex-encoded string prepended with 0x.
`digest.hmac_sha1_base64(<key>, <message>)`	Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a base64-encoded (https://en.wikipedia.org/wiki/Base64) string.
`digest.hmac_sha256(<key>, <message>))`	Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a hex-encoded string prepended with 0x.
`digest.hmac_sha256_base64(<key>, <message>)`	Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a base64-encoded (https://en.wikipedia.org/wiki/Base64) string.
`digest.base64(<string>)`	Base64 (https://en.wikipedia.org/wiki/Base64) encoding. Returns the base64-encoded version of the input-string.
`digest.base64url(<string>)`	Base64 (https://en.wikipedia.org/wiki/Base64#URL_applications) encoding. Returns the base64-encoded version of the input-string. Replaces +/ with -_ for url safety.
`digest.base64url_nopad(<string>)`	Base64 (https://en.wikipedia.org/wiki/Base64#URL_applications) encoding. Returns the base64-encoded version of the input-string. Replaces +/ with -_ for url safety. Has no length padding (https://en.wikipedia.org/wiki/Base64#Padding).
`digest.base64_decode(<string>)`	Decode Base64. Returns a string.

`digest.base64url_decode(<string>)`	Decode Base64 with url safe characters in. Returns a string.
`digest.base64url_nopad_decode(<string>)`	Decode Base64 with url safe characters. Returns a string. Identical to base64_url_decode.
`digest.hash_sha1(<string>)`	Use the SHA1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.
`digest.hash_sha224(<string>)`	Use the SHA224 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.
`digest.hash_sha256(<string>)`	Use the SHA256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.
`digest.hash_sha384(<string>)`	Use the SHA384 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.
`digest.hash_sha512(<string>)`	Use the SHA512 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.
`digest.hash_gost(<string>)`	Use the Gost (https://en.wikipedia.org/wiki/GOST_(hash_function)) hash function. Returns a hex-encoded string.
`digest.hash_md2(<string>)`	Use the MD2 (https://en.wikipedia.org/wiki/MD2_(cryptography)) hash. Returns a hex-encoded string.
`digest.hash_md4(<string>)`	Use the MD4 (https://en.wikipedia.org/wiki/MD4) hash. Returns a hex-encoded string.
`digest.hash_md5(<string>)`	Use the MD5 (https://en.wikipedia.org/wiki/MD5) hash. Returns a hex-encoded string.
`digest.hash_crc32(<string>)`	Use a 32 bit Cyclic Redundancy Checksum (https://en.wikipedia.org/wiki/CRC32). Returns a hex-encoded string.
`digest.hash_crc32b(<string>)`	A reversed CRC32 (for compatibility with some PHP applications (http://php.net/manual/en/function.hash-file.php#104836)). Returns a hex-encoded string.
`digest.hash_adler32(<string>)`	Use the Adler32 (https://en.wikipedia.org/wiki/Adler32) checksum algorithm. Returns a hex-encoded string.
`digest.hash_haval128(<string>)`	Use the 128 bit Haval (https://en.wikipedia.org/wiki/HAVAL) hash function. Returns a hex-encoded string.
`digest.hash_haval160(<string>)`	Use the 160 bit Haval (https://en.wikipedia.org/wiki/HAVAL) hash function. Returns a hex-encoded string.
`digest.hash_haval192(<string>)`	Use the 192 bit Haval (https://en.wikipedia.org/wiki/HAVAL) hash function. Returns a hex-encoded string.
`digest.hash_haval224(<string>)`	Use the 224 bit Haval (https://en.wikipedia.org/wiki/HAVAL) hash function. Returns a hex-encoded string.
`digest.hash_haval256(<string>)`	Use the 256 bit Haval (https://en.wikipedia.org/wiki/HAVAL) hash function. Returns a hex-encoded string.
`digest.hash_ripemd128(<string>)`	Use the 128 bit RIPEMD (https://en.wikipedia.org/wiki/RIPEMD) hash function. Returns a hex-encoded string.
`digest.hash_ripemd160(<string>)`	Use the 160 bit RIPEMD (https://en.wikipedia.org/wiki/RIPEMD) hash function. Returns a hex-encoded string.
`digest.hash_ripemd256(<string>)`	Use the 256 bit RIPEMD (https://en.wikipedia.org/wiki/RIPEMD) hash function. Returns a hex-encoded string.
`digest.hash_ripemd320(<string>)`	Use the 320 bit RIPEMD (https://en.wikipedia.org/wiki/RIPEMD) hash function. Returns a hex-encoded string.
`digest.hash_tiger(<string>)`	Use the standard, 192 bits Tiger (https://en.wikipedia.org/wiki/Tiger_(cryptography)) hash function. Returns a hex-encoded string.
`digest.hash_tiger128(<string>)`	Use the truncated 128 bit Tiger (https://en.wikipedia.org/wiki/Tiger_(cryptography)) hash function. Returns a hex-encoded string.
`digest.hash_tiger160(<string>)`	Use the truncated 160 bit Tiger (https://en.wikipedia.org/wiki/Tiger_(cryptography)) hash function. Returns a hex-encoded string.
`digest.hash_snefru128(<string>)`	Use the 256 bit Snefru (https://en.wikipedia.org/wiki/Snefru) hash function. Returns a hex-encoded string.

`digest.hash_snefru256(<string>)`	Use the 128 bit Snefru (https://en.wikipedia.org/wiki/Snefru) hash function. Returns a hex-encoded string.
`digest.hash_whirlpool(<string>)`	Use the Whirlpool (https://en.wikipedia.org/wiki/Whirlpool_(cryptography)) hash function. Returns a hex-encoded string.
`digest.rsa_verify(ID hash_method, STRING_LIST public_key, STRING_LIST payload, STRING_LIST digest [, ID base64_method ])`	A boolean function that returns true if the RSA digest using `public_key` of `payload` matches `digest`. The `hash_method` parameter selects the digest function to use. It can be `sha256`, `sha384`, `sha512`, or `default` (`default` is equivalent to `sha256`). The `base64_method` parameter is optional. It can be `standard`, `url`, `url_nopad`, or `default` (`default` is equivalent to `url_nopad`).
`digest.secure_is_equal(STRING_LIST s1, STRING_LIST s2)`	A boolean function that returns true if s1 and s2 are equal. The comparison is done in constant time to defend against timing attacks.
`digest.time_hmac_md5(<base64 encoded key>, <interval>, <offset>)`	Time based One Time Password using MD5. Returns base64 encoded output.
`digest.time_hmac_sha1(<base64 encoded key>, <interval>, <offset>)`	Time based One Time Password using SHA1. Returns base64 encoded output.
`digest.time_hmac_sha256(<base64 encoded key>, <interval>, <offset>)`	Time based One Time Password using SHA256. Returns base64 encoded output.

Notes

In base64 decoding, the output theoretically could be in binary but is interpreted as a string. So if the binary output contains '\0' then it could be truncated.

The time based One-Time Password algorithm initializes the HMAC using the key and appropriate hash type. Then it hashes the message

`(<time now in seconds since UNIX epoch> / <interval>) + <offset>`

as a 64bit unsigned integer (little endian) and base64 encodes the result.

Examples

One-Time Password Validation (Token Authentication)

Use this to validate tokens with a URL format like the following:

`http://cname-to-fastly/video.mp4?6h2YUl1CB4C50SbkZ0E6U3dZGjh+84dz3+Zope2Uhik=`

Example implementations for token generation in various languages can be found in GitHub (https://github.com/fastly/token-functions).

Example VCL

```
sub vcl_recv {
#FASTLY recv

  set req.http.token = regsub(req.url, ".*\?(.*)$", "\1");
  if (req.http.token != digest.time_hmac_sha256("RmFzdGx5IFRva2VuIFRlc3Q=", 60, 0) &&
  req.http.token != digest.time_hmac_sha256("RmFzdGx5IFRva2VuIFRlc3Q=", 60, -1)) {
    error 403;
  }
}
```

Signature

```
set resp.http.x-data-sig = digest.hmac_sha256("secretkey",resp.http.x-data);
```

Base64 decoding

A snippet like this in `vcl_error` would set the response body to the value of the request header field named `x-parrot` after base64-decoding the value:

```
synthetic digest.base64_decode(req.http.x-parrot);
```

However, if the base64-decoded string contains a NUL byte (0x00), then that byte and any bytes following it will not be included in the response.

Keep that in mind if you intend to send a synthetic response that contains binary data. There is currently no way to send a synthetic response containing a NUL byte.

§ Custom responses that don't hit origin servers (/guides/vcl/custom-responses-that-dont-hit-origin-servers)

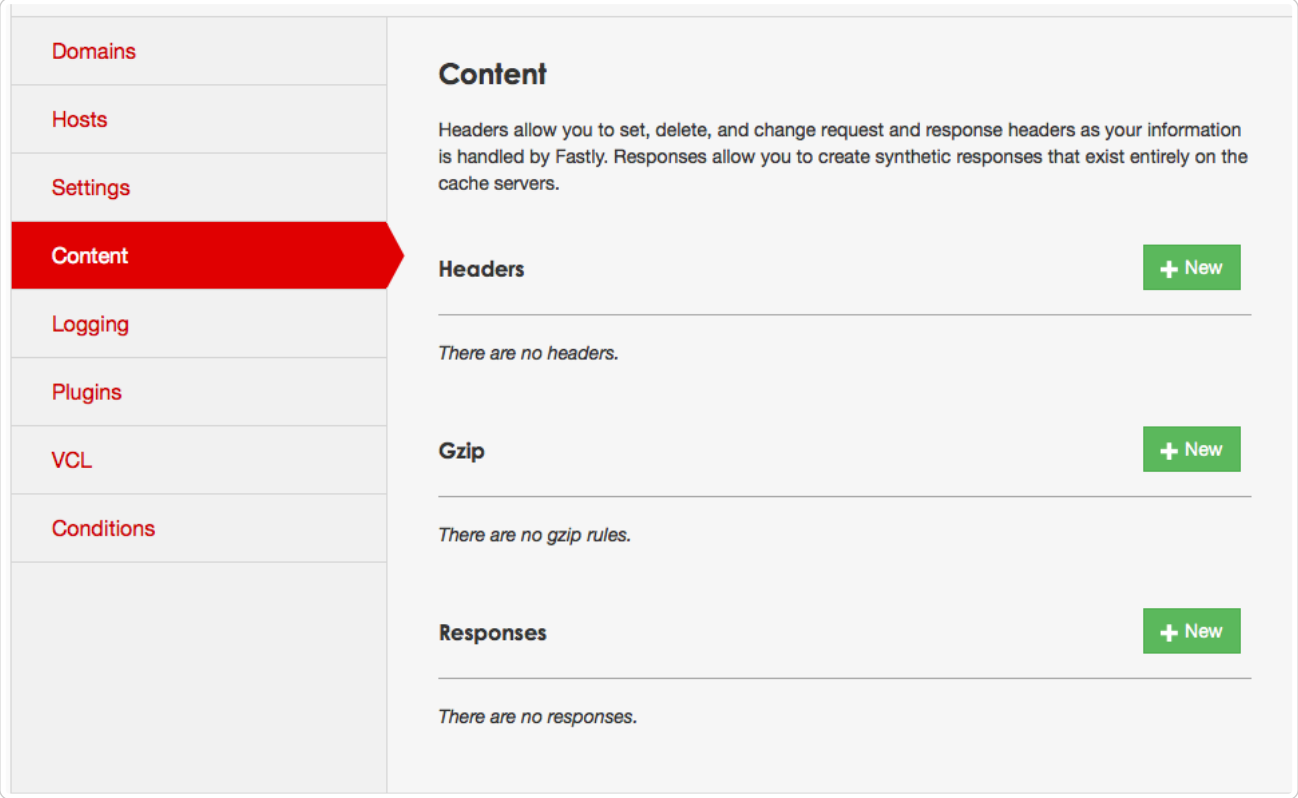
Fastly can send custom responses for certain requests that you don't want to hit your origin servers. For example, if you wanted to restrict caching to a URL subtree that contains images and scripts, you could configure Fastly to return a `HTTP 404 Not Found` response to requests for anything other than `/Content/*` or `/Scripts/*`. To illustrate how to implement this example, we'll show you how to create a response and corresponding request condition.

Creating the response

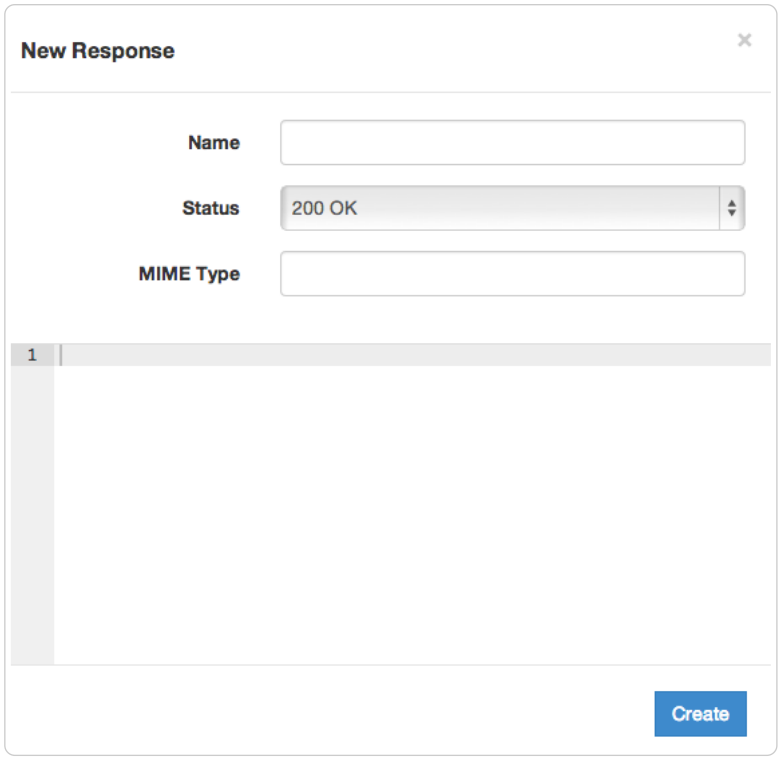
Follow these instructions to create a new response for your service:

1. Log in to the Fastly application.
2. Click the **configure** tab (wrench icon).

3. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for the selected service appear.
4. Click **Content** from the section list on the left.



5. In the **Responses** area at the bottom of the page, click the **New** button. The New Response window appears.

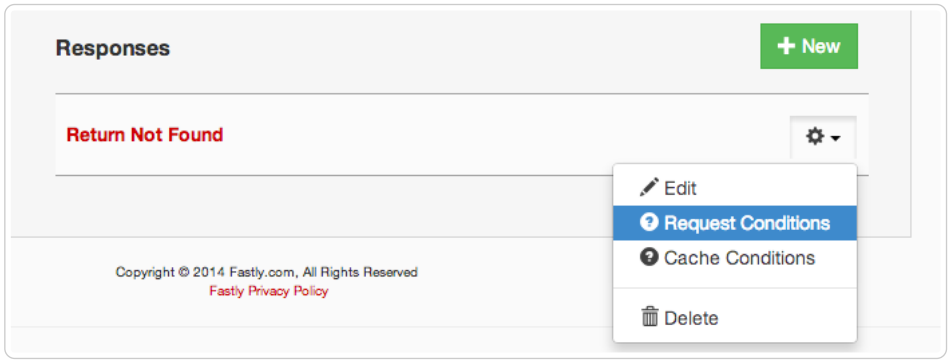


6. Fill out the **New Response** fields as follows:
   - In the **Name** field, type a human-readable name for the response.
   - From the **Status** field, select an HTTP code to return to the client.
   - In the **MIME Type** field, type the MIME type of the response.
   - In the **Content** field, type the plaintext or HTML content to return to the client.
7. Click the **Create** button to create the response.

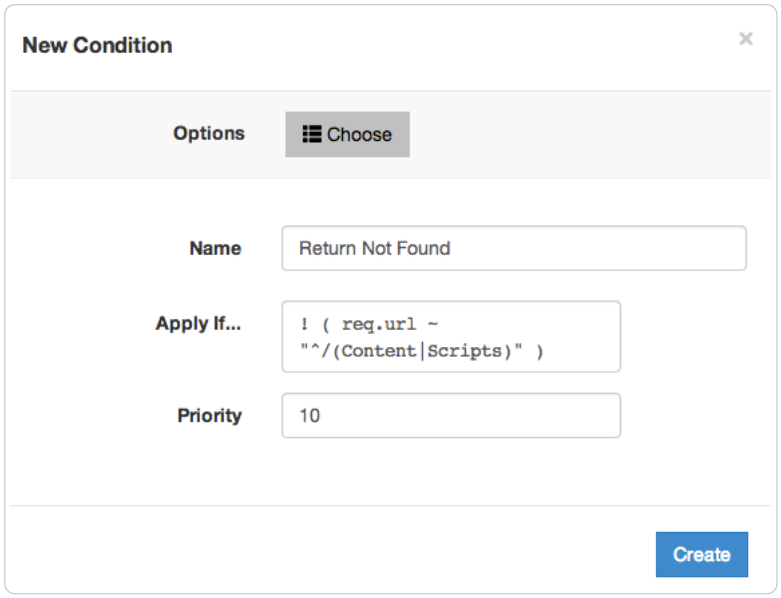
Creating the request condition

Follow these instructions to attach a request condition to the response you just created:

1. Click the gear icon next to the response that you just created and select **Request Conditions**.



The New Condition window appears.



2. Fill out the **New Condition** fields as follows:
   - In the **Name** field, type a human-readable name for the condition.
   - In the **Apply If** field, type the request condition you want inserted into a VCL if statement. See the examples below for more information.
   - In the **Priority** field, type the priority for the request condition. Leave as default if this is the only request condition.
3. Click the **Create** button to create the condition.

To activate the new response and request condition, deploy the version of the service you are editing.

Example request conditions

Respond only if URLs don't match a certain mask, in this case `/Content/*` or `/Scripts/*`:

```
! (req.url ~ "^/(Content|Scripts)/")
```

Respond only if URLs match `/secret/*` or are Microsoft Word or Excel documents (`*.doc` and `*.xls` file extensions):

```
! (req.url ~ "^/secret/" || req.url ~ "\.(xls|doc)$")
```

Ignore POST and PUT HTTP requests:

```
req.request == "POST" || req.request == "PUT"
```

Deny spider or crawler with `user-agent` `"annoying_robot"`:

```
req.http.user-agent ~ "annoying_robot"
```

Prevent a specific IP from connecting, in this case the IP `225.0.0.1`:

```
client.ip == "225.0.0.1"
```

Use geographic variables (/guides/vcl/geoip-related-vcl-features) to block traffic from a specific location, in this case China:

```
geoip.country_code == "CN"
```

Match `client.ip` against a CIDR range, such as `240.24.0.0/16` (this requires first creating an ACL object in VCL (/guides/vcl/using-access-control-lists)):

```
client.ip ~ ipRangeObject
```

§ Date- and time-related VCL features (/guides/vcl/date-and-time-related-vcl-features)

By default VCL includes the `now` variable, which provides the current time (for example, `Wed, 17 Sep 2025 23:19:06 GMT`). Fastly adds several new Varnish variables and functions (/guides/vcl/) that allow more flexibility when dealing with dates and times.

Variables

The following variables have been added:

Name	Location	Description
`now`	all	The current time in RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt) format.
`now.sec`	all	Like the `now` variable, but in seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).
`time.start`	all	The time the request started, using RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt).
`time.start.sec`	all	The time the request started in seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).
`time.start.msec`	all	The time the request started in milliseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).
`time.start.msec_frac`	all	The time the request started in milliseconds since the last whole second.
`time.start.usec`	all	The time the request started in microseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).
`time.start.usec_frac`	all	The time the request started in microseconds since the last whole second
`time.end`	deliver, log	The time the request ended, using RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt). Also useful for strftime.
`time.end.sec`	deliver, log	The time the request ended in seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).
`time.end.msec`	deliver, log	The time the request ended in milliseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).
`time.end.msec_frac`	deliver, log	The time the request started in milliseconds since the last whole second.
`time.end.usec`	deliver, log	The time the request ended in microseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).
`time.end.usec_frac`	deliver, log	The time the request started in microseconds since the last whole second.
`time.elapsed`	deliver, log	The time since the request start, using RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt). Also useful for strftime.
`time.elapsed.sec`	deliver, log	The time since the request start in seconds.
`time.elapsed.msec`	deliver, log	The time since the request start in milliseconds.
`time.elapsed.msec_frac`	deliver, log	The time the request started in milliseconds since the last whole second.
`time.elapsed.usec`	deliver, log	The time since the request start in microseconds.
`time.elapsed.usec_frac`	deliver, log	The time the request started in microseconds since the last whole second
`time.to_first_byte`	deliver, log	The time interval since the request started up to the point before the `vcl_deliver` function ran. When used in a string context, an RTIME variable like this one will be formatted as a number in seconds with 3 decimal digits of precision. In `vcl_deliver` this interval will be very close to `time.elapsed`. In `vcl_log`, the difference between `time.elapsed` and `time.to_first_byte` will be the time that it took to send the response body.

Functions

The following functions have been added:

Name	Description
`time.hex_to_time(<divider>, <hextime>)`	Takes a hexadecimal string value, divides by `divider` and interprets the result as seconds since UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).

`time.add(<time>, <offset>)`	Adds `offset` to `time`.
`time.sub(<time>, <offset>)`	Subtracts `offset` from `time`.
`time.is_after(<time1>, <time2>)`	Returns `TRUE` if `time1` is after `time2`. (Normal timeflow and causality required.)
`strftime(<format>, <time>)`	Formats a time to a string. This uses standard POSIX strftime formats (http://www.unix.com/man-page/FreeBSD/3/strftime/).
`std.time(<string_to_parse>, <fallback_value>)`	Converts a string to a time variable. The following string formats are supported: `Sun, 06 Nov 1994 08:49:37 GMT`, `Sunday, 06-Nov-94 08:49:37 GMT`, `Sun Nov  6 08:49:37 1994`, `784111777.00`, `784111777`. Useful because time variables are needed as arguments for functions like `time.add` and `strftime`.
`std.integer2time(<seconds_since_epoch>)`	Converts an integer to a time variable. To use a string, use `std.atoi`.

Examples

The following examples illustrate how to use the variables and functions.

★ **TIP:** Regular strings ("short strings") in VCL use `%xx` escapes (percent encoding) for special characters, which would conflict with the `%` used in the strftime format. For the strftime examples, we use VCL "long strings" `{"..."}`, which do not use the `%xx` escapes. Alternatively, you could use `%25` for each `%`.

time.hex_to_time, time.add, and time.is_after

```
if (time.is_after(time.add(now, 10m), time.hex_to_time(1, "d0542d8"))) {
  ...
}
```

strftime

```
set resp.http.Now = strftime({"%Y-%m-%d %H:%M"}, now)

set resp.http.Start = strftime({"%a, %d %b %Y %T %z"}, time.start)
```

std.time and std.integer2time

```
set resp.http.X-Seconds-Since-Modified = strftime({"%s"}, time.sub(now, std.time(resp.http.Last-Modified, now)));

std.integer2time(std.atoi("1445445162"));
```

Comparison operators like `> < >= <= == !=` do not work with `std.time` or `std.integer2time`. Instead, you can compare two times using something similar to this:

```
if (time.is_after(now, std.integer2time(std.atoi("1445445162")))) {
  # do something
}
```

§ Delivering different content to different devices (/guides/vcl/delivering-different-content-to-different-devices)

The easiest way to deliver different content based on the device being used is to rewrite the URL of the request based on what the user agent is. We've written an article that describes how to change the URL based on conditions (/guides/vcl/conditionally-changing-a-url) using our user interface but in pure VCL it would look something like this:

```
sub vcl_recv {
  if (req.http.User-Agent ~ "(?i)ip(hone|od)") {
    set req.url = "/mobile" req.url;
  } elsif (req.http.User-Agent ~ "(?i)ipad") {
    set req.url = "/tablet" req.url;
  }
#FASTLY recv
}
```

Obviously the code fragment above doesn't contain a comprehensive list of mobile and tablet devices. Google has an official blog post (https://webmasters.googleblog.com/2011/03/mo-better-to-also-detect-mobile-user.html) on detecting Android mobile versus tablet and this VCL fragment (https://github.com/varnishcache/varnish-devicedetect/blob/master/devicedetect.vcl) from Varnish Software can detect several different types of devices quite reliably, although it doesn't include Windows mobile and tablet, Blackberry Playbook, and the Kindle user agents.

The most comprehensive device detection routine we've seen so far is this one:

```
# based on https://github.com/varnish/varnish-devicedetect/blob/master/devicedetect.vcl
sub detect_device {
  unset req.http.X-UA-Device;
  unset req.http.X-UA-Vendor;

  set req.http.X-UA-Device = "desktop";
  set req.http.X-UA-Vendor = "generic";

  # Handle that a cookie or url param may override the detection altogether
  if (req.url ~ "[&|?]device_force=([^&\s]+)") {
    set req.http.X-UA-Device = regsub(req.url, ".*[&|?]device_force=([^&\s]+).*", "\1");
  } elsif (req.http.Cookie ~ "(?i)X-UA-Device-force") {
    # ;?? means zero or one ;, non-greedy to match the first
    set req.http.X-UA-Device = regsub(req.http.Cookie, "(?i).*X-UA-Device-force=([^;]+);??.*", "\1");
    # Clean up our mess in the cookie header
    set req.http.Cookie = regsuball(req.http.Cookie, "(^|; ) *X-UA-Device-force=[^;]+;? *", "\1");
    # If the cookie header is now empty, or just whitespace, unset it
    if (req.http.Cookie ~ "^ *$") { unset req.http.Cookie; } # "$ # stupid syntax highlighter
  } else {
    if (req.http.User-Agent ~ "(?i)(ads|google|bing|msn|yandex|baidu|ro|career|)bot" ||
      req.http.User-Agent ~ "(?i)(baidu|jike|symantec)spider" ||
      req.http.User-Agent ~ "(?i)scanner" ||
      req.http.User-Agent ~ "(?i)(web)crawler") {
      set req.http.X-UA-Device = "bot";
    } elsif (req.http.User-Agent ~ "(?i)ipad") {
      set req.http.X-UA-Device = "tablet";
      set req.http.X-UA-Vendor = "apple";
    } elsif (req.http.User-Agent ~ "(?i)ip(hone|od)") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "apple";
    # how do we differ between an android phone and an android tablet?
    #   http://stackoverflow.com/questions/5341637/how-do-detect-android-tablets-in-general-useragent
    #   http://googlewebmastercentral.blogspot.com/2011/03/mo-better-to-also-detect-mobile-user.html
    } elsif (req.http.User-Agent ~ "(?i)android.*(mobile|mini)") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "android";
    # android 3/honeycomb was just about tablet-only, and any phones will probably handle a bigger page layout
    } elsif (req.http.User-Agent ~ "(?i)android") {
      set req.http.X-UA-Device = "tablet";
      set req.http.X-UA-Vendor = "android";
    # see http://my.opera.com/community/openweb/idopera/
    } elsif (req.http.User-Agent ~ "Opera Mobi") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "android";
    } elsif (req.http.User-Agent ~ "PlayBook; U; RIM Tablet") {
      set req.http.X-UA-Device = "tablet";
      set req.http.X-UA-Vendor = "blackberry";
    } elsif (req.http.User-Agent ~ "hp-tablet.*TouchPad") {
      set req.http.X-UA-Device = "tablet";
      set req.http.X-UA-Vendor = "hp";
    } elsif (req.http.User-Agent ~ "Kindle/3") {
      set req.http.X-UA-Device = "tablet";
      set req.http.X-UA-Vendor = "kindle";
    } elsif (req.http.User-Agent ~ "Mobile.+Firefox") {
      set req.http.X-UA-Device = "mobile";
      set req.http.X-UA-Vendor = "firefoxos";
    } elsif (req.http.User-Agent ~ "^HTC") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "htc";
    } elsif (req.http.User-Agent ~ "Fennec") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "fennec";
    } elsif (req.http.User-Agent ~ "IEMobile") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "microsoft";
    } elsif (req.http.User-Agent ~ "BlackBerry" || req.http.User-Agent ~ "BB10.*Mobile") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "blackberry";
    } elsif (req.http.User-Agent ~ "GT-.*Build/GINGERBREAD") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "android";
    } elsif (req.http.User-Agent ~ "SymbianOS.*AppleWebKit") {
      set req.http.X-UA-Device = "smartphone";
      set req.http.X-UA-Vendor = "symbian";
    } elsif (req.http.User-Agent ~ "(?i)symbian" ||
      req.http.User-Agent ~ "(?i)^sonyericsson" ||
```

```
      req.http.User-Agent ~ "(?i)^nokia" ||
      req.http.User-Agent ~ "(?i)^samsung" ||
      req.http.User-Agent ~ "(?i)^lg" ||
      req.http.User-Agent ~ "(?i)bada" ||
      req.http.User-Agent ~ "(?i)blazer" ||
      req.http.User-Agent ~ "(?i)cellphone" ||
      req.http.User-Agent ~ "(?i)iemobile" ||
      req.http.User-Agent ~ "(?i)midp-2.0" ||
      req.http.User-Agent ~ "(?i)u990" ||
      req.http.User-Agent ~ "(?i)netfront" ||
      req.http.User-Agent ~ "(?i)opera mini" ||
      req.http.User-Agent ~ "(?i)palm" ||
      req.http.User-Agent ~ "(?i)nintendo wii" ||
      req.http.User-Agent ~ "(?i)playstation portable" ||
      req.http.User-Agent ~ "(?i)portalmmm" ||
      req.http.User-Agent ~ "(?i)proxinet" ||
      req.http.User-Agent ~ "(?i)sonyericsson" ||
      req.http.User-Agent ~ "(?i)symbian" ||
      req.http.User-Agent ~ "(?i)windows\ ?ce" ||
      req.http.User-Agent ~ "(?i)winwap" ||
      req.http.User-Agent ~ "(?i)eudoraweb" ||
      req.http.User-Agent ~ "(?i)htc" ||
      req.http.User-Agent ~ "(?i)240x320" ||
      req.http.User-Agent ~ "(?i)avantgo") {
      set req.http.X-UA-Device = "mobile";
    }
  }
}
```

§ GeoIP-related VCL features (/guides/vcl/geoip-related-vcl-features)

Fastly exposes a number of geographic variables for you to take advantage of inside VCL (/guides/vcl/guide-to-vcl). These are based on the MaxMind IP Geolocation database and appear as follows:

Variable	Description
`geoip.latitude`	The latitude associated with the IP address.
`geoip.longitude`	The longitude associated with the IP address.
`geoip.city`	The city or town name associated with the IP address. See MaxMind's list of cities (http://www.maxmind.com/GeoIPCity-534-Location.csv) for a list of all the possible return values. Their list is updated on a regular basis. The returned string uses the character encoding ISO-8859-1 (latin1).
`geoip.city.latin1`	An alias of `geoip.city`. The returned string uses the character encoding ISO-8859-1 (latin1).
`geoip.city.utf8`	The string from `geoip.city` encoded using the UTF-8 character encoding.
`geoip.city.ascii`	The string from `geoip.city` encoded using the ASCII character encoding (an ASCII approximation of the original string with diacritics removed).
`geoip.continent_code`	A two-character code representing the continent associated with the IP address. Possible codes are: **AF** - Africa, **AS** - Asia, **EU** - Europe, **NA** - North America, **OC** - Oceania, **SA** - South America.
`geoip.country_code`	A two-character ISO 3166-1 (https://en.wikipedia.org/wiki/ISO_3166-1) country code for the country associated with the IP address. In addition to the standard codes, we may also return one of the following: **A1** - an anonymous proxy (http://dev.maxmind.com/faq/category/geoip-general/#anonproxy), **A2** - a satellite provider (http://dev.maxmind.com/faq/category/geoip-general/#satellite), **EU** - an IP in a block used by multiple European (http://dev.maxmind.com/faq/category/geoip-general/#euapcodes) countries, **AP** - an IP in a block used by multiple Asia/Pacific region (http://dev.maxmind.com/faq/category/geoip-general/#euapcodes) countries. Note: The **US** country code is returned for IP addresses associated with overseas US military bases.
`geoip.country_code3`	The same as `country_code`; however, it returns a ISO 3166-1 alpha-3 (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3) three-character code.
`geoip.country_name`	The country name associated with the IP address. The returned string uses the character encoding ISO-8859-1 (latin1).
`geoip.country_name.ascii`	The string from `geoip.country_name` encoded using the ASCII character encoding (an ASCII approximation of the original string with diacritics removed).

`geoip.country_name.latin1`	An alias of `geoip.country_name`. The returned string uses the character encoding ISO-8859-1 (latin1).
`geoip.country_name.utf8`	The string from `geoip.country_name` encoded using the UTF-8 character encoding.
`geoip.postal_code`	The postal code associated with the IP address. These are available for some IP addresses in Australia, Canada, France, Germany, Italy, Spain, Switzerland, United Kingdom, and the US. We return the first 3 characters for Canadian postal codes. We return the first 2-4 characters (outward code) for postal codes in the United Kingdom.
`geoip.region`	The region name associated with the IP address. See MaxMind's FAQ (https://dev.maxmind.com/faq/how-do-i-convert-region-codes-to-names/) about converting region codes to names. The returned string uses the character encoding ISO-8859-1 (latin1).
`geoip.region.latin1`	An alias of `geoip.region`. The returned string uses the character encoding ISO-8859-1 (latin1).
`geoip.region.utf8`	The string from `geoip.region` encoded using the UTF-8 character encoding.
`geoip.region.ascii`	The string from `geoip.region` encoded using the ASCII character encoding (an ASCII approximation of the original string with diacritics removed).
`geoip.area_code`	The telephone area code associated with the IP address. These are only available for IP addresses in the US.
`geoip.metro_code`	The metro code associated with the IP address. These are only available for IP addresses in the US. MaxMind returns the same metro codes as the Google AdWords API (https://developers.google.com/adwords/api/docs/appendix/cities-DMAregions).

NOTE: GeoIP information, including data streamed by our log streaming service (/guides/streaming-logs/), is intended to be used only in connection with your use of Fastly services. Use of GeoIP data for other purposes may require permission of a GeoIP vendor, such as MaxMind (https://www.maxmind.com/en/license_agreement).

Fastly also exposes codes that describe the location of the data center the request came through as follows:

Variable	Description
`server.region`	A code representing the general region of the world in which the POP location resides. One of: APAC, Asia, EU-Central, EU-East, EU-West, Middle-East, North-America, SA-East, South-America, US-Central, US-East, US-West.
`server.datacenter`	A code representing one of Fastly's POP locations (/guides/about-fastly-services/fastly-pop-locations).

NOTE: All strings are returned using ISO-8859-1 character encoding (https://en.wikipedia.org/wiki/ISO/IEC_8859-1) (also known as latin1).

§ Guide to VCL (/guides/vcl/guide-to-vcl)

About Varnish and why Fastly uses it

Varnish is the open source software (https://www.fastly.com/open-source) Fastly commercialized with performance and capacity (among other) enhancements. Fastly's Varnish is based on Varnish 2.1 and our Varnish syntax is specifically compatible with Varnish 2.1.5. The principal configuration mechanism of Varnish software is the Varnish Configuration Language (VCL), the scripting language used to configure and add logic to Varnish caches. Varnish allows Fastly to apply changes to the cache software as it is executing. Specifically, VCL is generated, compiled, transmitted to all Fastly caches, loaded into the operating software, and activated immediately, with no waiting for maintenance windows and no server downtime.

The Fastly application generates VCL automatically per your specifications via the web interface (/guides/about-fastly-services/). We allow you to create your own VCL files with specialized configurations. Your custom VCL files can be uploaded (/guides/vcl/uploading-custom-vcl) into Fastly caches and activated. You can also mix and match (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl) custom VCL and Fastly VCL, using them together at the same time. You will never lose the options on the Fastly user interface when you use custom VCL, but keep in mind that custom VCL always takes precedence over any VCL generated by the user interface. Be mindful of where your custom VCL sits in the default VCL.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Fastly's VCL Extensions

In addition, Fastly has included a number of extensions to VCL that won't be covered by any other documentation. Specifically:

Extension	Description

cryptographic and hashing functions (/guides/vcl/cryptographic-and-hashing-related-vcl-functions)	Supports Hash-based Message Authentication Code (HMAC), a message authentication code that uses a cryptographic key in conjunction with a hash function.
date- and time-related features (/guides/vcl/date-and-time-related-vcl-features)	Supports the default VCL "now" variable that provides the current time as an RFC 850 formatted date (e.g., Tuesday, 29-Apr-14 08:41:55), as well as several new functions that allow you to have more flexibility when dealing with dates and times.
GeoIP features (/guides/vcl/geoip-related-vcl-features)	Provides the ability to search a database from MaxMind.com for a given host or IP address, and return information about the country, city or Internet Service Provider (ISP) for that IP address.
randomness features (/guides/vcl/randomness-related-vcl-features)	Supports the insertion of random strings, content cookies, and decisions into requests.
size-related variables (/guides/vcl/size-related-vcl-variables)	Supports reporting variables that offer insight into what happened in a request.
TLS and HTTP/2 variables (/guides/vcl/tls-and-http2-vcl-variables)	Supports the use of variables related to TLS and HTTP/2.
miscellaneous features and variables (/guides/vcl/miscellaneous-VCL-extensions)	Provides miscellaneous VCL extensions not easily grouped into other categories.

Embedding inline C code in VCL

Currently, we don't provide embedded C access to our users. Fastly is a shared infrastructure. By allowing the use of inline C code, we could potentially give a single user the power to read, write to, or write from everything. As a result, our varnish process (i.e., files on disk, memory of the varnish user's processes) would become unprotected because inline C code opens the potential for users to do things like crash servers, steal data, or run a botnet.

We appreciate feedback from our customers. If you are interested in a feature that requires C code, please contact support@fastly.com (mailto:support@fastly.com). Our engineering team looks forward to these kinds of challenges.

Where to learn more about Varnish and VCL

The official Varnish documentation (https://www.varnish-cache.org/docs/2.1/tutorial/vcl.html) is a good place to start when looking for online information. In addition, Varnish Software, who provides commercial support for Varnish, has written a free online book (http://info.varnish-software.com/the-varnish-book).

Roberto Moutinho's book *Instant Varnish Cache (http://www.amazon.com/Instant-Varnish-Cache-Roberto-Moutinho/dp/178216040X)* also provides information.

§ Isolating header values without regular expressions (/guides/vcl/isolating-header-values-without-regular-expressions)

Fastly supports the ability to extract header subfield values without regular expressions in a human-readable way.

"Headers subfields" refer to headers with a body syntax style similar to `value1=123value123; testValue=asdf_true; loggedInTest=true;` or `max-age=0, surrogate-control=3600` These headers include Cookie, Set-Cookie, Cache-Control, or a custom header. Fastly allows you to isolate these key values with the following syntax:

```
req.http.Header-Name:key-name
```

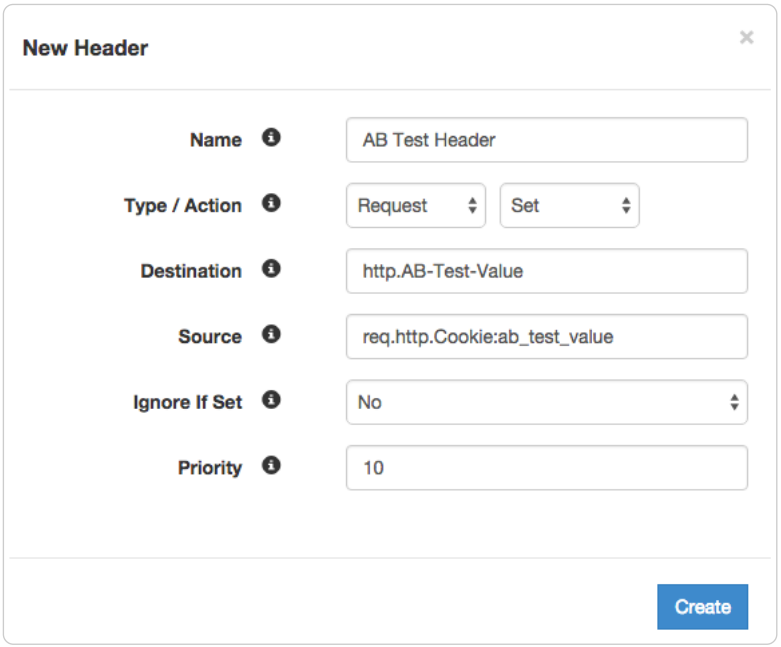
In cases where a `Set-Cookie` response from origin is `value1=123value123; testValue=asdf_true; loggedInTest=true;`, the code for isolating the `loggedInTest` value would be:

```
beresp.http.Set-Cookie:loggedInTest
```

This logic can be used in uploaded custom VCL, as well as throughout the UI. For example, using VCL this logic would execute based on the value of staff_user within req.http.Cookie.

```
# in vcl_recv
if (req.http.Cookie:staff_user ~ "true") {
  # some logic goes here
  return(pass);
}
```

For example, to isolate the value of `ab_test_value` from `Cookie` to the header `req.http.AB-Test-Value` in the header configuration UI (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses), set up a custom header with the following settings:



Fill out the fields with the following:

- In the **Name** field, type `AB Test Header`.
- From the **Type / Action** menu, select **Request** and **Set**.
- In the **Destination** field, type `http.AB-Test-Value`.
- In the **Source** field, type `req.http.Cookie:ab_test_value`.
- From the **Ignore If Set** menu, select **No**.
- In the **Priority** field, type `10`.

This will send the `AB-Test-Value` header in every inbound request.

§ Manipulating the cache key (/guides/vcl/manipulating-the-cache-key)

Redefining the cache key

WARNING:
By default, Fastly uses the URL and the host of a request (plus a special, internal Fastly variable for purging (/guides/purging/single-purges) purposes) to create unique HTTP objects. Although Fastly allows you to explicitly set the cache key to define this more precisely, changing the default behavior risks the following:

1. If you add too much information to the cache key, you can significantly reduce your hit ratio.
2. If you make a mistake when explicitly setting the cache key, you can cause all requests to get the same object.
3. If you add anything to the hash, you will need to send a purge for each combination of the URL and value you add in order to purge that specific information from the cache.

To avoid these dangers, consider using the Vary header (https://www.fastly.com/blog/best-practices-for-using-the-vary-header) instead of following the instructions below.

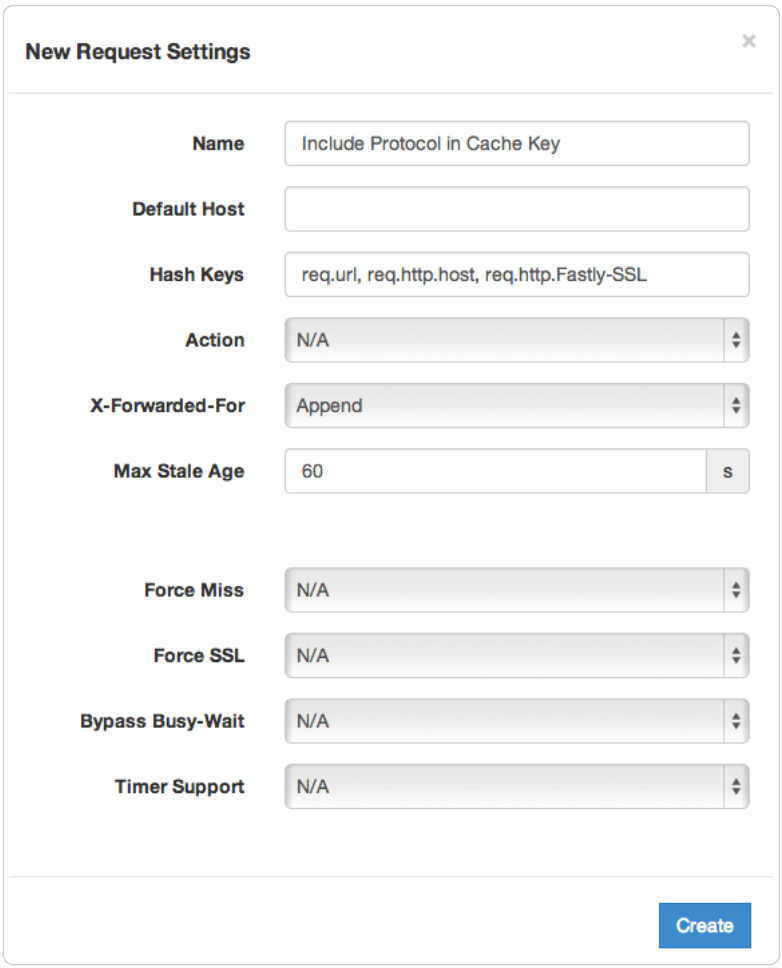
Explicitly setting the cache key

You can set the cache key explicitly (including attaching conditions) by adding a request setting via the Settings pane of your configuration control panel (/guides/about-fastly-services/about-the-web-interface-controls#about-the-configuration-control-panel) and including a comma-separated list of cache keys. The values of the cache keys listed are combined to make a single hash, and each unique hash is considered a unique object.

For example, if you don't want the query string to be part of the cache key, but you don't want to change `req.url` so that the query string still ends up in your logs, you could use the following text for the hash keys:

```
regsub(req.url, "\?.*$", ""), req.http.host
```

In the UI, the text would appear in the Hash Keys field of a new request setting:



As a general rule, you should always have `req.url` as one of your cache keys or as part of one. The example above includes `req.url` inside the `regsub()` therefore passing this requirement.

Purging adjustments when making additions to cache keys

Because purging works on individual hashes, additions to cache keys can complicate purging URLs. However, it can also be simplified.

For example, if you were to change your cache key to just `req.url` and not the default `req.url, req.http.host`, then purging `http://foo.example.com/file.html` would also purge `http://bar.example.com/file.html`. Keep in mind this is because they're actually the same object in the cache!

On the other hand, if you were to change your cache key `req.url, req.http.host, req.http.Fastly-SSL`, you would have to purge `http://example.com/` and `https://example.com/` individually.

In the latter case, if you were to use the Vary header instead of changing the cache key, you could still have different content on the two URLs, yet purge them with a single purge. In this case you would add a new Cache Header (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses), use `http.Vary` as the Destination, and use the following as the Source:

```
if(beresp.http.Vary, beresp.http.Vary ",", "") "Fastly-SSL"
```

Using a POST request body as a cache key

As long as the body of a POST request is less than 2K in size and the content type is `application/x-www-form-urlencoded`, then we allow you to use it as part of the cache key. Your VCL (/guides/vcl/) should look something like:

```
sub vcl_hash {
  set req.hash += req.url;
  set req.hash += req.http.host;
  if (req.request == "POST" && req.postbody) {
    set req.hash += req.postbody;
  }
  return(hash);
}
```

You'll also need to force a cache lookup, but only for requests that can be cached, by doing something like this in **vcl_recv**:

```
  if (req.request == "POST" && req.postbody ~ "(^|&)action=list(&|$)") {
    return(lookup);
  }
```

★ **TIP:** To refine this, you could add only the important parts of `req.postbody` to the hash using `regsub()`.

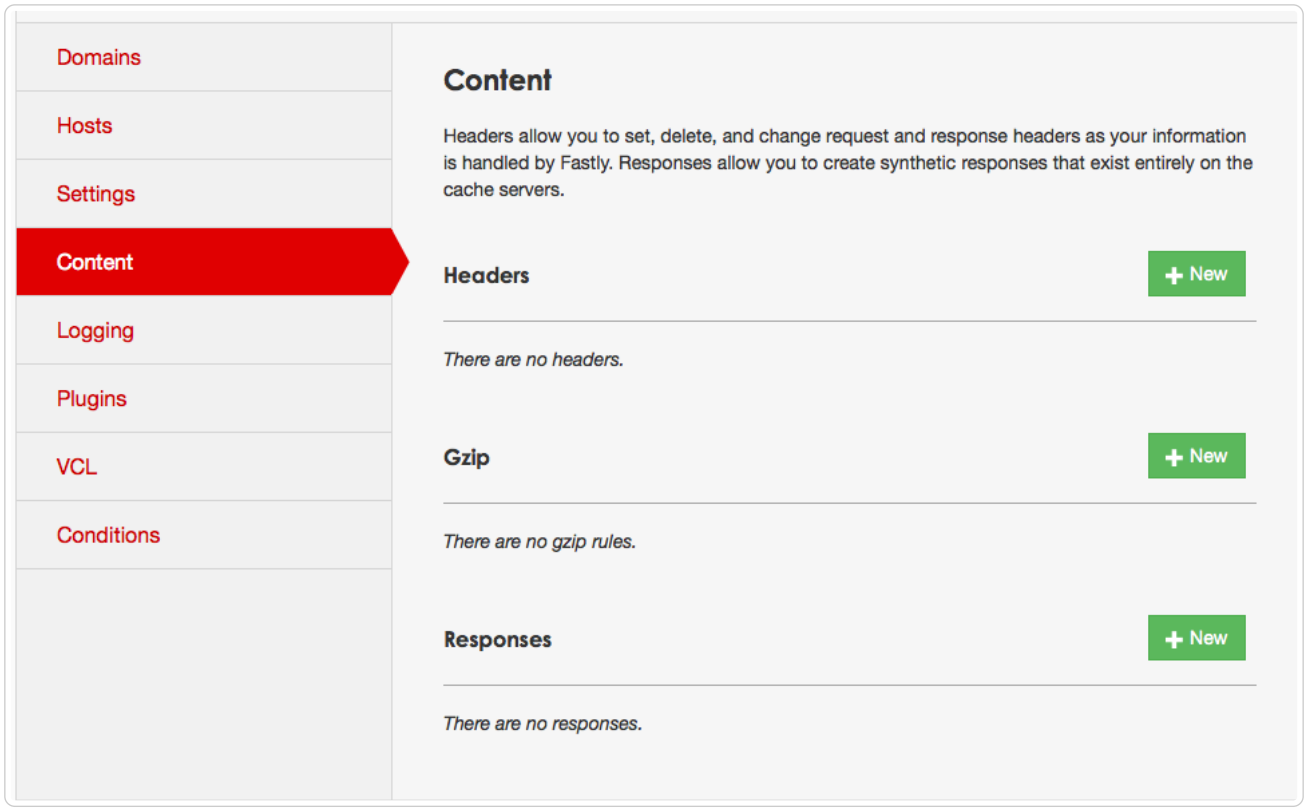
Using a cookie as a cache key

You can use a cookie as a cache key or just check for the presence of a cookie set to a specific value by controlling its request conditions (/guides/conditions/). Both methods are simple and shown in the steps below.

To use a cookie as a cache key

Using a cookie as a cache key looks complicated but it's actually quite simple. Let's say your cookie is called "MyCookie" and it looks like `mycookie=`

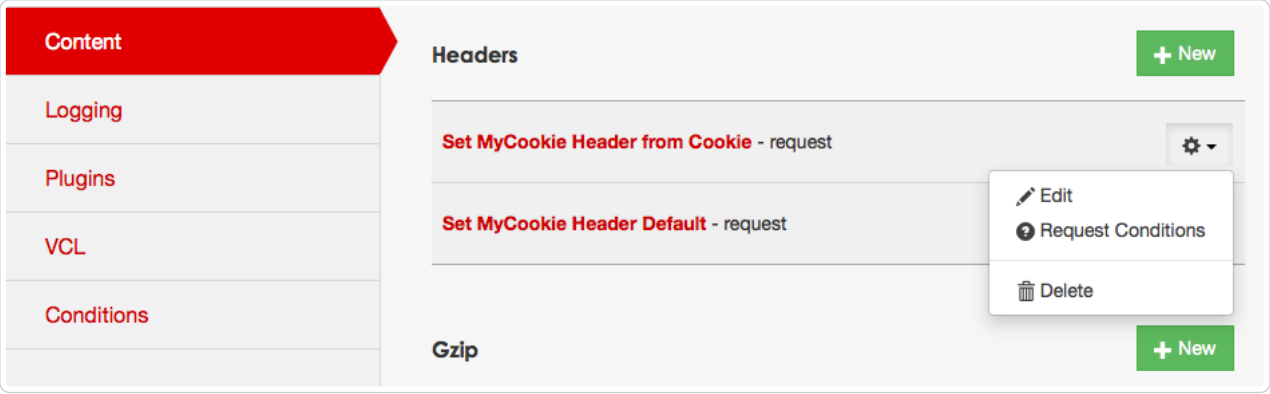
1. Go to the **configuration** tab, select your service, and select the **Content** pane.



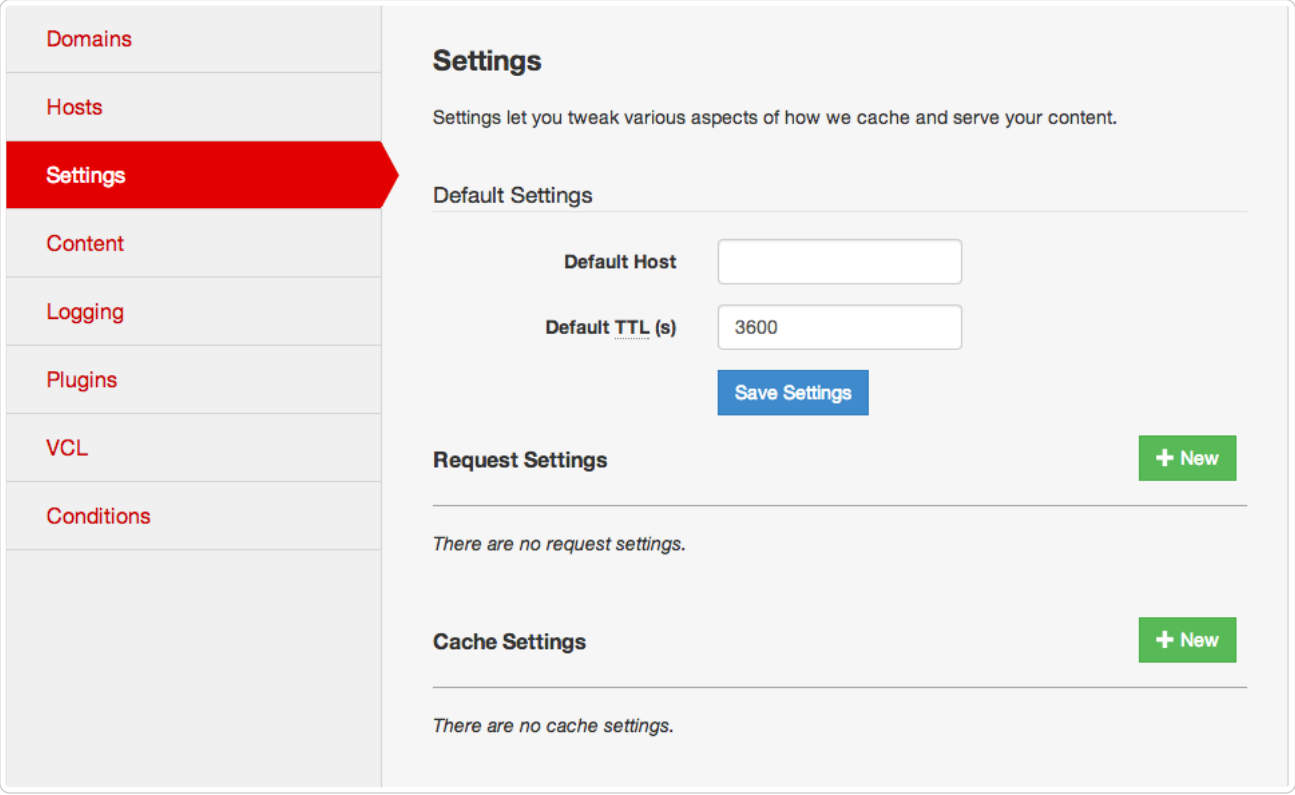
2. Click the **New** button next to the **Headers** section and create a new header called `Set MyCookie Header Default`.
   - Set the **Type** to `Request`
   - Set the **Action** to `Set`
   - Set the **Destination** to `http.X-MyCookie`
   - Set the **Source** to `"0"` (with quotes)
3. Create another new header called `Set MyCookie Header from Cookie`.

- Set the **Type** to `Request`
- Set the **Action** to `Set`
- Set the **Destination** to `http.X-MyCookie`
- Set the **Source** to `regsub(req.http.cookie, ".*mycookie =([^;]+);.*", "\1")`
- Set the **Priority** to be larger than the default header you just created

4. Click the gear icon to the right of the newly created `Set MyCookie Header from Cookie` header and select **Request Conditions**.



5. Choose the request conditions.
   - Set the **Name** to `Has MyCookie cookie`
   - Set the **Apply If** to `req.http.cookie ~ "mycookie="`
6. Now select the **Settings** pane.



7. Click the **New** button next to the **Request Settings** section and create a new request setting.
   - Set the **Name** to `Set Hash from Cookie`
   - Set the **Hash Keys** to `req.url, req.http.host, req.http.X-MyCookie`
8. Click the gear icon to the right of the newly created request setting and select **Conditions**.

9. On the **Choose Conditions** window, click **New** to add a new condition.

To check for the presence of a cookie set to a specific value

An alternative way if you're just checking for the presence of the cookie set to some specific value (e.g., 1):

1. Add a new Request Setting where the Hash Key is `req.url, req.http.host, "Has mycookie"`
2. Add a condition to that setting where the "Apply if ..." is `req.http.cookie ~ ".*mycookie =1;.*"`

---

§ Miscellaneous VCL features (/guides/vcl/miscellaneous-VCL-extensions)

Fastly has added several miscellaneous features (/guides/vcl/guide-to-vcl) to Varnish that don't easily fit into specific categories (/guides/vcl/).

Feature	Description
`bereq.url.basename`	Same as `req.url.basename`, except for use between Fastly and your origin servers.
`bereq.url.dirname`	Same as `req.url.dirname`, except for use between Fastly and your origin servers.
`beresp.backend.ip`	The IP of the backend this response was fetched from (backported from Varnish 3).
`beresp.backend.name`	The name of the backend this response was fetched from (backported from Varnish 3).
`beresp.backend.port`	The port of the backend this response was fetched from (backported from Varnish 3).
`beresp.grace`	Defines how long an object can remain overdue and still have Varnish consider it for grace mode. Fastly has implemented `stale-if-error` (/guides/performance-tuning/serving-stale-content#usage) as a parallel implementation of `beresp.grace`.
`beresp.pci`	Specifies that content be cached in a manner that satisfies PCI DSS requirements. See our PCI compliance description (/guides/compliance/pci-compliant-caching) for instructions on enabling this feature for your account.
`beresp.hipaa`	Specifies that content not be cached in non-volatile memory to help customers meet HIPAA security requirements. See our guide on HIPAA and caching PHI (/guides/compliance/hipaa-and-caching-phi) for instructions on enabling this feature for your account. An alias of `beresp.pci`.
`boltsort.sort`	Sorts URL parameters. For example, `boltsort.sort("/foo?b=1&a=2&c=3");` returns `"/foo?a=2&b=1&c=3"`.
`client.port`	Returns the remote client port. This could be useful as a seed that returns the same value both in an ESI and a top level request. For example, you could hash `client.ip` and `client.port` to get a value used both in ESI and the top level request.
`goto`	Performs a one-way transfer of control to another line of code. See the example for more information.
`http_status_matches`	Determines whether or not an HTTP status code matches a pattern. The arguments are an integer (usually `beresp.status` or `resp.status`) and a comma-separated list of status codes, optionally prefixed by a `!` to negate the match. It returns `TRUE` or `FALSE`. For example, `if (http_status_matches(beresp.status, "!200,404")) {`.
`if()`	Implements a ternary operator for strings; if the expression is true, it returns `TRUE`; if the expression is false, it returns `FALSE`. For example, you have an `if(x, true-expression, false-expression)`; if this argument is true, the `true-expression` is returned. Otherwise, the `false-expression` is returned. See the example for more information.
`req.grace`	Defines how long an object can remain overdue and still have Varnish consider it for grace mode.

`req.http.host`	The full host name, without the path or query parameters. For example, in the request `www.example.com/index.html?a=1&b=2`, `req.http.host` will return `www.example.com`.
`req.restarts`	Counts the number of times the VCL has been restarted.
`req.topurl`	In an ESI subrequest, returns the URL of the top-level request.
`req.url.basename`	The file name specified in a URL. For example, in the request `www.example.com/1/hello.gif?foo=bar`, `req.url.basename` will return `hello.gif`.
`req.url.dirname`	The directories specified in a URL. For example, in the request `www.example.com/1/hello.gif?foo=bar`, `req.url.dirname` will return `/1`. In the request `www.example.com/5/inner/hello.gif?foo=bar`, `req.url.dirname` will return `/5/inner`.
`req.url.ext`	The file extension specified in a URL. For example, in the request `www.example.com/1/hello.gif?foo=bar`, `req.url.ext` will return `gif`.
`req.url.path`	The full path, without any query parameters. For example, in the request `www.example.com/index.html?a=1&b=2`, `req.url.path` will return `/index.html`.
`req.url`	The full path, including query parameters. For example, in the request `www.example.com/index.html?a=1&b=2`, `req.url` will return `/index.html?a=1&b=2`.
`return`	Returns (with no return value) from a custom subroutine to exit early. See the example for more information.
`stale.exists`	Specifies if a given object has stale content (/guides/performance-tuning/serving-stale-content) in cache. Returns `TRUE` or `FALSE`.
`std.atoi`	Takes a string (which represents an integer) as an argument and returns its value.
`std.strstr`	Finds the first occurrence of a byte string and returns its value.
`std.tolower`	Changes the case of a string to lower case. For example, `std.tolower("HELLO");` will return `"hello"`.
`std.toupper`	Changes the case of a string to upper case. For example, `std.toupper("hello");` will return `"HELLO"`.
`urldecode`	Decodes a percent-encoded string. For example, `urldecode("hello%20world+!");` will return `"hello world !"`.
`urlencode`	Encodes a string for use in a URL. This is also known as percent-encoding (https://en.wikipedia.org/wiki/Percent-encoding). For example, `urlencode("hello world");` will return `"hello%20world"`.

Examples

Use the following examples to learn how to implement the features.

Goto

In the C programming language, `goto` is used to jump to a common error handling block before returning from a function. It can be used to do something similar in VCL, such as perform logic or set headers before returning lookup, error, or pass. It also works in custom subroutines. You can only use it to jump forward in the code, not backward.

```
sub vcl_recv {
  if (!req.http.Foo) {
    goto foo;
  }

foo:
  set req.http.Foo = "1";
}
```

Return

You can use `return` to exit early from a custom subroutine.

```
sub custom_subroutine {
  if (req.http.Cookie:user_id) {
    return;
  }

  # do a bunch of other stuff
}
```

If

You can use `if()` as a construct to make simple conditional expressions more concise.

```
set req.http.foo-status = if(req.http.foo, "present", "absent");
```

§ Mixing and matching Fastly VCL with custom VCL (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl)

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Fastly Varnish syntax is specifically compatible with Varnish 2.1.5 (https://www.varnish-cache.org/docs/2.1). We run a custom version with added functionality and our VCL parser has its own pre-processor. To mix and match Fastly VCL with your custom VCL successfully, remember the following:

- **You can only restart Varnish requests three times.** This limit exists to prevent infinite loops.
- **VCL doesn't take kindly to Windows newlines (line breaks).** It's best to avoid them entirely.
- **It's best to use `curl -X PURGE` to initiate purges via API (/api/purge).** To restrict access to purging, check for the `FASTLYPURGE` method not the `PURGE` method. When you send a request to Varnish to initiate a purge, the HTTP method that you use is "PURGE", but it has already been changed to "FASTLYPURGE" by the time your VCL runs that request.
- **If you override TTLs with custom VCL, your default TTL set in the configuration (/guides/performance-tuning/serving-stale-content) will not be honored** and the expected behavior may change.

Inserting custom VCL in Fastly's VCL boilerplate

DANGER: Include all of the Fastly VCL boilerplate as a template in your custom VCL file, especially the VCL macro lines (they start with `#FASTLY`). VCL macros expand the code into generated VCL. Add your custom code *in between* the different sections as shown in the example unless you specifically intend to override the VCL at that point.

Custom VCL placement example

```
sub vcl_miss {
  # my custom code
  if (req.http.User-Agent ~ "Googlebot") {
    set req.backend = F_special_google_backend;
  }
#FASTLY miss
}
```

Fastly's VCL boilerplate

```
sub vcl_recv {
#FASTLY recv

  if (req.request != "HEAD" && req.request != "GET" && req.request != "FASTLYPURGE") {
    return(pass);
  }

  return(lookup);
}

sub vcl_fetch {
#FASTLY fetch

  if ((beresp.status == 500 || beresp.status == 503) && req.restarts < 1 && (req.request == "GET" || req.request == "HEAD")) {
    restart;
  }

  if (req.restarts > 0) {
    set beresp.http.Fastly-Restarts = req.restarts;
  }

  if (beresp.http.Set-Cookie) {
    set req.http.Fastly-Cachetype = "SETCOOKIE";
    return(pass);
  }

  if (beresp.http.Cache-Control ~ "private") {
    set req.http.Fastly-Cachetype = "PRIVATE";
    return(pass);
  }

  if (beresp.status == 500 || beresp.status == 503) {
    set req.http.Fastly-Cachetype = "ERROR";
    set beresp.ttl = 1s;
    set beresp.grace = 5s;
    return(deliver);
  }

  if (beresp.http.Expires || beresp.http.Surrogate-Control ~ "max-age" || beresp.http.Cache-Control ~ "(s-maxage|max-age)") {
    # keep the ttl here
  } else {
    # apply the default ttl
    set beresp.ttl = 3600s;
  }

  return(deliver);
}

sub vcl_hit {
#FASTLY hit

  if (!obj.cacheable) {
    return(pass);
  }
  return(deliver);
}

sub vcl_miss {
#FASTLY miss
  return(fetch);
}

sub vcl_deliver {
#FASTLY deliver
  return(deliver);
}

sub vcl_error {
#FASTLY error
}

sub vcl_pass {
#FASTLY pass
}
```

§ Overriding which IP address the GeoIP features use (/guides/vcl/overriding-which-ip-address-the-geoip-features-use)

By default the GeoIP lookup is based on the IP address of the user. In some cases, such as with traffic through proxies, this type of lookup doesn't work properly. In particular, users of Opera Mini always browse through a proxy and the true IP address appears in the X-Forwarded-For header (https://dev.opera.com/articles/opera-mini-request-headers/#x-forwarded-for). Similarly, the Amazon Silk browser (https://developer.amazon.com/appsandservices/community/post/TxOMW3RNF3FYRK/Amazon-Silk-Tips-for-Site-Owners) can optionally come through a proxy, indicated via the User Agent string.

To work around this, you can set this variable to account for both the Opera Mini and Amazon Silk browsers:

`set geoip.use_x_forwarded_for = true;`

To do so, you would use code like this in `vcl_recv`:

```
if (req.http.X-OperaMini-Features || req.http.User-Agent ~ " Silk-Accelerated=true$") {
  set geoip.use_x_forwarded_for = true;
}
```

which tells Fastly to use the X-Forwarded-For header for the IP address. If it is not available, then the code will fall back to using the IP address of the client.

Finally, just in case there's some scenario or browser we haven't anticipated, you can also override based on an arbitrary header:

```
set geoip.ip_override = req.http.Custom-IP-Override;
```

Setting either of these variables:

```
set geoip.use_x_forwarded_for = true;
```

```
set geoip.ip_override = req.http.Custom-IP-Override;
```

will force the GeoIP information to be reloaded.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

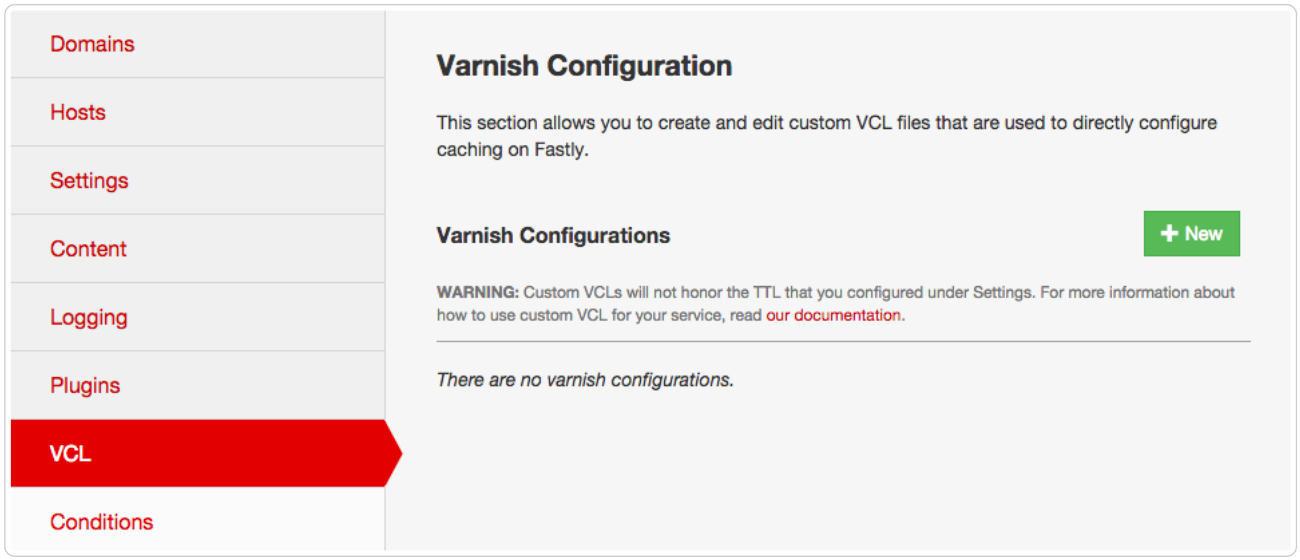
§ Previewing VCL before activating it (/guides/vcl/previewing-and-testing-vcl-before-activating-it)

Previewing VCL before activation

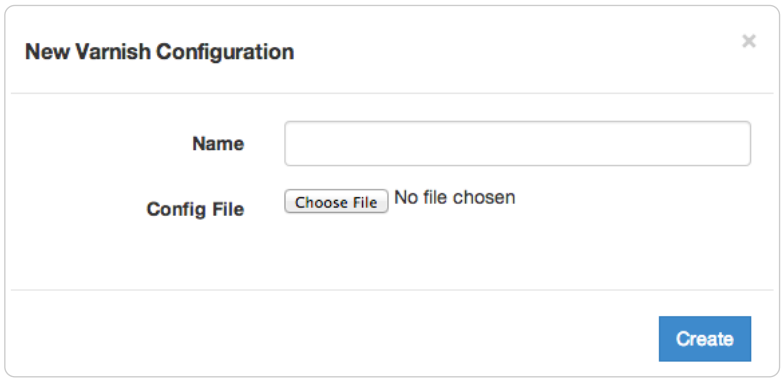
You can preview VCL prior to activating a service version.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

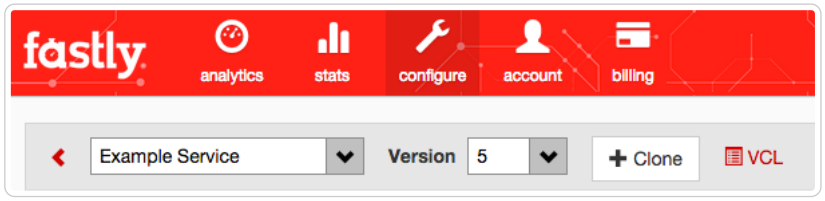
3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **VCL** section to view the Varnish Configuration information, and then click the **New** button to upload your configuration file.



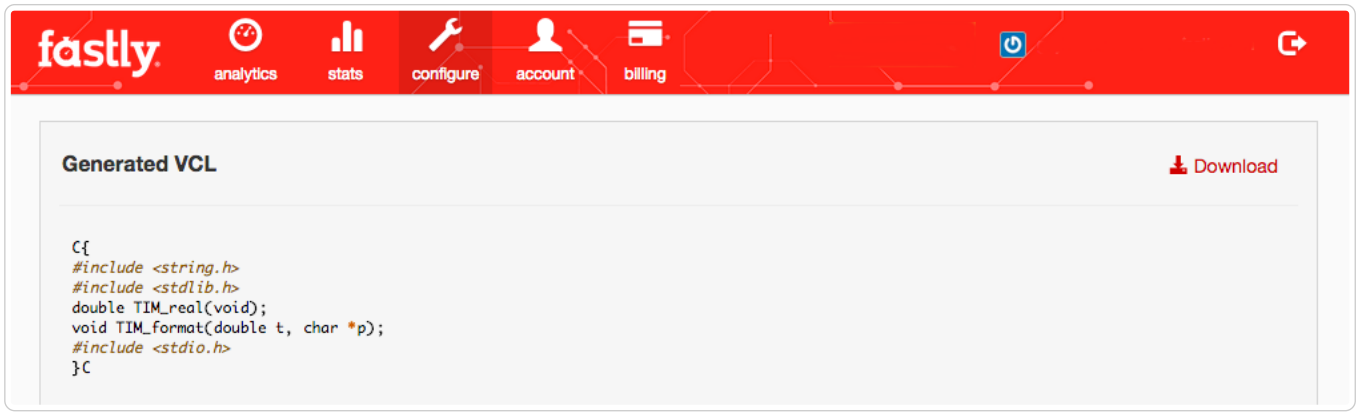
The New Varnish Configuration window appears:



6. In the **Name** field, type the name of the varnish configuration information you're uploading.
7. Click the **Choose File** button and navigate to the text file you plan to upload.
8. Click the **Create** button to create the new varnish configuration.
9. At the top of the application window, click the **VCL** button to the right of the service name and version fields.



The Generated VCL preview appears.



Testing VCL configurations

You don't need a second account to test your VCL configurations with the Fastly application. We recommend adding a new service within your existing account that's specifically designed for testing. A name like "QA" or "testing" or "staging" makes distinguishing between services much easier.

Once created, simply point your testing service to your testing or QA environment. Edit your Fastly configurations for the testing service as if you were creating them for production. Preview your VCL, test things out and tweak them to get them perfect.

When your testing is complete, make the same changes in your production service that you made to your testing service. If you are using custom VCL, upload the VCL file (/guides/vcl/uploading-custom-vcl) to the production service you'll be using.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

§ Query string manipulation VCL features (/guides/vcl/query-string-manipulation-vcl-features)

Fastly provides several functions in VCL for query-string manipulation. It is based on Dridi Boukelmoune's vmod-querystring (https://github.com/Dridi/libvmod-querystring) for Varnish — you can also refer to the documentation (https://github.com/Dridi/libvmod-querystring) for more information.

Functions

Function	Description	Example
`querystring.clean(<string>)`	Returns the given URL without empty parameters. The query-string is removed if empty (either before or after the removal of empty parameters). Note that a parameter with an empty value does not constitute an empty parameter, so a query string "?something" would not be cleaned.	`set req.url = querystring.clean(req.url);`
`querystring.filter(<string>, <string_list>)`	Returns the given URL without the listed parameters.	`set req.url = querystring.filter(req.url, "utm_source" + querystring.filtersep() + "utm_medium" + querystring.filtersep() + "utm_campaign");`
`querystring.filter_except(<string>, <string_list>)`	Returns the given URL but only keeps the listed parameters.	`set req.url = querystring.filter_except(req.url, "q" + querystring.filtersep() + "p");`
`querystring.filtersep()`	Returns the separator needed by the `filter` and `filter_except` functions.	
`querystring.globfilter(<string>, <string>)`	Returns the given URL without the parameters matching a glob.	`set req.url = querystring.globfilter(req.url, "utm_*");`
`querystring.globfilter_except(<string>, <string>)`	Returns the given URL but only keeps the parameters matching a glob.	`set req.url = querystring.globfilter_except(req.url, "sess*");`
`querystring.regfilter(<string>, <string>)`	Returns the given URL without the parameters matching a regular expression.	`set req.url = querystring.regfilter(req.url, "utm_.*");`
`querystring.regfilter_except(<string>, <string>)`	Returns the given URL but only keeps the parameters matching a regular expression.	`set req.url = querystring.regfilter_except(req.url, "^(q|p)$");`
`querystring.remove(<string>)`	Returns the given URL with its query-string removed.	`set req.url = querystring.remove(req.url);`

`querystring.sort(<string>)`	Returns the given URL with its query-string sorted.	`set req.url = querystring.sort(req.url);`

Examples

In your VCL, you could use `querystring.sort` as follows:

```
import querystring;

sub vcl_hash {
  # sort the URL before the request hashing
  set req.url = querystring.sort(req.url);
}
```

You can use `querystring.regfilter` to specify a list of arguments that must not be removed (everything else will be) with a negative look-ahead expression:

```
set req.url = querystring.regfilter(req.url, "^(?!param1|param2)");
```

§ Randomness-related VCL features (/guides/vcl/randomness-related-vcl-features)

Fastly provides several functions in VCL (/guides/vcl/) that control randomness-related activities.

⚠ **WARNING:** We use BSD random number functions from the GNU C Library (http://www.gnu.org/software/libc/manual/html_node/BSD-Random.html), not true randomizing sources. These VCL functions should not be used for cryptographic (/guides/vcl/cryptographic-and-hashing-related-vcl-functions) or security purposes.

Random strings

Use the function `randomstr(length [, characters])`. When characters aren't provided, the default will be the 64 characters of `A-Za-z0-9_-`.

```
sub vcl_deliver {
  set resp.http.Foo = "randomstuff=" randomstr(10);
  set resp.http.Bar = "morsecode=" randomstr(50, ".-"); # 50 dots and dashes
}
```

Random content cookies in pure VCL

```
sub vcl_deliver {
  add resp.http.Set-Cookie = "somerandomstuff=" randomstr(10) "; expires=" now + 180d "; path=/;";
}
```

This adds a cookie named "somerandomstuff" with 10 random characters as value, expiring 180 days from now.

Random decisions

Use the function `randombool( _numerator_, _denominator_ )`, which has a numerator/denominator chance of returning true.

```
sub vcl_recv {
  if (randombool(1, 4)) {
    set req.http.X-AB = "A";
  } else {
    set req.http.X-AB = "B";
  }
}
```

This will add a X-AB header to the request, with a 25% (1 out of 4) chance of having the value "A", and 75% chance of having the value "B".

The `randombool()` function accepts INT function return values, so you could do something this:

```
if (randombool(std.atoi(req.http.Some-Header), 100))  {
  # do something
}
```

Another function, `randombool_seeded()`, takes an additional seed argument. Results for a given seed will always be the same. For instance, in this example the value of the response header will always be `no`:

```
if (randombool_seeded(50, 100, 12345)) {
  set resp.http.Seeded-Value = "yes";
} else {
  set resp.http.Seeded-Value = "no";
}
```

This could be useful for stickiness. For example, if you based the seed off of something that identified a user, you could perform A/B testing without setting a special cookie.

⚠ **WARNING:** The `randombool` and `randombool_seeded` functions do not use secure random numbers and should not be used for cryptographic purposes.

§ Response Cookie handling (/guides/vcl/response-cookie-handling)

The traditional way to read response cookies in VCL is to inspect either the `beresp.http.Set-Cookie` or the `resp.http.Set-Cookie` variables and then extract values using regular expressions. However this is not ideal since attempting to parse potentially complicated or quoted strings with regular expressions is brittle and prone to being tripped up by edge cases. It also doesn't allow for reading multiple headers with the same name such as when an origin sends multiple `Set-Cookie` headers. Because of these two reasons Fastly supports a method for extracting a named value out of `Set-Cookie` headers no matter how many there are.

To access a named value simply use the function with either `beresp` or `resp` depending on what part of the request you're in - so either

```
setcookie.get_value_by_name(beresp, "name")
```

or

```
setcookie.get_value_by_name(resp, "name")
```

as appropriate, replacing `"name"` with whatever the name of the value is. So for example, given this HTTP response from an origin

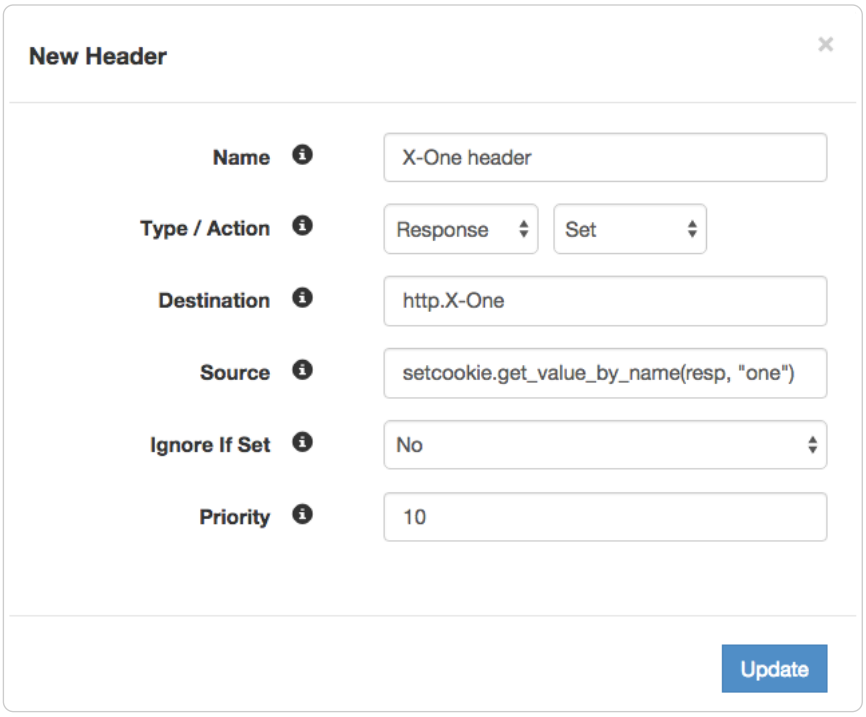
```
HTTP/1.1 200 OK
Cache-Control: max-age=60
Content-Type: text/html; charset=utf-8
Content-Length: 80806
Accept-Ranges: bytes
Date: Tue, 11 Aug 2015 19:00:04 GMT
Age: 123
Connection: keep-alive
Set-Cookie: one=a; httponly; secure
Set-Cookie: two=b or not to b; httponly
```

then using the function like this

```
set resp.http.X-One = setcookie.get_value_by_name(resp, "one");
set resp.http.X-Two = setcookie.get_value_by_name(resp, "two");
```

will set `resp.http.X-One` to be "a" and `resp.http.X-Two` to "b or not to b".

This logic can be used in uploaded custom VCL (/guides/vcl/uploading-custom-vcl), as well as throughout the UI. For example:



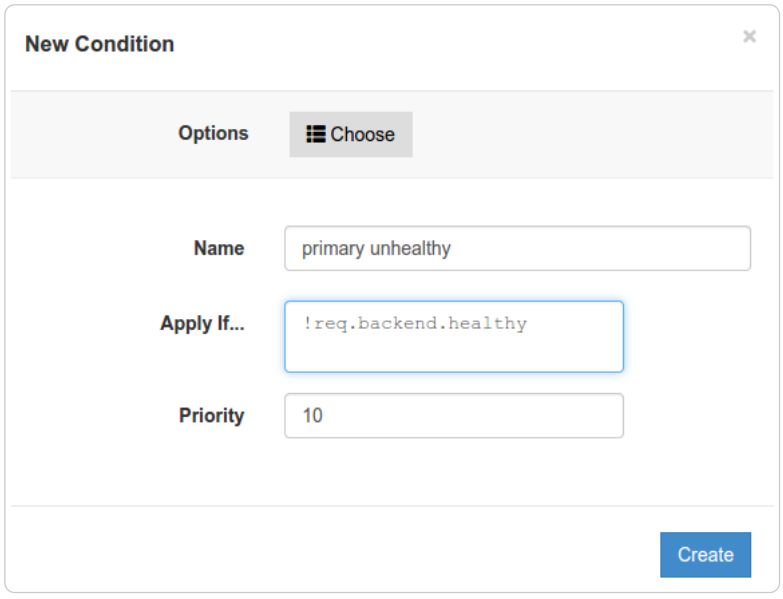
§ Setting up redundant origin servers (/guides/vcl/setting-up-redundant-origin-servers)

Sometimes you want to set up two different origin servers, one as a primary and one as a backup in case the primary becomes unavailable. You can do this via the user interface or using custom VCL.

Using the user interface

Set up redundant origins via the UI using these steps.

1. In the Fastly application, define a health check (/guides/basic-configuration/health-checks-tutorial) and assign it to the primary origin server.
2. Add a condition to the secondary origin server, with the following settings:



3. Click the VCL button at the top to view the generated VCL, and confirm the following snippets appear in `vcl_recv`:

```
# default conditions
set req.backend = F_primary;
```

```
# Request Condition: primary unhealthy Prio: 10
if (!req.backend.healthy) {
  set req.backend = F_secondary;
}
#end condition
```

Using custom VCL

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Set up redundant origins with custom VCL using these steps.

1. In the Fastly application, define a health check (/guides/basic-configuration/health-checks-tutorial) and assign it to the primary origin server.
2. Copy the boilerplate VCL from our guide on mixing Fastly VCL with custom VCL (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl), and paste it into a new file.
3. Replace the `vcl_recv` sub with:

   ```
   sub vcl_recv {
   #FASTLY recv
     set req.backend = F_<primary_origin>;
     if (!req.backend.healthy) {
       set req.backend = F_<secondary_origin>;
     }
     if (req.request != "HEAD" && req.request != "GET" && req.request != "FASTLYPURGE") {
       return(pass);
     }
     return(lookup);
   }
   ```

   To find the exact backend names, view the generated VCL (/guides/vcl/previewing-and-testing-vcl-before-activating-it) using the VCL button at the top of the configuration control panel.
4. Upload (/guides/vcl/uploading-custom-vcl) your VCL file.

§ Size-related VCL variables (/guides/vcl/size-related-vcl-variables)

To allow better reporting, Fastly has added several variables (/guides/vcl/guide-to-vcl#fastly's-vcl-extensions) to VCL to give more insight into what happened in a request.

Name	Where	Description
`req.bytes_read`	deliver log	How big a request was in total bytes.
`req.header_bytes_read`	all	How big the header of a request was in total bytes.
`req.body_bytes_read`	deliver log	How big the body of a request was in total bytes.
`resp.bytes_written`	log	How many bytes in total were sent as a response.
`resp.header_bytes_written`	log	How many bytes were written for the header of a response.
`resp.body_bytes_written`	log	How many bytes were written for body of a response.
`resp.completed`	log	Whether the response completed successfully or not.

§ Support for the Edge-Control header (/guides/vcl/support-for-the-edge-control-header)

VCL provides the building blocks to access information inside the Edge-Control response header field from the origin. We support this by honoring `cache-maxage` from Edge-Control as the time to live (TTL) of the object on the Fastly edge, and honoring `downstream-ttl` from Edge-Control as the TTL to be sent down from the Fastly edge to the end user's browser.

In order to incorporate this Edge-Control header support, include the following snippet in your `vcl_fetch` function via custom VCL (/guides/vcl/uploading-custom-vcl):

```
if (parse_time_delta(subfield(beresp.http.Edge-Control, "downstream-ttl")) >= 0) {
  set beresp.http.Cache-Control = "max-age=" parse_time_delta(subfield(beresp.http.Edge-Control, "downstream-ttl"));
}

if (parse_time_delta(subfield(beresp.http.Edge-Control, "cache-maxage")) >= 0) {
  set beresp.ttl = parse_time_delta(subfield(beresp.http.Edge-Control, "cache-maxage"));
}
```

The subfield function parses the Edge-Control field for subfields, and the `parse_time_delta` function converts time values like "7m" into a number of seconds. You can then use that number of seconds to populate `beresp.ttl` (the TTL of the object on the Fastly edge) or you can use it to construct a Cache-Control header field for downstream. The `parse_time_delta` function will return -1 if the subfield is not well-formed as a time value, or if it is entirely absent. The above snippet honors `cache-maxage` and `downstream-ttl` from Edge-Control if present and usable.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

§ TLS and HTTP/2 VCL variables (/guides/vcl/tls-and-http2-vcl-variables)

Fastly has added several variables (/guides/vcl/guide-to-vcl#fastly-39-s-vcl-extensions) that expose information about the TLS and HTTP/2 attributes of a request.

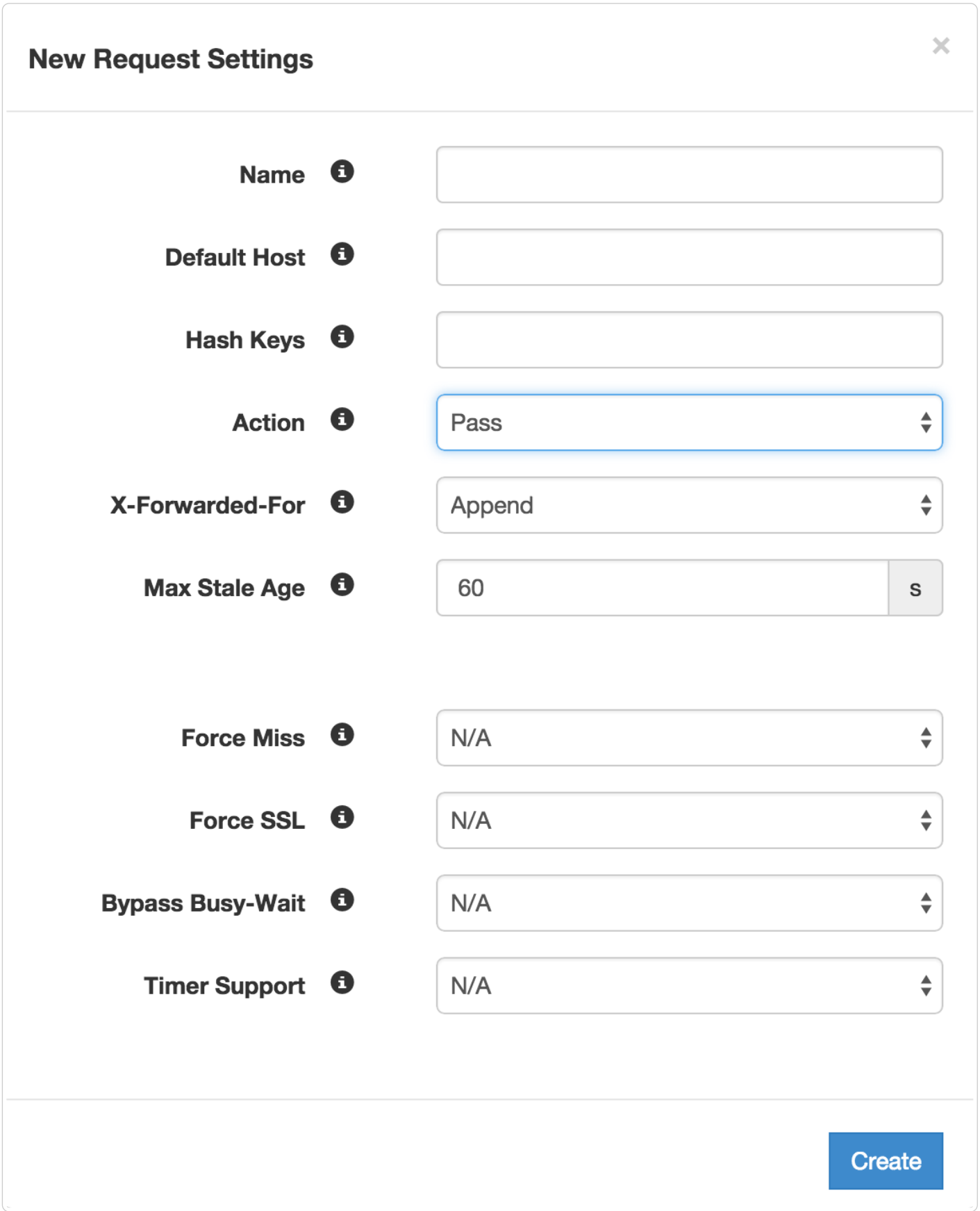
IMPORTANT: These VCL variables are currently only allowed to appear within the VCL hooks `vcl_recv`, `vcl_hash`, `vcl_deliver` and `vcl_log`.

Feature	Description
`fastly_info.is_h2`	Whether or not the request was made using HTTP/2. Returns `TRUE` or `FALSE`.
`fastly_info.h2.is_push`	If the request was made over HTTP/2, whether or not the request is part of a push request. Returns `TRUE` or `FALSE`.
`fastly_info.h2.stream_id`	If the request was made over HTTP/2, the underlying HTTP/2 stream id.
`tls.client.protocol`	The TLS protocol version that this connection is speaking over. Example: `"TLSv1.2"`
`tls.client.servername`	The Server Name Indication (SNI) that the client sent in the `ClientHello` TLS record. Returns `""` if the client did not send SNI. Returns `NULL` (the undefined string) if the request is not a TLS request.
`tls.client.cipher`	The cipher suite used to secure the client TLS connection. Example: `"ECDHE-RSA-AES128-GCM-SHA256"`
`tls.client.ciphers_sha`	A SHA1 of the two-byte cipher suite identifiers sent from the client as part of the TLS handshake, represented in base64.
`tls.client.tlsexts_sha`	A SHA1 of the TLS extension identifiers sent from the client as part of the TLS handshake, represented in base64.

§ Understanding the different PASS action behaviors (/guides/vcl/understanding-the-different-pass-action-behaviors)

Passing with a request setting and with a cache setting triggers very different behavior in Varnish (/guides/vcl/guide-to-vcl). Within VCL, passing with a request setting is the same as `return(pass)` in `vcl_recv`. Passing with a cache setting is the same as `return(pass)` in `vcl_fetch`. If you are familiar with Varnish 3+, passing with a cache setting is equivalent to `return(hit_for_pass)`.

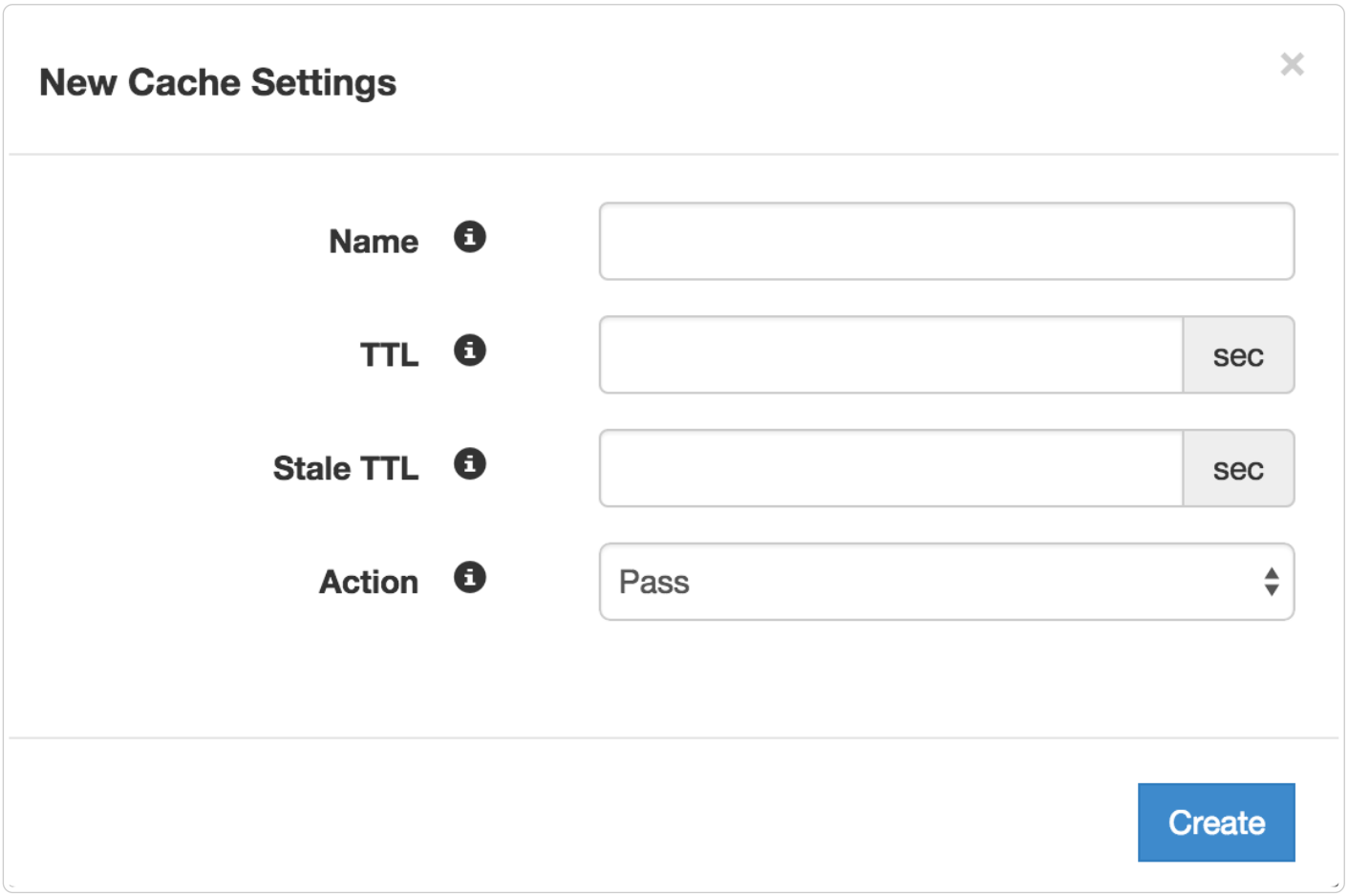
Using a request setting



Passing with a request setting translates within your generated VCL to `return(pass)` in `vcl_recv`. Varnish will not perform a lookup to see if an object is in cache and the response from the origin will not be cached.

Passing in this manner disables request collapsing. Normally simultaneous requests for the same object that result in cache misses will be collapsed down to a single request to the origin. While the first request is sent to the origin, the other requests for that object are queued until a response is received. When requests are passed in `vcl_recv`, they will all go to the origin separately without being collapsed.

Using a cache setting



Passing with a cache setting translates within your generated VCL to `return(pass)` in `vcl_fetch`. At this point in the flow of a request, Varnish has performed a lookup and determined that the object is not in cache. A request to the origin has been made; however, in vcl_fetch we have determined that the response is not cacheable. In Fastly's default VCL, this can happen based on the presence of a `Set-Cookie` response header from the origin.

Passing in `vcl_fetch` is often not desirable because request collapsing is *not* disabled. This makes sense since Varnish is not aware in `vcl_recv` that the object is uncacheable. On the first request for an object that will be later passed in `vcl_fetch`, all other simultaneous cache misses will be queued. Once the response from the origin is received and Varnish has realized that the request should be passed, the queued requests are sent to the origin.

This creates a scenario where two users request an object at the same time, and one user must wait for the other before being served. If these requests were passed in `vcl_recv`, neither user would need to wait.

To get around this disadvantage, when a request is passed in `vcl_fetch`, Varnish creates what is called a hit-for-pass object. These objects have their own TTLs and while they exist, Varnish will pass any requests for them as if the pass had been triggered in `vcl_recv`. For this reason, it is important to set a TTL that makes sense for your case when you pass in `vcl_fetch`. All future requests for the object will be passed until the hit-for-pass object expires. Hit-for-pass objects can also be purged like any other object.

Even with this feature, there will be cases where simultaneous requests will be queued and users will wait. Whenever there is not a hit-for-pass object in cache, these requests will be treated as if they are normal cache misses and request collapsing will be enabled. Whenever possible it is best avoid relying on passing in `vcl_fetch`.

Using req.hash_always_miss and req.hash_ignore_busy

Setting `req.hash_always_miss` forces a request to miss whether it is in cache or not. It is different when passing in `vcl_recv` in that the response will be cached and request collapsing will not be disabled. Later on the request can still be passed in `vcl_fetch` if desired.

A second relevant variable is `req.hash_ignore_busy`. Setting this to true disables request collapsing so that each request is sent separately to origin. When `req.hash_ignore_busy` is enabled all responses will be cached and each response received from the origin will overwrite the last. Future requests for the object that are served from cache will receive the copy of the object from the last cache miss to complete. `req.hash_ignore_busy` is used mostly for avoiding deadlocks in complex multi-Varnish setups.

Setting both these variables can be useful to force requests to be sent separately to the origin while still caching the responses.

§ Uploading custom VCL (/guides/vcl/uploading-custom-vcl)

Fastly allows you create your own Varnish Configuration Language (VCL) files with specialized configurations. By uploading custom VCL files to the Fastly application, you can use custom VCL and Fastly VCL together at the same time (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl). Keep in mind that custom VCL always takes precedence over VCL generated by the Fastly application.

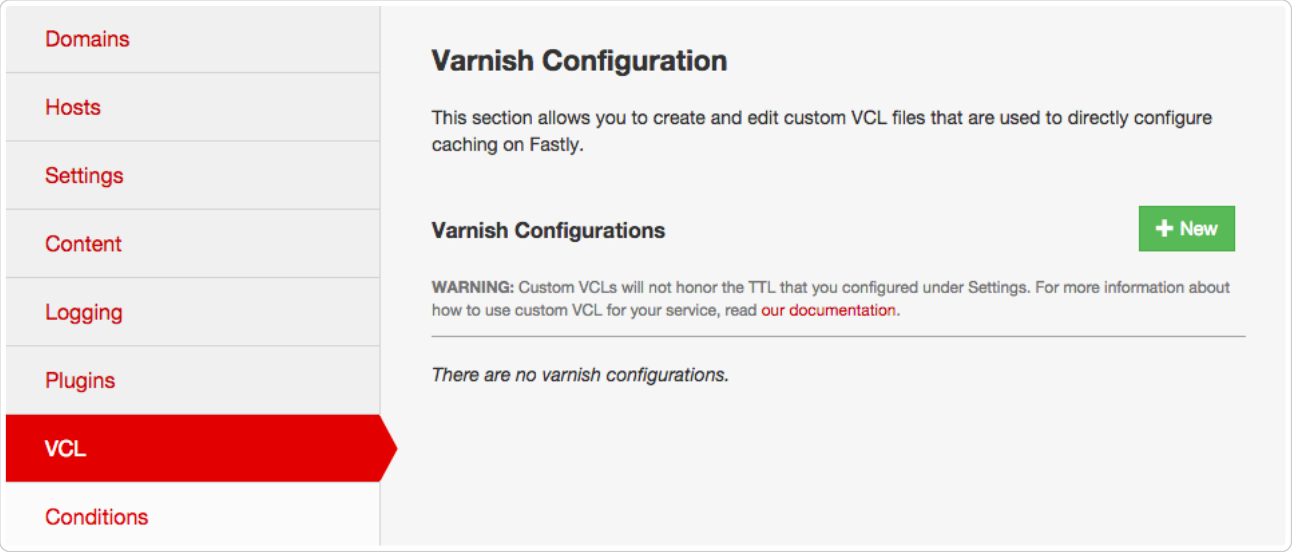
IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Accessing the VCL interface

After Fastly support has enabled the ability to upload custom VCL to your account, you can access the VCL interface in the Fastly application. Follow these instructions to access the VCL interface:

1. Log in to the Fastly application.
2. Click the **configure** tab (wrench icon).

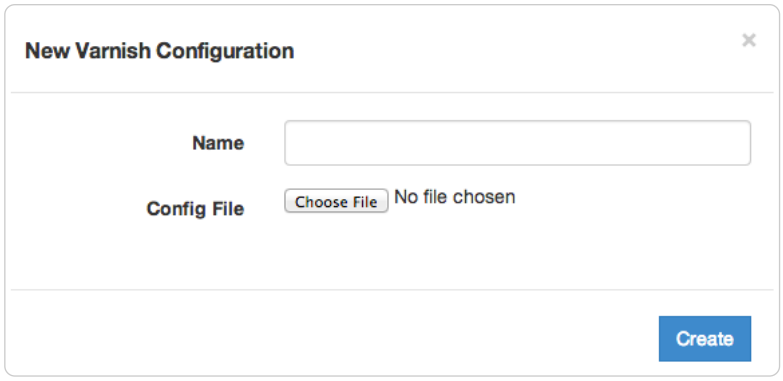
3. From the **Service** menu, select a service and then click the blue **Configure** button. The main controls for your selected service appear.
4. Click **VCL** from the section list on the left. The Varnish Configuration page appears.



Uploading a VCL file

Follow these instructions to upload a custom VCL file:

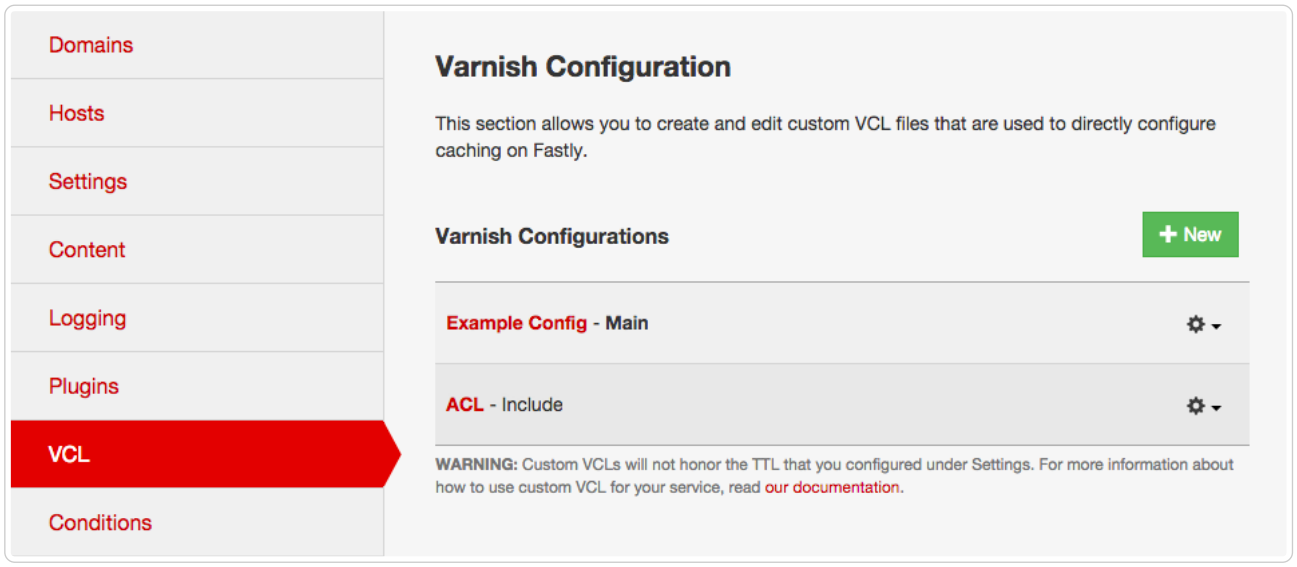
1. In the **Varnish Configuration** area, click the **New** button. The New Varnish Configuration window appears.



2. In the **Name** field, enter the name of the VCL file. For included files, this name must match the include statement in the main VCL file. See things to know when uploading multiple files for more information.
3. Click **Choose File** and select a file to upload.

> **IMPORTANT:** Don't upload generated VCL that you've downloaded from the Fastly application. Instead, edit and then upload a copy of Fastly's VCL boilerplate (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl#fastly-39-s-vcl-boilerplate) to avoid errors.

4. Click **Create**. The VCL file appears in the Varnish Configurations area.



Things to know when uploading multiple files

The first VCL file you upload will be marked as **Main** to the right of the file name in the Varnish configurations area. Any other VCL files will be marked as **Include** files. They must be included in the main VCL using the syntax `include "VCL Name"` where `VCL Name` is the name of an included VCL object you've created.

For example, if you've created an included VCL object called "ACL" (to use an access control list (/guides/vcl/using-access-control-lists) for code manageability) and the file is named `acl.vcl`, your main VCL configuration file would need to contain this line:

```
include "ACL"
```

§ Using access control lists (/guides/vcl/using-access-control-lists)

Varnish (/guides/vcl/guide-to-vcl) allows you to use Access Control Lists (ACLs), a feature that enables fast matching of a client's IP address against a list of defined IP addresses. An ACL looks like this:

```
# Who is allowed access ...
acl local {
  "localhost";
  "192.168.1.0"/24; /* and everyone on the local network */
  ! "192.168.1.23"; /* except for the dial-in router */
}
```

Defining an ACL

Using ACLs requires you to create and add custom VCL to Fastly's boilerplate VCL. To define an ACL in your Fastly configuration:

1. Ask support (mailto:support@fastly.com) to enable custom VCL uploading (/guides/vcl/uploading-custom-vcl) for your account.
2. Read about how to mix and match custom VCL (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl) with Fastly's VCL.
3. Create a custom VCL file with your ACL definitions included in the appropriate location. Use the example shown below as a guide. You can reference the ACL in your configuration (`vcl_recv`) using a match operation which can be located above or below `#FASTLY recv` - the placement only matters for the order of operations within Varnish's execution of your configuration.

   ```
   # If you are using the "include" keyword
   include "myACL1.vcl";

   # And/or if you are using an actual ACL block
   acl local {
     "localhost";
     "192.168.1.0"/24; /* and everyone on the local network */
     ! "192.168.1.23"; /* except for the dial-in router */
   }

   sub vcl_recv {
     # block any requests to Admin pages not from local IPs
     if (req.url ~ "^/admin" && client.ip !~ local) {
       error 403 "Forbidden";
     }
   }
   ```

4. Upload the file (/guides/vcl/uploading-custom-vcl) in the Varnish Configuration area of your service.

Shielding Caveats

Be aware that if you've enabled shielding (/guides/performance-tuning/shielding#enabling-shielding), you need to ensure the client IP check is only executed at the edge. For example:

```
sub vcl_recv {
  # block any requests to Admin pages not from local IPs
  if (req.url ~ "^/admin" && client.ip !~ local && !req.http.Fastly-FF){
    error 403 "Forbidden";
  }
}
```

The `client.ip` provides the source address connecting to Fastly. In the case of Origin Shielding (/guides/about-fastly-services/about-fastlys-origin-shielding-features), that address gets overwritten as one Fastly POP (https://www.fastly.com/network-map) connects to another. The `Fastly-FF` header in the above example ensures this code is not executed on the shield server because it gets added by the edge node.

You can create different behaviors based on any other attributes from a request as well, such as location and cookie presence.

---

§ VCL regular expression cheat sheet (/guides/vcl/vcl-regular-expression-cheat-sheet)

IMPORTANT: Varnish (/guides/vcl/) regular expressions are case sensitive.

Basic matching

```
req.url == "phrase"
```

Matches only if `req.url` is exactly `phrase`.

```
req.url ~ "phrase"
```

Matches `phrase` anywhere.

Matching at the beginning or end of a string

```
req.http.host ~ "^www"
```

Matches if `req.http.host` starts with `www`.

```
req.url ~ "\.jpg$"
```

Matches if `req.url` ends with `.jpg`.

Multiple matches

```
req.url ~ "\.(png|jpg|css|js)$"
```

Matches if `req.url` ends with either `.png`, `.jpg`, `.css`, or `.js`.

```
req.url ~ "\.php(\?.*)?$"
```

Matches if `req.url` ends with `.php`, `.php?foo=bar` or `.php?`, but not `.phpa`.

NOTE: You can also use `req.url.ext` to find the file extension specified in a URL. For example, in the request `www.example.com/1/hello.gif?foo=bar`, `req.url.ext` will return `gif`. See our Miscellaneous VCL features (/guides/vcl/miscellaneous-VCL-extensions) guide for more information.

```
req.url ~ "\.[abc]server$"
```

Matches if `req.url` ends with `.aserver`, `.bserver` or `.cserver`.

Matching wildcards

```
req.url ~ "jp.g$"
```

Matches if `req.url` ends with `jpeg`, `jpag`, and `jp0g`, but doesn't match if `req.url` ends with `jpg`. It also matches if any other character is between the `jp` and the `g`.

```
req.url ~ "jp.*g$"
```

Matches `jp` followed by 0 or more random characters ending with the letter `g` (`jpeg`, `jpg`, and `jpeeeeg` all match).

Capturing matches

```
set req.http.Foo = "abbbccccc";
if (req.http.Foo ~ "^(a+)(b+)(c+)") {
  set resp.http.match0 = re.group.0; # now equals 'abbbccccc'
  set resp.http.match1 = re.group.1; # now equals 'a'
  set resp.http.match2 = re.group.2; # now equals 'bbb'
  set resp.http.match3 = re.group.3; # now equals 'ccccc'
}
```

The `re.group.[0-9]` objects allow you to capture matches. The `re.group.0` object evaluates to the entire matched string even if no capture groups have been supplied. You can use these objects to replace this example:

```
if (req.url ~ "(?i)\?.*some_query_arg=([^&]*)") {
  set req.http.Thing-I-Want = regsub(req.url, "(?i)\?.*some_query_arg=([^&]*).*", "\1");
}
```

You can use `re.group` to greatly simplify the previous example:

```
if (req.url ~ "(?i)\?.*some_query_arg=([^&]*)") {
  set req.http.Thing-I-Want = re.group.1;
}
```

You could even get really fancy and do something like this:

```
set req.http.Thing-I-Want = if(req.url ~ "(?i)\?.*some_query_arg=([^&]*)", re.group.1, "");
```

Replacing content

```
set req.http.host = regsub(req.http.host, "^www\.","");
```

Removes a leading `www.` from the host, if present.

```
set req.http.api-test = regsub(req.http.host, "^www.","api.");
```

Sets the `api-test` header to contain the host-header, but replaces a leading `www.` with `api`:

```
Host: www.example.com ->
Host: www.example.com
api-test: api.example.com
Host: example.com ->
Host: example.com
api-test: example.com
```

• Guides (/guides/) > Developer's tools (/guides/devtools) > Tutorials (/guides/tutorials/)

§ Cache control tutorial (/guides/tutorials/cache-control-tutorial)

You are in full control of how Fastly caches your resources. The most preferred way of instructing Fastly is to use backend HTTP headers. The other way is to use Varnish Configuration Language (VCL).

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Backend HTTP headers

You can set four different types of HTTP headers which will have different effects on our caches and on web browsers. If you use more than one type, they are prioritized in the order listed below:

Surrogate-Control

Format:

```
Surrogate-Control: max-age=(time in seconds)
```

Example:

```
Surrogate-Control: max-age=3600
```

will cache something on our caches for one hour. This header gets stripped and is only visible to Fastly caches.

Cache-Control: s-maxage

Format:

```
Cache-Control: s-maxage=(time in seconds)
```

This is the same as `Surrogate-Control`, except the header is not stripped and will be respected by Fastly caches and any caches between Fastly and the browser, but not the browser itself.

Cache-Control: max-age

Format:

```
Cache-Control: max-age=(time in seconds)
```

This header will be respected by Fastly caches, any caches between Fastly and the browser, and the browser itself.

Expires

This header caches content until it expires as specified. It will be respected by Fastly caches, any caches between Fastly and the browser and the browser itself. Please read section 14.21 of RFC2616 (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21) for an explanation of the format.

Do not cache

If you don't want certain resources cached, set the header as follows:

```
Cache-Control: private
```

If you just set `max-age=0` or an `Expires` in the past, there is still no guarantee that the content won't be used to satisfy multiple outstanding requests at the same time. Also, an expired (stale) object might be used in case of errors or while a request to your backend for the same object is already in progress.

More Complicated Examples

Say that you want Fastly to cache your resources forever but send headers to browsers so that they don't cache it at all (so that every browser miss hits Fastly but isn't a cache miss from your service).

The best way to do this would be to send us both the `Cache-Control` header as you want it to go to the browsers, and use `Surrogate-Control` to tell us how long to cache for. So for instance:

```
Cache-Control: no-cache, no-store, must-revalidate
Surrogate-Control: max-age=3600
Pragma: no-cache
Expires: 0
```

Except for when the `Cache-Control` header is set to `private`, the `Surrogate-Control` header takes priority over `Cache-Control`, but unlike `Cache-Control` it is stripped so the browsers don't see it.

Example backend configs

Apache Config

If you are using Apache, the easiest way to add headers is to use the mod_expires module (http://httpd.apache.org/docs/current/mod/mod_expires.html). For example, to cache GIF images for 75 minutes after the image was last accessed by the cache server, you would add a directive like this under the VirtualHost (or globally). For example:

```
ExpiresActive On
ExpiresByType image/gif "access plus 1 hours 15 minutes"
```

You can also cache whole URL subtrees. For example:

```
<Location "/css">
  ExpiresActive On
  ExpiresDefault "access plus 1 year"
</Location>

<Location "/static/">
  ExpiresActive On
  ExpiresDefault "access plus 1 day"
</Location>
```

NGINX Configuration

To configure NGINX, add the expires directive. For example:

```
location ~* \.(js|css|png|jpg|jpeg|gif|ico)$ {
   expires 1h;
}
```

Alternatively, if you need more flexibility in modifying headers you can try the HttpHeadersMore Module (http://nginx.org/en/docs/http/ngx_http_headers_module.html).

Amazon S3 configuration

By default, S3 doesn't have a facility for setting Cache-Control headers across multiple objects, so you will have to do this file-by-file using the S3Explorer, or in an automated fashion by using a cloud library like boto. Remember that you can combine long cache time with instant purges to enhance your performance.

★ **TIP:** While it's difficult to get S3 to set `Surrogate-Control`, you can set `x-amz-meta-surrogate-control` (/guides/purging/setting-surrogate-key-headers-for-amazon-s3-origins) instead on origin and Fastly will honor that.

```
from boto.s3.connection import S3Connection

connection = S3Connection('aws access key', 'aws secret key')

buckets = connection.get_all_buckets()

for bucket in buckets:
    for key in bucket.list():
        print('%s' % key)

        if key.name.endswith('.jpg'):
            contentType = 'image/jpeg'
        elif key.name.endswith('.png'):
            contentType = 'image/png'
        else:
            continue

        key.metadata.update({
            'Content-Type': contentType,
            'Cache-Control': 'max-age=864000'
        })
        key.copy(
            key.bucket.name,
            key.name,
            key.metadata,
            preserve_acl=True
        )
```

IMPORTANT: The above example provides an S3 configuration option for customers with small- to medium-sized buckets. However, it iterates over every object in those buckets. If you have millions of objects this may not be the right approach. For millions of objects, we would recommend using Varnish Configuration Language (/guides/vcl/guide-to-vcl) (VCL). Please contact us (mailto:support@fastly.com) for assistance.

Custom Headers in Programming Languages and Frameworks

PHP

More information: http://php.net/manual/en/function.header.php (http://php.net/manual/en/function.header.php)

Example: add this to your PHP code before you send any output to cache certain HTML for an hour

```
header('Cache-Control: max-age=3600');
```

Django

More information: https://docs.djangoproject.com/en/dev/ref/request-response/#setting-headers (https://docs.djangoproject.com/en/dev/ref/request-response/#setting-headers)

Example:

```
response = HttpResponse()
response['Cache-Control'] = 'max-age=3600'
```

Sinatra

More information: http://sinatra.rubyforge.org/doc/ (http://sinatra.rubyforge.org/doc/)

Example:

```
get '/' do
    headers['Cache-Control'] = 'max-age=3600'
end
```

★ **TIP:** Expiration times in these examples are provided for guidance only. You can use longer expirations coupled with our purging API (/api/purge) to make your site faster and your backend less loaded.

§ Enabling URL token validation (/guides/tutorials/enabling-url-token-validation)

Token validation allows you to create URLs that expire. Tokens are generated within your web application and appended to URLs in a query string. Requests are authenticated at Fastly's edge instead of your origin server. When Fastly receives a request for the URL, the token is validated before serving the content. After a configurable period of time, the token expires.

Adding custom VCL

To enable token validation, you'll need to create a Varnish configuration (/guides/vcl/uploading-custom-vcl) named `vcl_recv` and add the following example code to it.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

```
/* make sure there is a token */
if (req.url !~ ".+\?.*token=(\d{10,11})_([^&]+)") {
  error 403;
}

/* extract token expiration and signature */
set req.http.X-Exp = re.group.1;
set req.http.X-Sig = re.group.2;

/* validate signature */
if (req.http.X-Sig == regsub(digest.hmac_sha1(digest.base64_decode("YOUR%SECRET%KEY%IN%BASE64%HERE"),
req.url.path req.http.X-Exp), "^0x", "")) {

  /* check that expiration time has not elapsed */
  if (time.is_after(now, std.integer2time(std.atoi(req.http.X-Exp)))) {
    error 410;
  }

} else {
  error 403;
}

/* cleanup variables */
unset req.http.X-Sig;
unset req.http.X-Exp;
```

WARNING: You must replace `YOUR%SECRET%KEY%IN%BASE64%HERE` in the example VCL with your own randomly generated secret key. The example key will intentionally cause an error if you use it. Anyone who learns this key can bypass your token validation, so it's critical that you keep this key secret.

The key found in `digest.hmac_sha1` can be any string. The one in this example was generated with the command `openssl rand -base64 32`.

A token is expected in the `?token=` GET parameter. Tokens take the format `[expiration]_[signature]` and look like this:

```
1441307151_4492f25946a2e8e1414a8bb53dab8a6ba1cf4615
```

The full request URL looks like this:

```
http://www.example.com/foo/bar.html?token=1441307151_4492f25946a2e8e1414a8bb53dab8a6ba1cf4615
```

What the custom VCL does

The custom VCL code checks for two things:

- Is the current time greater than the expiration time specified in the token?
- Does our signature match the signature of the token?

If the signature is invalid, Varnish returns a 403 response. If the signature is valid but the expiration time has elapsed, Varnish returns a 410 response. The different response codes are helpful for debugging.

Configuring your application

You'll need to write custom code in your application to generate tokens and authenticate with Varnish. We provide examples in our token functions (https://github.com/fastly/token-functions) repository on GitHub. Review the examples in the repository to learn how to generate custom tokens within your application.

Testing

To test your configuration, append a token generated by your application to a URL in a query string. For example:

`http://www.example.com/foo/bar.html?token=1441307151_4492f25946a2e8e1414a8bb53dab8a6ba1cf4615`

If the token is valid, you will receive a normal response. If it is invalid, you will receive a 403 response.

Troubleshooting NUL bytes

You should verify that your secret key is devoid of NUL bytes. If the base64-decoded string contains a NUL byte (0x00), then that byte and any bytes following it will not be included in the response. See base64 decoding (/guides/vcl/cryptographic-and-hashing-related-vcl-functions#base64-decoding) for more information.

• Guides (/guides/) > Diagnostics and performance (/guides/diagnostics) > Streaming logs (/guides/streaming-logs/)

§ Custom log formats (/guides/streaming-logs/custom-log-formats)

Fastly provides two versions of custom log formats. All logs use the version 1 custom log format by default, but you can upgrade to the version 2 custom log format and then make version 2 look like version 1 for the sake of continuity. We've described the key advantages of the version 2 custom log format below.

⚠ **WARNING:** To upgrade to version 2 custom log formats, contact support@fastly.com (mailto:support@fastly.com) for instructions. Logging objects using version 2 formatting cannot be downgraded to version 1 because version 1 only provides a limited subset of logging directives and is not strictly compatible with Apache log format.

Version 1 log format

This table details version 1 of Fastly's custom log formats.

String	Description
`b`	The content size of the response, calculated using the `Content-Length` header rather than actually checking the length of the response (and may therefore be wrong).
`h`	The remote IP address.
`l`	The remote log name. Always returns the hardcoded value `"-"`.
`r`	The HTTP verb and request path (e.g., `GET /index.html`). Unlike Apache and version 2 log formats, the protocol version is not included.
`>s`	The status of the last request.
`t`	The time the request was received, in Unix `ctime` format (e.g., `Thu, 01 Jan 1970 00:00:00 GMT`) rather than Apache's Standard English format (e.g., `01/Jan/1970:00:00:00 -0700`).
`u`	The remote user. Always returns the hardcoded value `"-"`.

Version 2 log format

This table details version 2 of Fastly's custom log formats.

String	Description
`%`	The percent sign.
`a`	The client IP address of the request.
`A`	The local IP address.
`B`	The size of response in bytes, excluding HTTP headers.
	The size of response in bytes, excluding HTTP headers. In Common Log Format

`b`	(https://httpd.apache.org/docs/trunk/logs.html#common) (CLF), that means a `"-"` rather than a `0` when no bytes are sent.
`{Foobar}C`	The contents of cookie `Foobar` in the request sent to the server.
`D`	The time taken to serve the request, in microseconds.
`{FOOBAR}e`	The contents of the environment variable `FOOBAR`. Always returns `NIL`.
`f`	The filename.
`h`	The remote hostname.
`H`	The request protocol.
`{Foobar}i`	The contents of `Foobar:` header lines in the request sent to the server.
`I`	Bytes received, including request and headers. Cannot be zero.
`k`	The number of keepalive requests handled on this connection. Always returns `0`.
`l`	The remote logname (from identd, if supplied). This will return a dash unless mod_ident is present and IdentityCheck is set On. Always returns `"-"`.
`m`	The request method.
`{Foobar}n`	The contents of note `Foobar` from another module. Always returns `NIL`.
`{Foobar}o`	The contents of `Foobar:` header lines in the reply.
`O`	Bytes sent, including headers. Cannot be zero.
`p`	The canonical port of the server serving the request. Always returns `80`.
`{format}p`	The canonical port of the server serving the request. Valid formats are `canonical`, `local`, or `remote`. Returns `80` for HTTP requests and `443` for HTTPS requests.
`P`	The process ID of the child that serviced the request. Always returns `NIL`.
`{format}P`	The process ID or thread ID of the child that serviced the request. Valid formats are `pid`, `tid`, and `hextid`. Always returns `NIL`.
`q`	The query string (prepended with a `?` if a query string exists, otherwise an empty string).
`r`	The first line of the request.
`R`	The handler generating the response (if any).
`s`	The status. For requests that got internally redirected, this is the status of the *original* request. Use `%>s` for the final status.
`t`	The time the request was received, in Standard English format (e.g., `01/Jan/1970:00:00:00 -0700`). The last number indicates the timezone offset from GMT.
`{format}t`	The time, in the form given by `format`, which should be in `strftime(3)` format (potentially localized). If the format starts with `begin:` (the default) the time is taken at the beginning of the request processing. If it starts with `end:` it is the time when the log entry gets written, close to the end of the request processing. In addition to the formats supported by `strftime(3)`, the following format tokens are supported: `sec` (number of seconds since the Epoch), `msec` (number of milliseconds since the Epoch), `usec` (number of microseconds since the Epoch), `msec_frac` (millisecond fraction), and `usec_frac` (microsecond fraction).
`T`	The time taken to serve the request, in seconds.
`u`	The remote user if the request was authenticated. May be bogus if return status (`%s`) is 401 (unauthorized). Always returns `"-"`.
`U`	The URL path requested, not including any query string.
`v`	The canonical `ServerName` of the server serving the request.
`V`	The server name according to the `UseCanonicalName` setting.
`{vcl}V`	The literal VCL to include without quoting (a Fastly extension).
`X`	The connection status when response is completed. Statuses include `X` (connection aborted before the response completed), `+` (connection may be kept alive after the response is sent), and `-` (connection will be closed after the response is sent).

Advantages of using the version 2 custom log format

The key advantages of using the version 2 custom log format include the following:

- Log lines are generated in `vcl_log` instead of `vcl_deliver` to allow us to accurately set the various size variables because `vcl_log` is run after the object has been delivered to the browser.

- The `%t` time directive is compatible with Apache log format. In version 1, we used a non-standard time format.
- The `%r` "first line of request" directive is compatible with Apache log format. In version 1, we incorrectly left off the protocol.
- When using the `%b` directive, which represents the size of a response in bytes, excluding HTTP headers is more accurate. In version 1, we used the reported Content-Length from the origin, which could be inaccurate (especially with ESI (/guides/performance-tuning/using-edge-side-includes)).
- We've added all Apache logging directives that make sense. In version 1, we used a smaller subset.

Making version 2 logs look like version 1

The default logging format for version 1 is as follows:

```
%h %l %u %t %r %>s
```

Most of the directives in version 2 are exactly the same - only `%t` and `%r` are different. After you upgrade to version 2 log formats, you can recreate the appearance of version 1 logs using the new `%{..}V` directive, which allows you to specifically include VCL in logging directives:

```
%h %l %u %{now}V %{req.request}V %{req.url}V %>s
```

In addition, if you are using the `%b` directive in version 1, then you can use this directive instead:

```
%{resp.http.Content-Length}V
```

---

§ Log streaming: Amazon S3 (/guides/streaming-logs/log-streaming-amazon-s3)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to Amazon Simple Storage Service (https://aws.amazon.com/s3/) (Amazon S3). Amazon S3 is a static file storage service used by developers and IT teams. You can also use the instructions in this guide to configure log streaming to another S3-compatible service.

Prerequisites

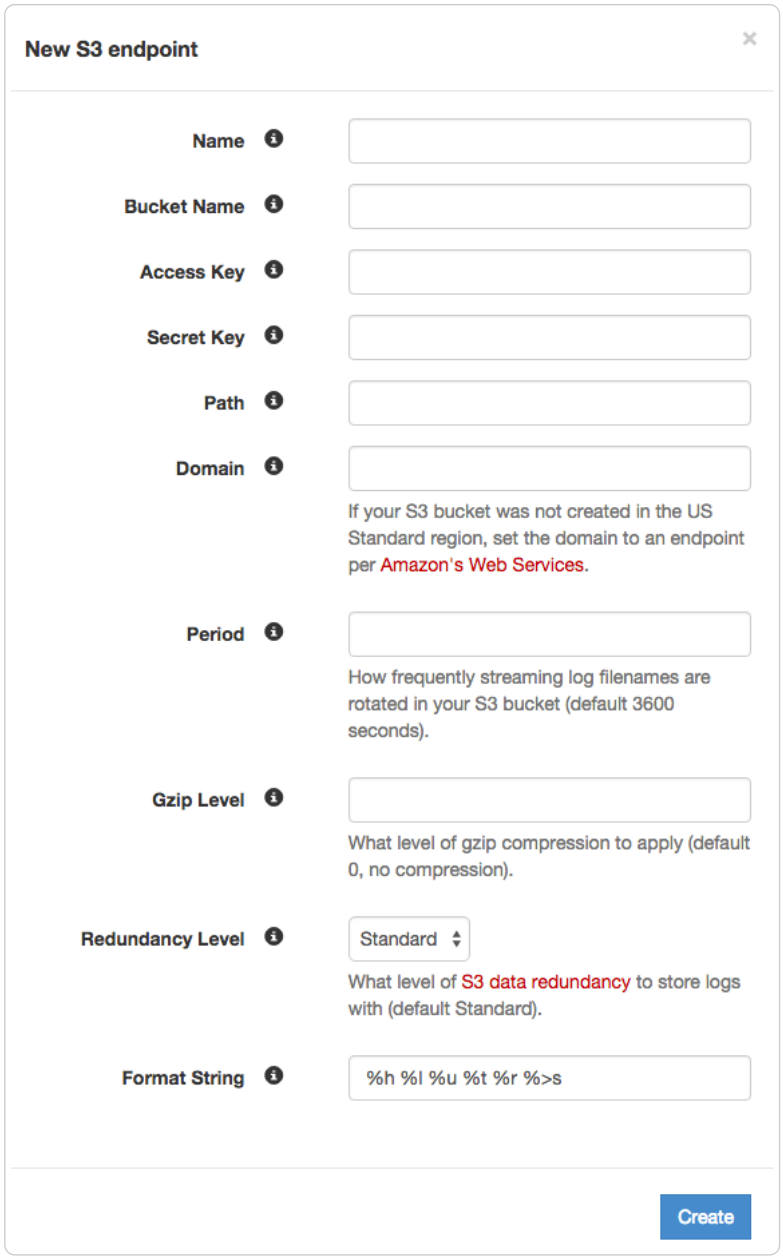
Before adding Amazon S3 as a logging endpoint in the Fastly application, we recommend creating an Identity and Access Management (IAM) user in Amazon S3 specifically for Fastly. Grant the user `ListBucket`, `GetObject`, and `PutObject` permissions for the directory in which you want to store logs. For more information, see Amazon's Getting Your Access Key ID and Secret Access Key (http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSGettingStartedGuide/AWSCredentials.html) page.

Adding Amazon S3 as a logging endpoint

After you've registered for an Amazon S3 account and created an IAM user in Amazon S3, follow these instructions to add Amazon S3 as a logging endpoint:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **Amazon S3 endpoints** area and click **New**.

The New S3 endpoint window appears.



3. Fill out the **New S3 endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Bucket Name** field, type the name of the Amazon S3 bucket in which to store the logs.
   - In the **Access Key** field, type the access key associated with the Amazon S3 bucket. See Amazon's S3 Getting Started Guide (http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSGettingStartedGuide/AWSCredentials.html) for more information.
   - In the **Secret Key** field, type the secret key associated with the Amazon S3 bucket. See Amazon's S3 Getting Started Guide (http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSGettingStartedGuide/AWSCredentials.html) for more information.
   - In the **Path** field, optionally type the path within the bucket to store the files. The path ends with a trailing slash. If this field is left empty, the files will be saved in the bucket's root path.
   - In the **Domain** field, optionally type the domain of the Amazon S3 endpoint. If your Amazon S3 bucket was not created in the US Standard region, you must set the domain to match the appropriate endpoint URL. Use the table in the S3 section of the Regions and Endpoints (http://docs.aws.amazon.com/general/latest/gr/rande.html#s3_region) Amazon S3 documentation page. If you want to use an S3-compatible storage system (such as Dreamhost's DreamObjects (https://www.dreamhost.com/cloud/storage/)), set the domain to match the domain name for that service (for example, in the case of DreamObjects, the domain name would be `objects.dreamhost.com`).
   - In the **Period** field, optionally type an interval (in seconds) to control how frequently your log files are rotated. This value defaults to `3600` seconds.
   - In the **Gzip Level** field, optionally type the level of gzip compression you want applied to the log files. You can specify any whole number from `1` (fastest and least compressed) to `9` (slowest and most compressed). This value defaults to `0` (no compression).

- From the **Redundancy Level** menu, select a setting. This value defaults to **Standard**. See Amazon's Using Reduced Redundancy Storage Guide (http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingRRS.html) for more information on using reduced redundancy storage.
- In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming.html#using-format-strings) for more information.

4. Click **Create** to create the new logging endpoint.

Keep in mind that although Fastly continuously streams logs into Amazon S3, the Amazon S3 website and API do not make files available for access until after the upload is complete.

§ Log streaming: Cloud Files (/guides/streaming-logs/log-streaming-cloudfiles)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log file to Cloud Files (https://www.rackspace.com/cloud/files). Operated by Rackspace, Cloud Files is a file storage service used by developers and IT teams.

Prerequisites

If you don't already have a Rackspace Cloud account, you'll need to register (https://cart.rackspace.com/cloud) for one. Follow the instructions on Rackspace's website (https://cart.rackspace.com/cloud).

Creating a Cloud Files user and container

Start by creating a Cloud Files user with restricted permissions via Rackspace's cloud control panel (https://mycloud.rackspace.com/).

1. Log in to Rackspace's cloud control panel (https://mycloud.rackspace.com/).
2. From the user account menu, select **User Management**.
3. Click **Create User** and fill in all appropriate details.
4. In the **Product Access** section, set **User Role** to **Custom**.
5. Review the **Product Access** list. For all items in the **Product** column, set **Role** to **No Access** except the **Files** item.
6. Set the **Files** item **Role** to **Admin**. This will allow you to create the files to store the logs in, but not access any other services.

Next, find the API key for your Cloud Files account. You'll use this later to authenticate using the Cloud Files API.

1. From the user account menu, select **Account Settings**.
2. Show the API key in the **Login details** and make a note of it.

Now that you've created the Cloud Files user and found the API key, you can set up a Cloud Files container.

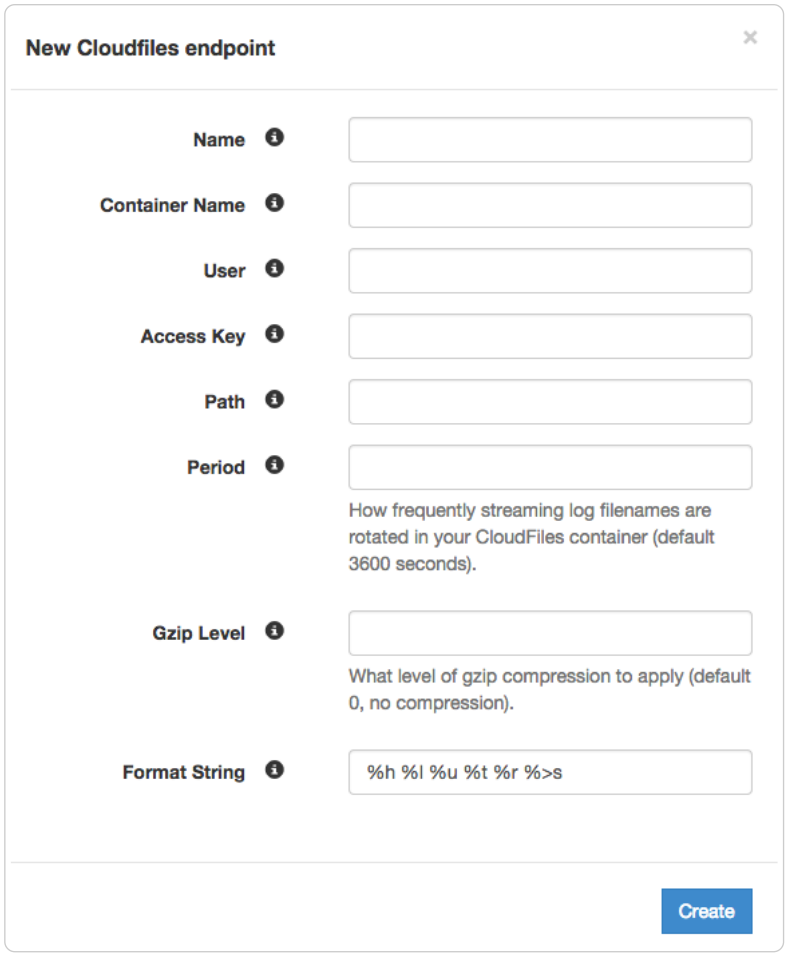
1. From the **Storage** menu, select **Files**.
2. Click **Create Container**.
3. Assign the container a meaningful name like `Fastly logs - my service`.
4. Choose a region to keep the files in and make sure the container is private.
5. Click **Create Container**.

Adding a Cloud Files logging endpoint

Once you have created the Cloud Files user and container, follow these instructions to add Cloud Files as a logging endpoint:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **Cloudfiles** area and click **New**.

The New Cloudfiles endpoint window appears.



3. Fill out the **New Cloudfiles endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Container Name** field, type the name of the Cloud Files container in which to store the logs.
   - In the **User** field, type the username of the Cloud Files user you created above.
   - In the **Access Key** field, type the API key you found above.
   - In the **Path** field, optionally type the path within the container to store the files. The path ends with a trailing slash. If this field is left empty, the files will be saved in the container's root path.
   - In the **Period** field, optionally type an interval (in seconds) to control how frequently your log files are rotated. This value defaults to `3600` seconds.
   - In the **Gzip Level** field, optionally type the level of gzip compression you want applied to the log files. You can specify any whole number from `1` (fastest and least compressed) to `9` (slowest and most compressed). This value defaults to `0` (no compression).
   - In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.
4. Click **Create** to create the new logging endpoint.

§ Log streaming: FTP (/guides/streaming-logs/log-streaming-ftp)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to password-protected and anonymous FTP servers.

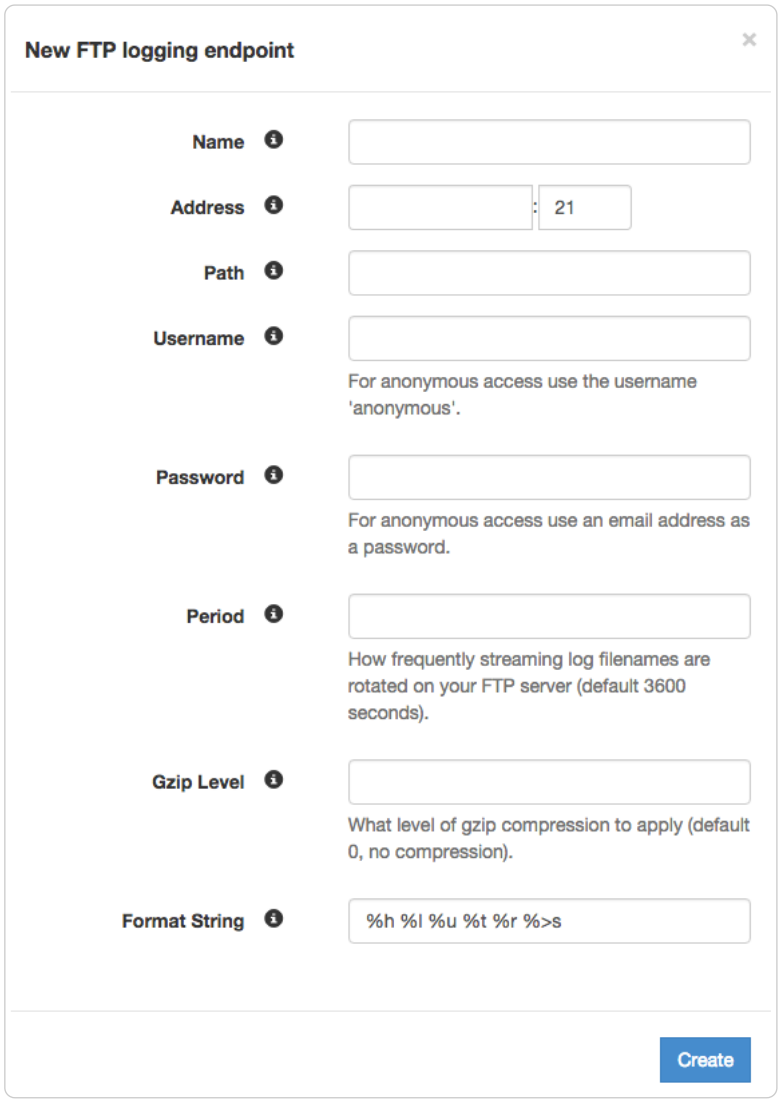
> **NOTE:** This logging endpoint is disabled by default. To enable this endpoint for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Adding FTP as a logging endpoint

After Fastly support has enabled the FTP endpoint for your account, follow these instructions to add FTP as a logging endpoint:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **FTP Logging endpoints** area and click **New**.

The New FTP logging endpoint window appears.



3. Fill out the **New FTP logging endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Address** field, type the hostname or IP address of the FTP server. In the port field, type the port number you're using for FTP (the default is `21`).
   - In the **Path** field, optionally type the path to store the files. The path ends with a trailing slash. If this field is left empty, the files will be saved in the root path.
   - In the **Username** field, type the username used to authenticate to the FTP server. For anonymous access, use the username `anonymous`.
   - In the **Password** field, type the password used to authenticate to the FTP server. For anonymous access, use an email address as the password.
   - In the **Period** field, optionally type an interval (in seconds) to control how frequently your log files are rotated. This value defaults to `3600`

seconds.

- In the **Gzip Level** field, optionally type the level of gzip compression you want applied to the log files. You can specify any whole number from `1` (fastest and least compressed) to `9` (slowest and most compressed). This value defaults to `0` (no compression).
- In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.

4. Click **Create** to create the new logging endpoint.

---

§ Log streaming: Google Cloud Storage (/guides/streaming-logs/log-streaming-google-cloud-storage)

Fastly's Real-Time Log Streaming feature (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) can send log files to Google Cloud Storage (https://cloud.google.com/storage/) (GCS). GCS is an online file storage service used for storing and accessing data on Google's infrastructure. One advantage of using GCS is that you can use Google BigQuery (https://cloud.google.com/bigquery/) to analyze the log files.

Prerequisites

Before adding GCS as a logging endpoint in the Fastly application, you will need to register for a GCS account, create a bucket and service account on Google's website, and obtain the `private_key` and `client_email` from the JSON file associated with the service account.

Creating a GCS bucket

You can create a new GCS bucket to hold the logs, or you can use an existing bucket. Be sure to note the name of the bucket as you will need it later. To learn how to create a GCS bucket, see Google's guide on creating a bucket (https://cloud.google.com/storage/docs/getting-started-console#create_a_bucket).

Creating a service account

GCS uses service accounts for third-party application authentication. You will need to create a new service account on Google's website. To learn how to create a service account, see Google's guide on generating a service account credential (https://cloud.google.com/storage/docs/authentication#generating-a-private-key). When you create the service account, be sure to set the **Key Type** to `JSON`.

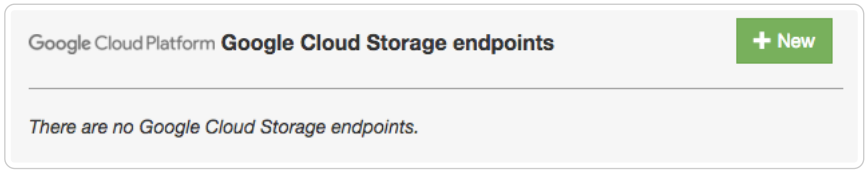
Obtaining the private key and client email

After you create the service account, a JSON file will be downloaded to your computer. This file contains the credentials for the GCS service account you just created. Open the file with a text editor and make a note of the `private_key` and `client_email`.

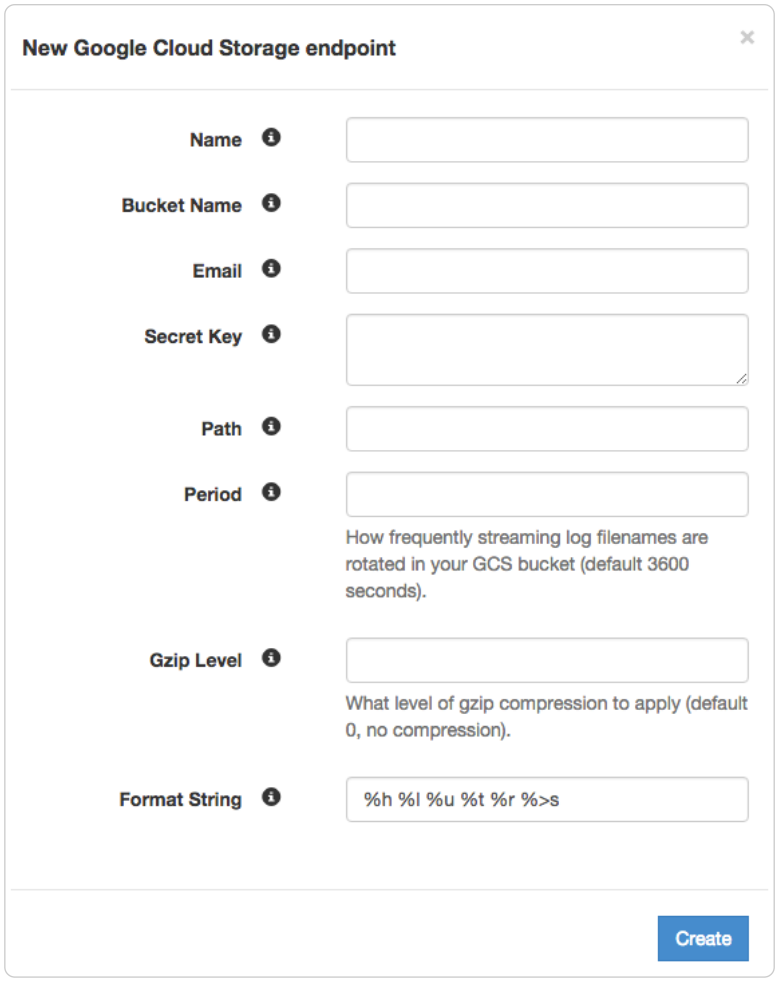
Adding GCS as a logging endpoint

After Fastly support has enabled the GCS endpoint for your account, follow these instructions to add GCS as a logging endpoint:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **Google Cloud Storage endpoints** area and click **New**.



The New Google Cloud Storage endpoint window appears.



3. Fill out the **New Google Cloud Storage endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Bucket Name** field, type the name of the GCS bucket in which to store the logs.
   - In the **Email** field, type the `client_email` address listed in the JSON file associated with the service account you created on Google's website.
   - In the **Secret Key** field, type the `private_key` value listed in the JSON file associated with the service account you created on Google's website. We strip out the JSON newline escape characters for you so don't worry about removing them.
   - In the **Path** field, optionally type the path within the bucket to store the files. Specify a directory by ending the path with a trailing slash (`/`). Leaving this field empty saves the files in the bucket's root path.
   - In the **Period** field, optionally type an interval (in seconds) to control how frequently your log files are rotated. This value defaults to `3600` seconds.
   - In the **Gzip Level** field, optionally type the level of gzip compression you want applied to the log files. You can specify any whole number from `1` (fastest and least compressed) to `9` (slowest and most compressed). This value defaults to `0` (no compression).
   - In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.
4. Click **Create** to create the new logging endpoint.

§ Log streaming: Log Shuttle (/guides/streaming-logs/log-streaming-log-shuttle)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to Log Shuttle (https://github.com/heroku/log-shuttle). Log Shuttle is an open source application designed to provide simpler encrypted and authenticated log delivery.

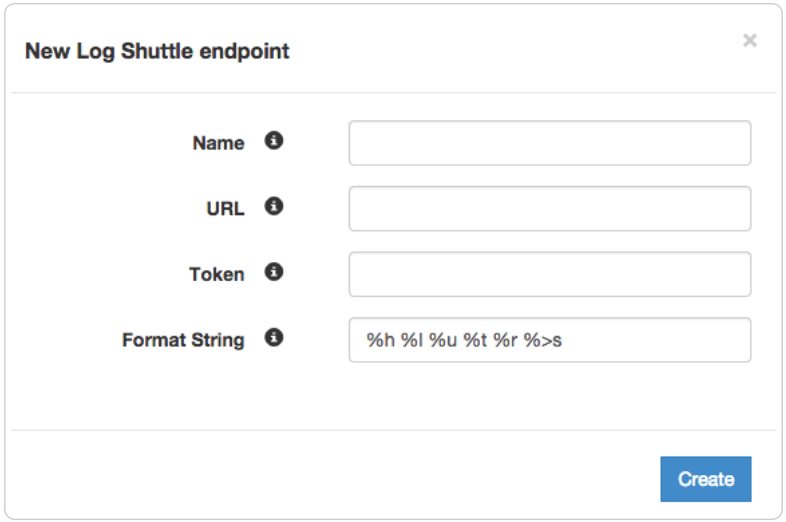
NOTE: This logging endpoint is disabled by default. To enable this endpoint for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Adding Log Shuttle as a logging endpoint

After Fastly support has enabled the Log Shuttle endpoint for your account, follow these instructions to add Log Shuttle as a logging endpoint:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **Log Shuttle endpoints** area and click **New**.

The New Log Shuttle endpoint page appears.



3. Fill out the **New Log Shuttle endpoints** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **URL** field, type the URL to which log data will be sent (e.g., `https://logs.example.com/`).
   - In the **Token** field, optionally type the data authentication token. This is required for some endpoints like Heroku's Log Integration.
   - In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.
4. Click **Create** to create the new logging endpoint.

§ Log streaming: Logentries (/guides/streaming-logs/log-streaming-logentries)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to Logentries (https://logentries.com/). Logentries is a real-time log management and analytics system that you can use to monitor your Fastly logs.

Prerequisites

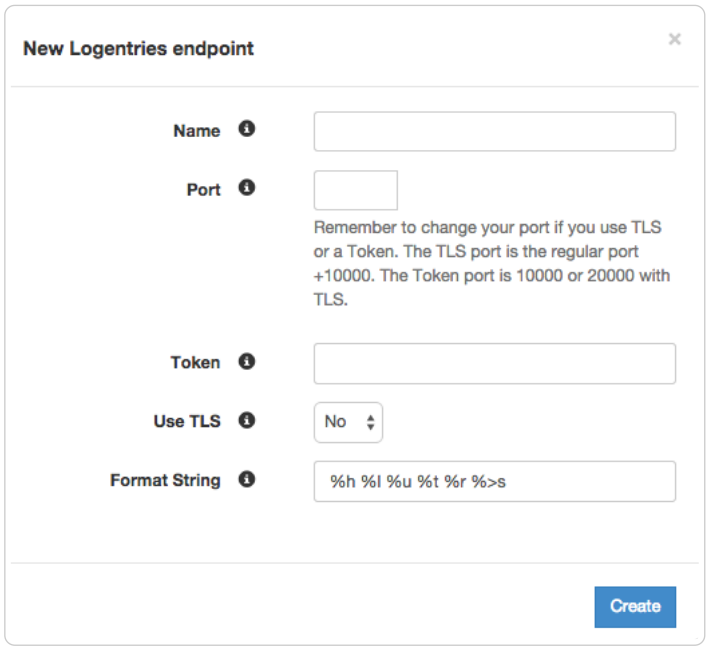
You'll need to register for a Logentries account and create a new log in the Logentries application. Follow the instructions provided on the Logentries website (https://logentries.com/doc/fastly/). When creating the new log, be sure to select **Manual Configuration** and **Token TCP**. Make a note of the token provided in the Logentries interface — you'll need it in the next section.

Adding Logentries as a logging endpoint

After you've created a new log in Logentries and found the token, follow these instructions to add Logentries as a logging endpoint in the Fastly application:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **Logentries endpoints** area and click **New**.

The New Logentries endpoint window appears.



3. Fill out the **New Logentries endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Port** field, type `20000`.
   - In the **Token** field, type the token you found in the Logentries configuration panel.
   - From the **Use TLS** menu, select **Yes**.
   - In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.
4. Click **Create** to create the new logging endpoint.

Next steps

Logentries maintains the Fastly Community Pack (https://blog.logentries.com/2015/01/fastly-community-pack/) which leverages custom VCL to provide advanced User-Agent statistics, regional statistics, error tracking, and more.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

§ Log streaming: OpenStack (/guides/streaming-logs/log-streaming-openstack)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to OpenStack (http://www.openstack.org/). OpenStack is an open-source platform for cloud-computing that many companies deploy as an infrastructure-as-a-service.

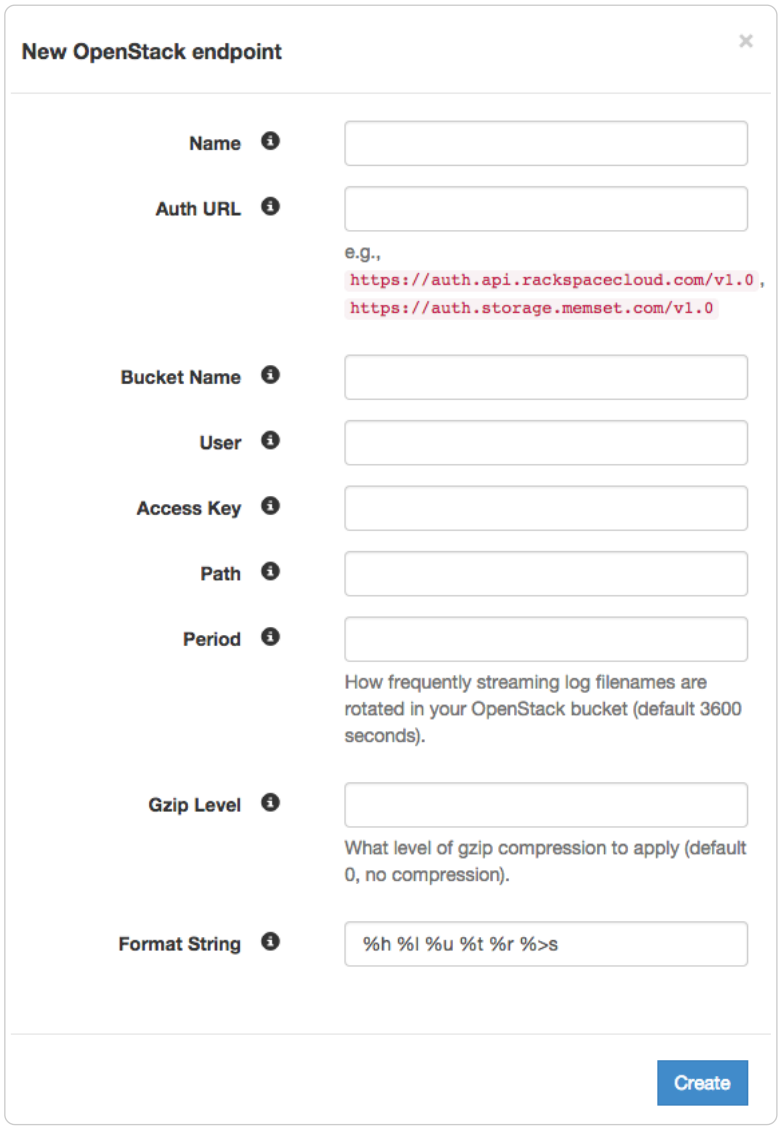
NOTE: This logging endpoint is disabled by default. To enable this endpoint for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Adding OpenStack as a logging endpoint

After Fastly support has enabled the OpenStack endpoint for your account, follow these instructions to add OpenStack as a logging endpoint:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **OpenStack endpoints** area and click **New**.

The New OpenStack endpoint window appears.



3. Fill out the **New OpenStack endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Auth URL** field, type the URL used for OpenStack authentication (e.g., `https://auth.api.rackspacecloud.com/v1.0`).

- In the **Bucket Name** field, type the name of the OpenStack bucket in which to store the logs.
- In the **User** field, type your OpenStack username.
- In the **Access Key** field, type your OpenStack access key.
- In the **Path** field, optionally type the path within the bucket to store the files. The path ends with a trailing slash. If this field is left empty, the files will be saved in the bucket's root path.
- In the **Period** field, optionally type an interval (in seconds) to control how frequently your log files are rotated. This value defaults to `3600` seconds.
- In the **Gzip Level** field, optionally type the level of gzip compression you want applied to the log files. You can specify any whole number from `1` (fastest and least compressed) to `9` (slowest and most compressed). This value defaults to `0` (no compression).
- In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.

4. Click **Create** to create the new logging endpoint.

§ Log streaming: Papertrail (/guides/streaming-logs/log-streaming-papertrail)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to Papertrail (https://papertrailapp.com). Papertrail is a web-based log aggregation application used by developers and IT teams. Instructions for setting up remote log streaming via Papertrail are detailed in the Papertrail setup and configuration documentation (http://help.papertrailapp.com/kb/hosting-services/fastly/).

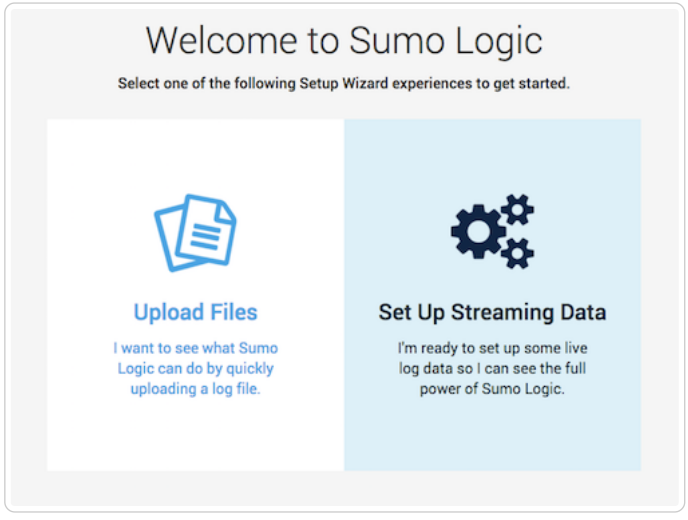
§ Log streaming: Sumo Logic (/guides/streaming-logs/log-streaming-sumologic)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to Sumo Logic (https://www.sumologic.com/). Sumo Logic is a web-based log analytics platform used by developers and IT teams.

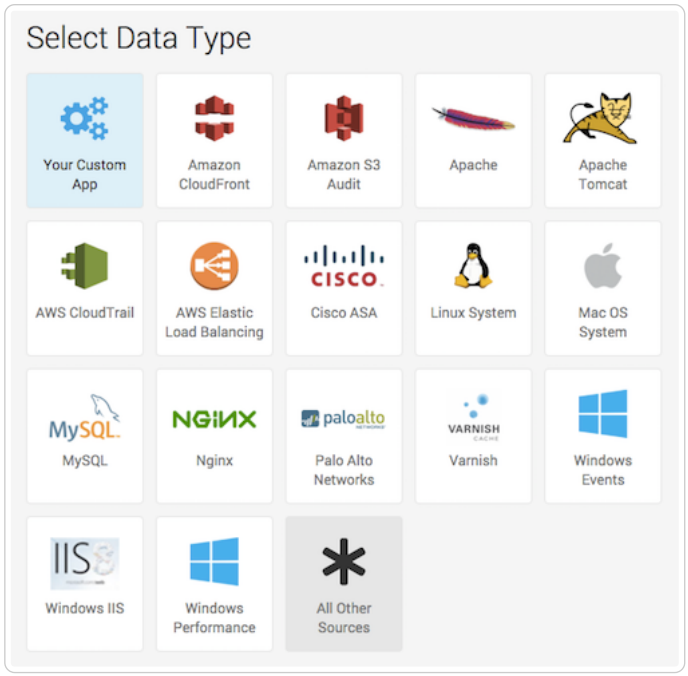
Setting up Sumo Logic

To use Sumo Logic as a logging endpoint, you'll need to create a Sumo Logic account, add a new source, and save the HTTP Source URL. Follow these instructions to add a new source in the Sumo Logic website:

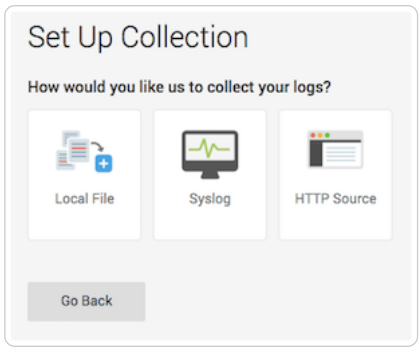
1. The process starts with the Sumo Logic Setup Wizard, which appears immediately after you create your Sumo Logic account. If you already have an account, you can access the wizard by selecting **Setup Wizard** from the **Manage** menu at the top of the Sumo Logic application.



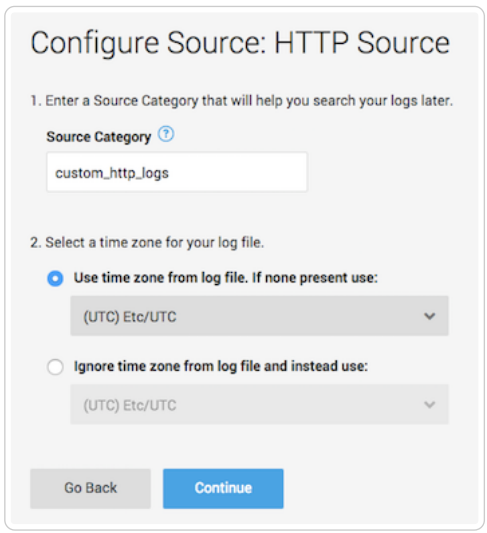
2. Click **Set Up Streaming Data**. The Select Data Type window appears.



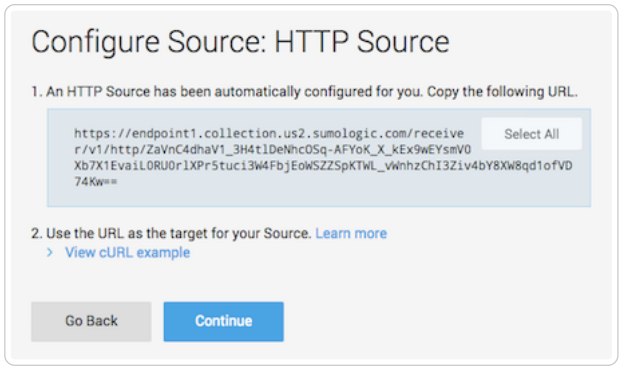
3. Click **All Other Sources**. The Set Up Collection window appears.



4. Click **HTTP Source**. The Configure Source: HTTP Source window appears.



5. In the **Source Category** field, type a human-readable name for the category (e.g., `fastly_cdn`) and select a time zone for your log file.
6. Click **Continue**. The HTTP Source URL appears.



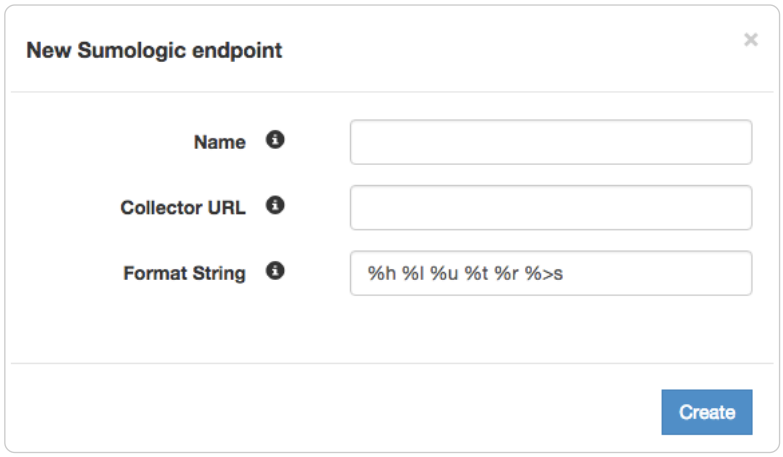
7. Copy the HTTP Source URL. You will enter this value in the Fastly application.
8. Click **Continue**. Sumo Logic will add the new source.

Adding Sumo Logic as a logging endpoint

After you've created a Sumo Logic account and obtained the HTTP Source URL, follow these instructions to add Sumo Logic as a logging endpoint in the Fastly application:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **Sumologic endpoints** area and click **New**.

The New Sumologic endpoint window appears.



3. Fill out the **New Sumologic endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Collector URL** field, type the address of the HTTP Source URL you found in the Sumo Logic website.
   - In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.
4. Click **Create** to create the new logging endpoint.

Troubleshooting

The Sumo Logic logging endpoint is designed for services with sustained levels of traffic. If you aren't seeing any logs in Sumo Logic, try waiting a bit.

---

§ Log streaming: Syslog (/guides/streaming-logs/log-streaming-

syslog)

Fastly's Real-Time Log Streaming (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) feature can send log files to syslog-based logging software. Syslog is a widely used standard for message logging.

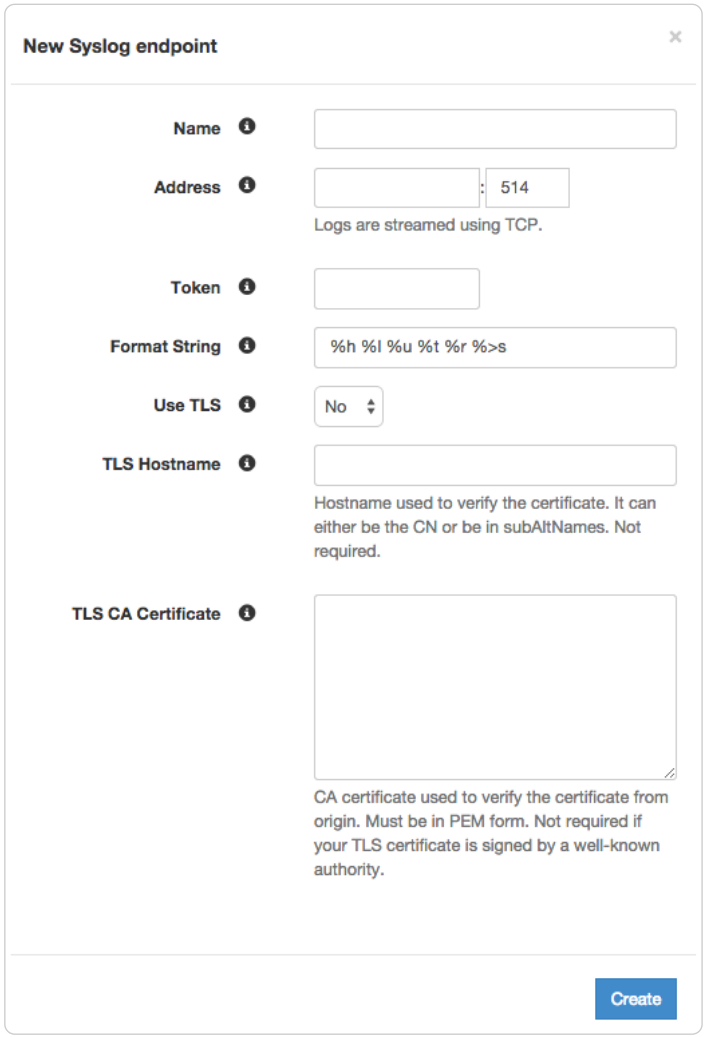
NOTE: Splunk Storm (http://www.splunk.com/) was end-of-lifed on April 1st, 2015. If you are using a Splunk product, please configure it to receive log streams (/guides/streaming-logs/setting-up-remote-log-streaming) over syslog.

Adding syslog as a logging endpoint

Follow these instructions to add syslog as a logging endpoint:

1. Review the information in our Setting Up Remote Log Streaming (/guides/streaming-logs/setting-up-remote-log-streaming) guide.
2. On the **Logging** page, find the **Syslog endpoints** area and click **New**.

The New Syslog endpoint window appears.



3. Fill out the **New Syslog endpoint** fields as follows:
   - In the **Name** field, type a human-readable name for the endpoint.
   - In the **Address** field, type the hostname or IP address and port to which logs should be sent. Be sure this port can receive incoming TCP traffic from Fastly. See the firewall considerations section for more information.

- In the **Token** field, optionally type a string prefix (line prefix) to send in front of each log line.
- In the **Format String** field, optionally type an Apache-style string or VCL variables to use for log formatting. The Apache Common Log format string appears in this field by default. See our guidance on format strings (/guides/streaming-logs/setting-up-remote-log-streaming#using-format-strings) for more information.
- From the **Use TLS** menu, select **No** to disable encryption for the syslog endpoint, or **Yes** to enable it.
- In the **TLS Hostname** field, optionally type the hostname used to verify the syslog server's certificate. This can be either the Common Name (CN) or Subject Alternate Name (SAN).
- In the **TLS CA Certificate** field, optionally copy and paste the Certificate Authority (CA) certificate used to verify that the origin server's certificate is valid. This must be in PEM format. Consider uploading the certificate if it's not signed by a well-known certificate authority. This value is not required if your TLS certificate is signed by a well-known authority.

4. Click **Create** to create the new logging endpoint.

Adding separators or static strings

To insert a separator or other arbitrary string into the syslog endpoint format:

1. Create a new header (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses) with the following fields:
   - From the **Type/Action** menus, select `request` and `set`
   - In the **Destination** field, type any suitable header name (for example, `http.X-Separator`)
   - In the **Source** field, type any special character or string you want (for example, `"|"`)
2. Reference the new header variable in the log format box for your specific provider (for example, `req.http.X-Separator`).

Syslog facility and severity

The syslog output includes the following facility and severity values:

```
facility: local0
severity: info
```

Firewall considerations

Syslog has limited security features. For this reason, it's best to create a firewall for your syslog server and only accept TCP traffic on your configured port from our address blocks. Our list of address blocks is dynamic, so we recommend programmatically obtaining the list from our JSON feed (https://app.fastly.com/public-ip-list) whenever possible.

§ Setting up remote log streaming (/guides/streaming-logs/setting-up-remote-log-streaming)

Fastly's Real-Time Log Streaming feature (/guides/about-fastly-services/about-fastlys-realtime-log-streaming-features) allows you to automatically save logs to a third-party service for storage and analysis. Logs provide an important resource for troubleshooting connectivity problems, pinpointing configuration areas that could use performance tuning (/guides/performance-tuning/), and identifying the causes of service disruptions. We recommend setting up remote log streaming when you start using Fastly (/quickstart/).

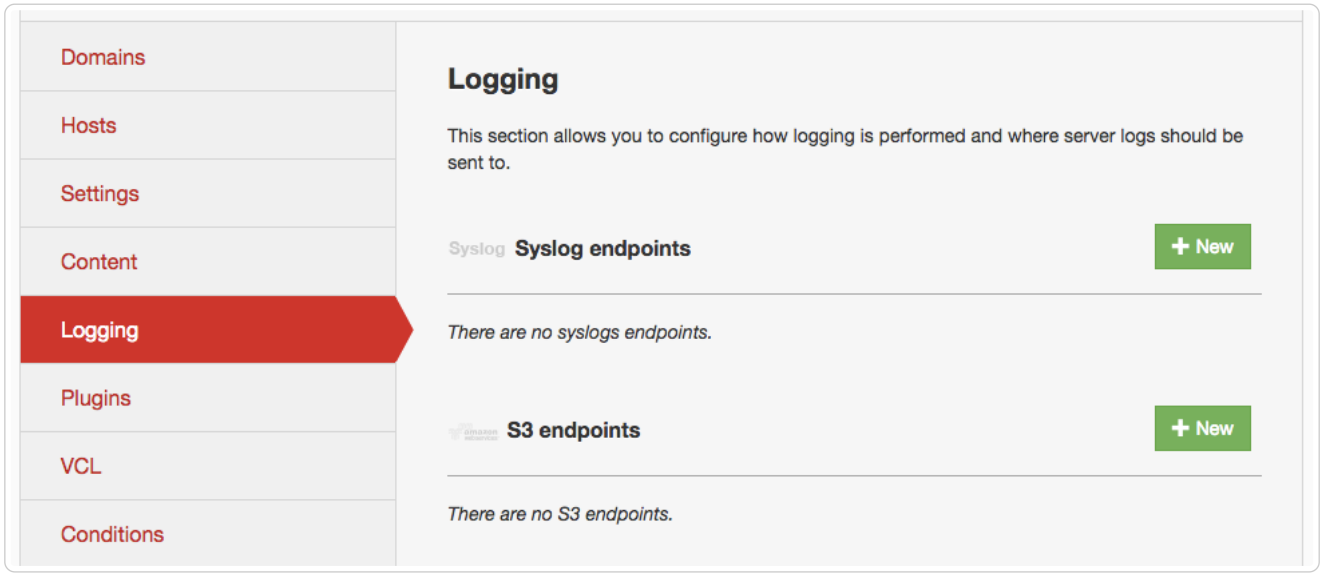
NOTE: Fastly does not provide direct support for third-party services. Please see Fastly's Terms of Service (https://www.fastly.com/terms) for more information.

Configuring logging endpoints

You can configure one or more logging endpoints in the Fastly application. Follow these instructions to access the logging settings:

1. Log in to the Fastly application.
2. Click the **configure** tab (the wrench icon).

3. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for your selected service appear.
4. From the section list on the left, click **Logging**. The list of third-party services appears.



> **NOTE:** Some third-party services are disabled by default, so you won't see them in the Fastly application user interface until they've specifically been enabled for your account. To enable these services, contact support@fastly.com (mailto:support@fastly.com).

5. Follow the instructions in one of our guides for third-party services to complete the set up process and deploy your changes:
   - Amazon S3 (/guides/streaming-logs/log-streaming-amazon-s3)
   - Cloud Files (/guides/streaming-logs/log-streaming-cloudfiles)
   - FTP (/guides/streaming-logs/log-streaming-ftp)
   - Google Cloud Storage (/guides/streaming-logs/log-streaming-google-cloud-storage)
   - Log Shuttle (/guides/streaming-logs/log-streaming-log-shuttle)
   - Logentries (/guides/streaming-logs/log-streaming-logentries)
   - OpenStack (/guides/streaming-logs/log-streaming-openstack)
   - Papertrail (/guides/streaming-logs/log-streaming-papertrail)
   - Sumo Logic (/guides/streaming-logs/log-streaming-sumologic)
   - Syslog (/guides/streaming-logs/log-streaming-syslog)

Once you've deployed your changes, events will begin being logged immediately. The logs may take a few moments to appear on your log server.

How, when, and where logs are streamed

To control log streaming, Fastly provides two versions of custom log formats (/guides/streaming-logs/custom-log-formats), each of which uses Apache-style logging directives (http://httpd.apache.org/docs/2.4/mod/mod_log_config.html). The logging format strings in each of these versions are based on the Common Log Format (https://httpd.apache.org/docs/trunk/logs.html#common) (CLF).

Logs are streamed over TCP, not UDP, optionally using TLS for security with supported endpoints. Additionally, if you are using custom VCL (/guides/vcl/uploading-custom-vcl) be sure to include the `#FASTLY deliver` (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl) macro in your `vcl_deliver` handler.

By default, logs are placed in your root directory every hour using the file naming format `YYYY-mm-ddThh:mm:ss-<server id>`. You can change both the frequency and path of these files. Fastly uses several different log-server aggregation points and each will send logs files, none of which contain duplicate entries. These log files are created as soon as streaming starts and they're written to over the entire time period you specify (or the default). Once that time has passed, the files aren't touched any more and the logging process creates a new batch of files.

Examples of other useful things you can log

Other useful things you can log include the following:

- `resp.http.X-Cache` - add HIT/MISS logging

- `req.http.LiterallyAnyHeaderYouWant` - any request header from the client
- `req.http.Accept-Language` - logs users' Accept Language

or any Varnish variable (https://www.varnish-cache.org/docs/2.1/reference/vcl.html#variables). Consider taking advantage of some of Fastly's extensions to VCL (/guides/vcl/guide-to-vcl#fastly's-vcl-extensions) as well.

Escaping characters in logs

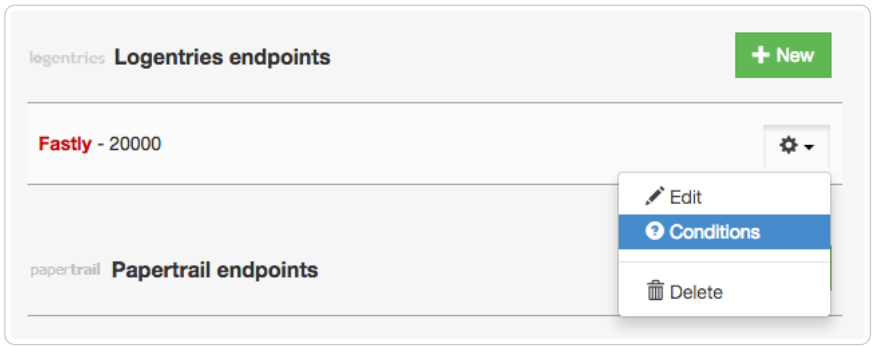
Logs respond to VCL (/guides/vcl/) like any other object. For example, the following code can escape quotes from User-Agent your log stream:

```
log {"syslog serviceid endpointname :: "} {"""} regsuball(req.http.user-agent, {"""}, {"\""}) {"""};
```

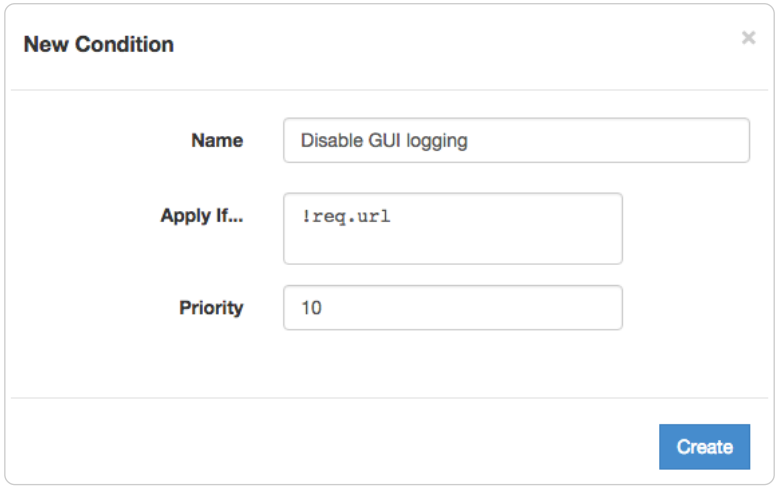
Preventing duplicate log entries when using custom VCL

If you use custom VCL (/guides/vcl/uploading-custom-vcl) commands for logging, you may notice duplicate entries in your logs. This happens because logs are being generated by both the Fastly application and the custom VCL logging commands. You can eliminate the duplicate entries by adding a condition that prevents the Fastly application from generating log entries. Follow these instructions to add the condition:

1. On the Logging page, click the gear icon next to the logging service you have enabled, and then select **Conditions**.



The New Condition window appears.



2. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a human-readable name for the condition.
   - In the **Apply If** field, type `!req.url`. That condition will never be met, and is what prevents the Fastly application from generating log entries.
   - Leave the default value set in the **Priority** field.
3. Click **Create**.

The Fastly application will stop generating log entries, and your logs will only contain entries generated by the custom VCL logging commands.

• Guides (/guides/) > Diagnostics and performance (/guides/diagnostics) > Debugging (/guides/debugging/)

§ Browser recommendations when using the Fastly application (/guides/debugging/browser-recommendations-when-using-the-fastly-application)

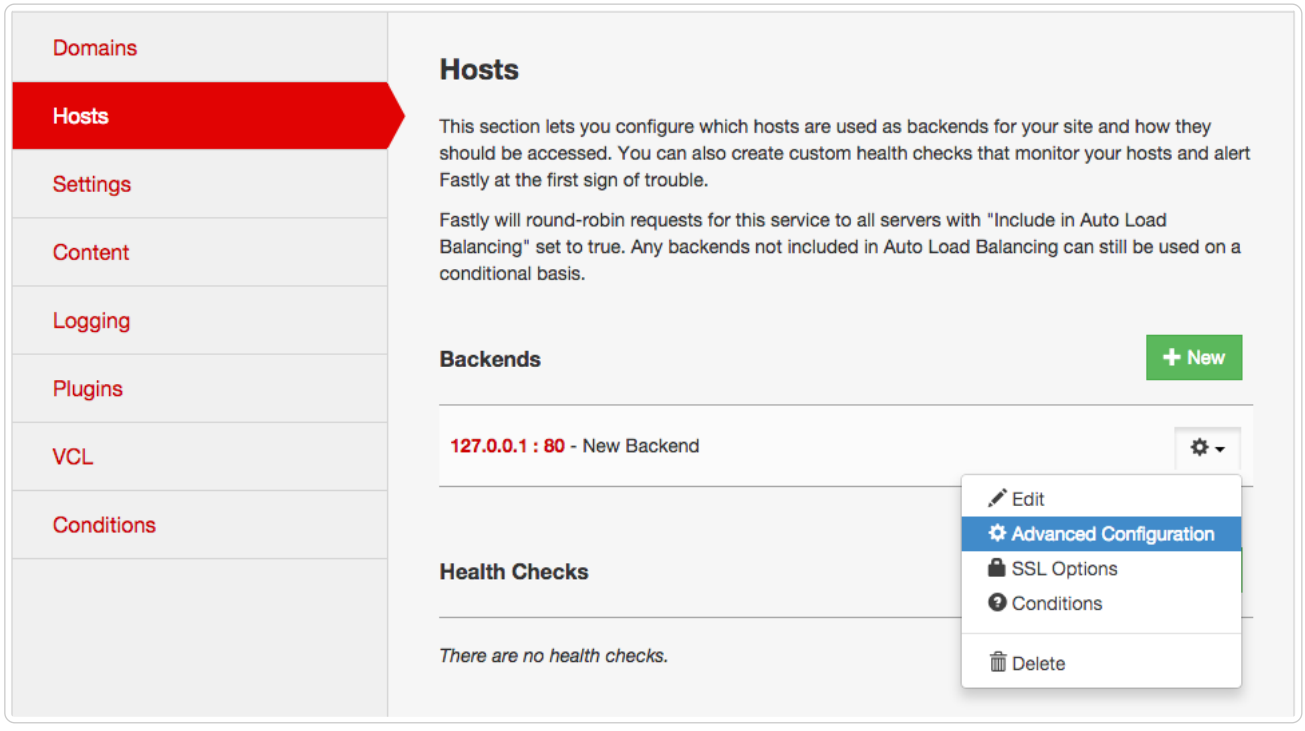
The Fastly application (https://app.fastly.com/) works well with all modern browsers. You can find the latest versions of all major browsers online. The list at Browse Happy (http://browsehappy.com/) may help you.

We recommend updating your browser before reporting bugs to Fastly Customer Support (mailto:support@fastly.com) and before beginning any debugging (/guides/debugging/) of Fastly services. Using the latest version of your favorite browser helps prevent malicious code from compromising your systems and ensures faster page loading. Old browser versions simply can't display support technologies that serve as the foundation for most modern websites and web applications.

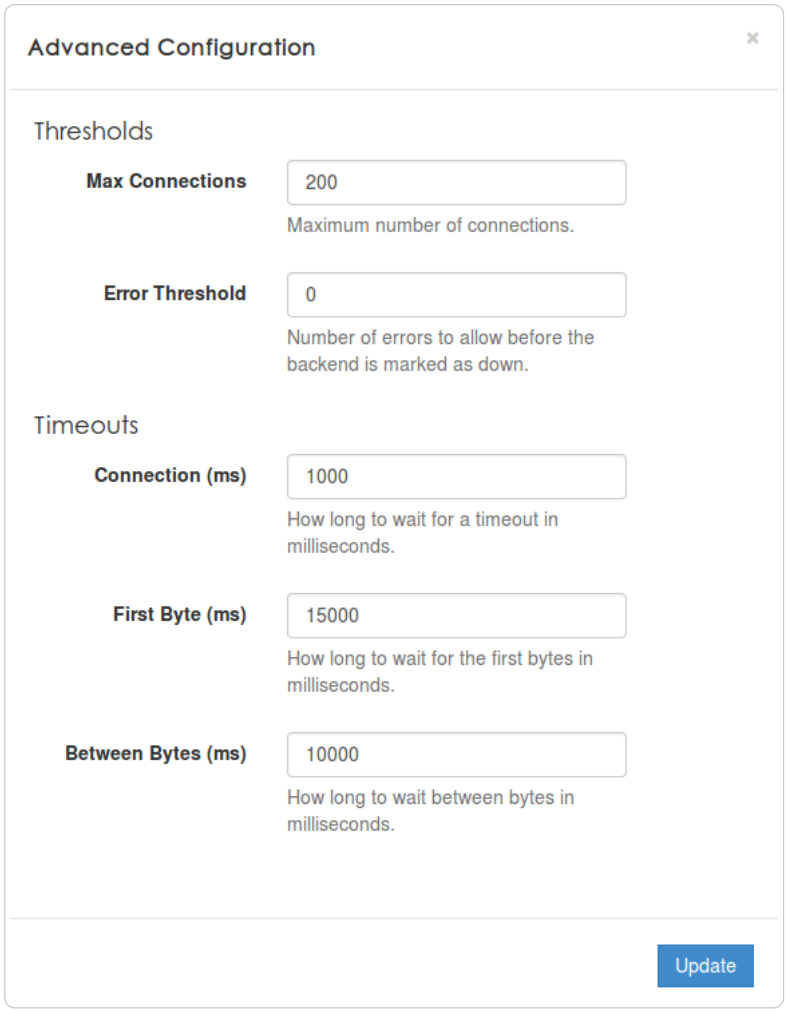
§ Changing connection timeouts to your origin (/guides/debugging/changing-connection-timeouts-to-your-origin)

Connection timeouts to your origin server control how long Fastly will wait for a response from your origin server before exiting with an error. Changing the connection timeout is a good way to start troubleshooting 503 backend read errors (/guides/debugging/common-503-errors#503-backend-read-error). Follow the steps below to change the connection timeouts to your origin server:

1. Log in to the Fastly application.
2. Click on the **configure** button (the wrench icon at the top of the application window).
3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the displayed service name.
5. Click the **Hosts** pane from the list on the left.
6. In the list of backends, click the gear icon to the right of the named backend you want to edit, then select **Advanced Configuration** from the menu.



The Advanced Configuration menu appears.



7. Type the new timeout in the appropriate field of the **Timeouts** section.
8. Click the **Update** button.

★ **TIP:** Additional techniques that help you gain insights into your service configurations can be found in our Debugging (/guides/debugging/) guides.

§ Common 503 errors (/guides/debugging/common-503-errors)

Varnish (/guides/vcl/guide-to-vcl), the software that runs on the Fastly CDN, will sometimes return standardized 503 responses due to various issues that can occur when attempting to fetch data from your origin servers. The generic status text associated with a 503 error is "Service Unavailable." It can mean a wide variety of things. The most common reasons this generic text appears include:

1. The origin server generated a 503 error and Fastly passed it through as is.
2. The origin returned a 503 error without a response header, so Fastly used the default response.
3. The status line of the HTTP response from the origin was not parseable.
4. VCL code was run that used the "error" statement without an appropriate response status (e.g., `error 503` instead of `error 503 "_broken thing_"`).

The following list provides the most common non-generic, standardized 503 responses and basic explanations for each.

⚠ **WARNING:** If you are seeing 503 errors, do not purge all (/guides/purging/single-purges#purging-all-content) cached content. Purge all overrides stale-if-error (/guides/performance-tuning/serving-stale-content) and increases the requests to your origin server, which could result in additional 503 errors.

503 Backend Read Error

This error typically appears if a timeout error occurs when Fastly cache servers attempt to fetch content from your origins. It can also be due to a variety of transient network issues that might cause a read to fail (e.g., router failovers, packet loss) or an origin overload.

Benchmarking your backend response times. Many outside factors cause backends response times to vary. Repeated, consistent Backend Read Errors frequently can be prevented by changing your backend timeout settings in the Fastly application. Start by running the following command to estimate response time for benchmarking purposes:

```
curl -s -w "%{time_total}\n" -o /dev/null http://example.com/path/to/file
```

Increasing your backend timeout settings. After benchmarking some of the slower paths in your application, you should have an idea of your ideal backend response time. Adjust the backend timeout values in the Hosts pane in the Advanced Configuration settings area. For more information, see our guide on changing connection timeouts to your origin (/guides/debugging/changing-connection-timeouts-to-your-origin).

503 Connection Timed Out

This error occurs if the request times out while waiting for Fastly to establish a TCP connection to your origin or waiting for your origin to respond to the request. Similar to Backend Read Errors, connection timeouts can be caused by transient network issues, long trips to origin, and origin latency. Two common ways to alleviate these timeout errors include increasing the connection timeout values set for the Fastly host (/guides/debugging/changing-connection-timeouts-to-your-origin) or setting up an origin shield (/guides/performance-tuning/shielding).

503 Backend Write Error

This error is similar to Backend Read Error but occurs when Fastly sends information in the form of a POST request to the backend.

503 Client Read Error

This error generally occurs because of a network issue between the client and Fastly. It is similar to the Backend Read Error but occurs when reading information from a client.

503 Connection Refused

This error occurs when Fastly attempts to make a connection to your origin over a specific port and the server refuses the connection. It typically appears when the wrong port is specified for the host in the Fastly application.

503 Network Unreachable

This error appears when Fastly cannot find a route to the given IP range. This generally occurs because of misconfigured or non-operational routers.

503 Backend Is Unhealthy

This error appears when custom health checks report a backend as down. It typically occurs when a Fastly edge server receives a client request and must make a request to your origin, but because the backend is considered unhealthy, Fastly doesn't try to send the request at all. This error may mistakenly appear instead of the Backend.max_conn Reached error the first time Fastly encounters the maximum number of connections to your backend.

503 Backend.max_conn Reached

This error occurs when Varnish makes a request to a backend in your Fastly service that has reached its defined maximum number of connections. By default, Fastly limits you to 200 origin connections from a single cache node to protect the origins from overload. To correct this error, increase the maximum connections limit to your origin. This error may also appear as "Maximum Threads for Service Reached."

503 Maximum Threads for Service Reached

See Backend.max_conn Reached.

503 Illegal Vary Header From Backend

This error occurs when a backend returns a malformed vary header with its response. A well-formed vary header (https://www.fastly.com/blog/best-practices-for-using-the-vary-header) tells Fastly to serve a different version of an object based on the value of the request header included within it.

503 No Stale Object Available

This error occurs when you configure Fastly to serve stale objects (/guides/performance-tuning/serving-stale-content) in the event of a backend failure but the stale object has expired and your backend is still failing for some reason (thus, no stale object is available).

503 Quorum Wait Not Reached

This error occurs when a Director (/api/config#director) used for balancing requests among a group of backends (only available via the Fastly API) cannot serve traffic based on its configuration because it does not have enough available backends in its group.

503 No Healthy Backends

This error occurs when a Director (/api/config#director) used for balancing requests among a group of backends (only available via the Fastly API) cannot cache the specified content because there are no healthy backends available in its group.

503 All Backends Failed or Unhealthy

This error occurs when a Director (/api/config#director) used for balancing requests among a group of backends (only available via the Fastly API) fails because all the backends are unhealthy or multiple backends from which the Director tried to fetch information failed with the same error.

503 SSL Handshake error

This error occurs when TLS negotiation between Fastly and your origin fails. To fix this error, review and correct your host's TLS configurations (/guides/securing-communications/connecting-to-origins-over-tls).

---

§ Curl and other caching verification methods (/guides/debugging/curl-and-other-caching-verification-methods)

The easiest way to tell if your request is caching in the Fastly network (https://www.fastly.com/network-map) is via the command line. The following command will display the request and response headers for a given object:

```
curl -svo /dev/null www.example.com/index.html
```

where `www.example.com/index.html` is replaced with the full object path of the object you're testing.

Examine the output looking for the `X-Cache` header. A properly caching object will display a value of `X-Cache: HIT` or `X-Cache: HIT, HIT`. For example, using `curl -svo /dev/null www.fastly.com` would produce something like the following:

```
* Rebuilt URL to: www.fastly.com/
* Hostname was NOT found in DNS cache
*   Trying 199.27.79.184...
* Connected to www.fastly.com (199.27.79.184) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.37.1
> Host: www.fastly.com
> Accept: */*
>
< HTTP/1.1 200 OK
* Server Apache is not blacklisted
< Server: Apache
< Last-Modified: Thu, 29 Jan 2015 22:53:28 GMT
< Timing-Allow-Origin: *
< Cache-Control: max-age=0
< Content-Type: text/html; charset=utf-8
< Via: 1.1 varnish
< Content-Length: 17493
< Accept-Ranges: bytes
< Date: Mon, 02 Feb 2015 22:55:01 GMT
< Via: 1.1 varnish
< Age: 85133
< Connection: keep-alive
< X-Served-By: cache-iad2120-IAD, cache-lax1433-LAX
< X-Cache: HIT, HIT
< X-Cache-Hits: 1, 920
< X-Timer: S1422917701.508078,VS0,VE0
< Vary: Accept-Encoding
<
{ [data not shown]
* Connection #0 to host www.fastly.com left intact
```

---

§ Debugging with mtr (/guides/debugging/debugging-with-mtr)

For diagnostics and debugging in the Fastly network, we think the mtr (http://www.bitwizard.nl/mtr/) tool offers a great way to test network speed, evaluate performance, and perform connection diagnostics. The program's source and installation instructions live in GitHub (https://github.com/traviscross/mtr).

While mtr provides a number of practical uses for network engineering needs, the following command works well:

```
mtr -c 20 -w -r www.example.com
```

Be sure to replace `www.example.com` with the hostname of the domain you're working with. The command will generate the network hops to the destination you specify, any packet loss experienced, and aggregate connection statistics.

For example, if we wanted to test the network connection from Fastly's San Francisco office to the CDN, we would use the above command for `www.fastly.com`. The following output would appear:

```
~  mtr -c 20 -w -r www.fastly.com
Start: Mon Feb  2 15:27:20 2015
HOST: test-local-machine.local                Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 10.100.20.2                            0.0%    20    2.1   2.2   1.6   4.3   0.5
  2.|-- ge-4-3-4.mpr4.sfo7.us.zip.zayo.com     0.0%    20    2.3   2.4   1.8   5.2   0.6
  3.|-- ae5.cr2.sjc2.us.zip.zayo.com           0.0%    20    4.6   6.5   2.9  35.3   7.7
  4.|-- ae10.mpr4.sjc7.us.zip.zayo.com         0.0%    20    4.7   4.8   3.6  14.5   2.3
  5.|-- be6461.ccr21.sjc03.atlas.cogentco.com  5.0%    20    5.1   5.9   4.2  15.3   2.6
  6.|-- fastly-inc.edge2.sanjose3.level3.net   0.0%    20    5.0   4.7   4.2   8.2   0.8
  7.|-- ???                                   100.0    20    0.0   0.0   0.0   0.0   0.0
  8.|-- 23.235.47.184                           0.0%    20    4.7  14.3   3.8  74.6  20.3
```

§ Debugging with WebPageTest (/guides/debugging/debugging-with-webpagetest)

It's important to establish habits of testing and performance before, during, and after migrating to Fastly. This allows you to clearly measure the impact of tests and changes to your infrastructure.

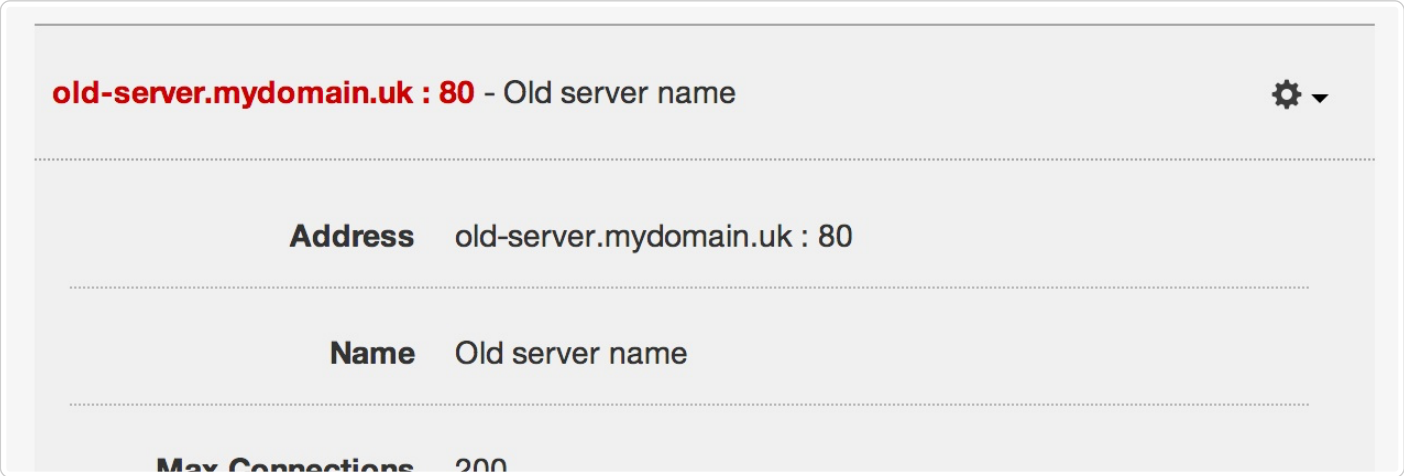
One tool that Fastly recommends for this purpose is WebPageTest.org (http://www.WebPageTest.org). WebPageTest provides a free and open source testing tool for deep performance analysis. It is built on browser technology to accurately replicate what your end users encounter when visiting a website.

We recommend using the WebPageTest defaults for basic testing, but keep a few rules in mind:

- On the Test Settings tab under Advanced Settings, Connection should always be set to Native Connection during initial benchmarks.
- Two to three test runs may be required before a site is properly caching in Fastly.
- Using WebPageTest's "Visual Comparison" (http://www.webpagetest.org/video/) feature offers an ideal way to A/B test potential changes.

§ Error 1000 with CloudFlare DNS (/guides/debugging/error-1000-with-cloudflare-dns)

Using CloudFlare for DNS and other CDNs can cause CloudFlare to show an Error 1000 indicating that your DNS points to prohibited IP addresses. This occurs when the hostnames are CNAMEed to Fastly (/guides/basic-setup/adding-cname-records) and an origin server is configured as a fully qualified domain name (FQDN) within Fastly:



To solve this error, direct Fastly to use the IP address as the host for any backend origin servers. This removes the need to resolve the hostname for traffic to the servers:

You can also change this by modifying the VCL configuration files directly. For example, this VCL:

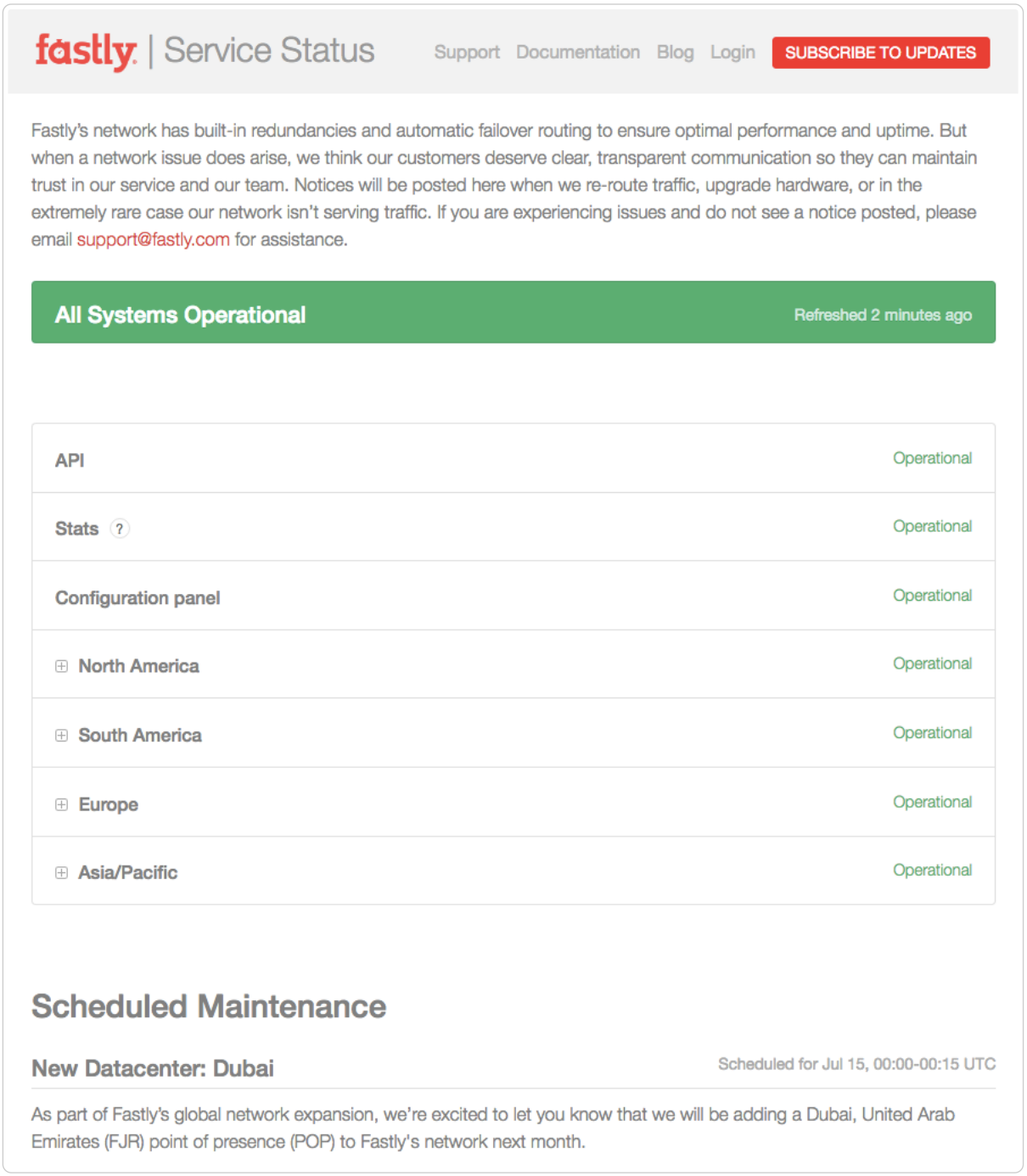
```
backend F_Hosting_server_Example_Backend {
  ...
  .port = "80";
  .host = "exampleserver.exampledomain.tld";
}
```

would become:

```
backend F_Hosting_server_Example_Backend {
  ...
  .port = "80";
  .host = "12.34.56.78";
}
```

§ Fastly's network status (/guides/debugging/fastlys-network-status)

Fastly continuously monitors the status of our global network and all related services. In the event of a service interruption, an update will be posted on the Fastly status page at status.fastly.com (https://status.fastly.com). If you are experiencing problems and do not see a notice posted, please email support@fastly.com (mailto:support@fastly.com) for assistance.

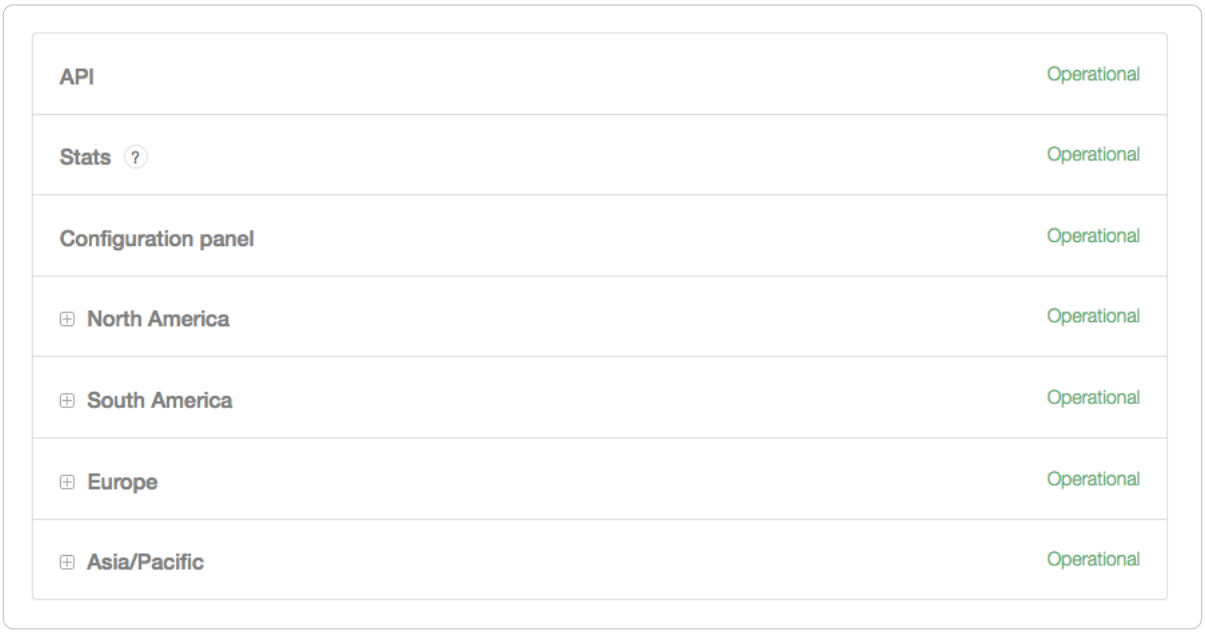


Overall system status

The current system status appears at the top of the Fastly status page and includes the last time the status was refreshed so that you know how current the information is.

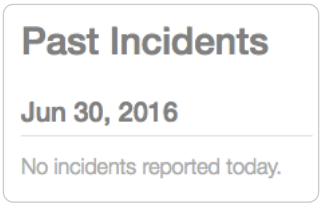
Individual component statuses

The status of the Fastly API (/api/), the Fastly application's configuration panel (https://app.fastly.com/), statistics collection and delivery, and each Fastly point of presence (https://www.fastly.com/network-map) (POP) appears immediately below the overall status. POPs are grouped by region. You can see the status of all POPs in a region by clicking the **+** icon next to the region's name.



Past incident statuses

Fastly keeps track of past incidents. Past incidents, if any, for approximately the past two weeks appear immediately below the individual component statuses.



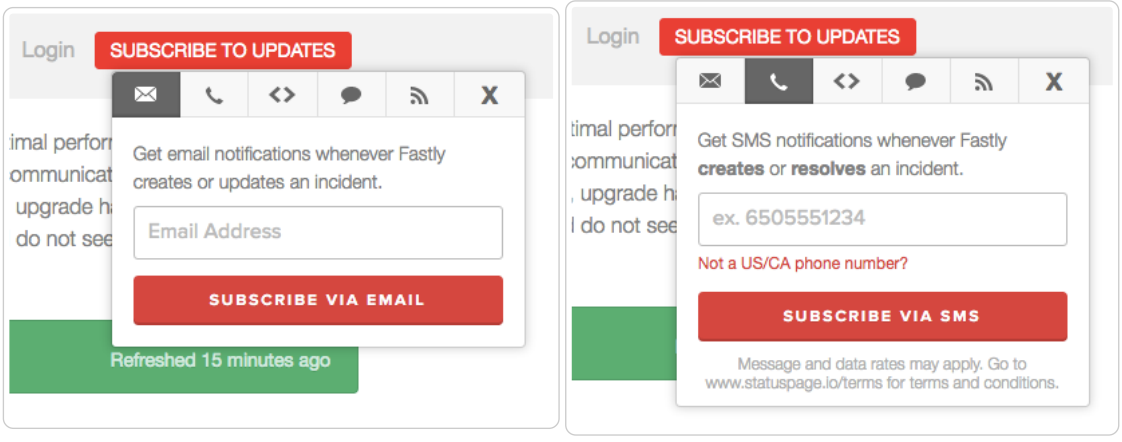
In addition to the textual description, each incident status appears in a color that indicates the level of service impact. The color indicators are as follows:

- Black - no component marked specifically as out of service or degraded
- Yellow - minor degradation or disruption
- Orange - significant degradation or traffic rerouting
- Red - component offline

We also keep track of all past incidents in an incident history page (https://status.fastly.com/history).

Subscribing to notifications

Fastly allows you to subscribe to status notifications via email or SMS text messaging. Simply click the **Subscribe to Updates** button in the upper right corner of the status page screen. Once subscribed, we'll email you any time we create or update an incident.



To subscribe to email notifications, click the letter icon, type your email address in the displayed field, and click **Subscribe Via Email**. You can unsubscribe at any time by clicking the unsubscribe link that appears at the bottom of every status email.

To subscribe via SMS text messaging, click the telephone icon, type your telephone number in the displayed field, and click **Subscribe Via SMS**. Unsubscribe from SMS text messaging at any time by replying STOP to any status message you receive.

§ Google Pagespeed module errors (/guides/debugging/google-pagespeed-module-errors)

If you are using the Google Pagespeed module and notice constant MISSes for HTML pages, check the Cache-Control settings in the module's .htcaccess file.

By default, Google Pagespeed serves all HTML with `Cache-Control: no-cache, max-age=0`. This setting conflicts with Fastly's default configuration. If your origin sends the headers `Cache-Control: private` or `Cache-Control: max-age=0`, Fastly will pass requests straight to the origin.

To change the Google Pagespeed directive and leave the original HTML caching headers, update your origin's .htcaccess file with:

`ModPagespeedModifyCachingHeaders off`

More details about the Pagespeed Module (https://developers.google.com/speed/pagespeed/module/install) can be found within Google Developers directory. For additional information about controlling how long Fastly caches your resources, start with our Cache Control Tutorial (/guides/tutorials/cache-control-tutorial)

§ Googlebot crawl stats (/guides/debugging/googlebot-crawl-stats)

Any time you notice any major changes in your SEO stats, indexing, or crawler behavior, start troubleshooting by asking these questions:

- Did you read the Google FAQs (https://sites.google.com/site/webmasterhelpforum/en/faq--crawling--indexing---ranking) for indexing, crawling, and ranking?
- Is your robots.txt file still accessible and were there any changes to it?
- Is your sitemap testing without errors (https://support.google.com/webmasters/answer/183669?hl=en&ref_topic=6080662&rd=2)?
- Did you adjust your Googlebot crawl rate (https://support.google.com/webmasters/answer/48620)?
- Have you had Google's "Fetch as Google" (https://support.google.com/webmasters/answer/6066468) tool re-crawl the URLs?

We recommend exploring Google's Webmaster Tools (https://www.google.com/webmasters/) if you're experiencing issues. Their "Fetch as Google" tool article and their article on troubleshooting sitmemap errors offer specific help for help debugging Googlebot crawl stats in this situation. Google also includes an entire section in their tools documentation on getting additional support (https://support.google.com/webmasters/answer/1249981?hl=en) if you're experiencing trouble.

★ **TIP:** Our debugging articles (/guides/debugging/) contain a variety of troubleshooting tips.

§ Loop detection (/guides/debugging/loop-detection)

Fastly automatically detects loops resulting from service configuration errors. When a loop is detected, Fastly blocks the requests and generates an error message. Loops can occur when the same hostname is configured as both the domain and the origin server, and the CNAME record for the domain is pointed at Fastly. For example, loop detection will be triggered if you set `www.example.com` as the domain and the origin server (/guides/basic-setup/sign-up-and-create-your-first-service#create-your-first-service-and-test-it) in your Fastly service and you add a CNAME DNS record (/guides/basic-setup/adding-cname-records) for `www.example.com` that points at Fastly.

How to avoid triggering loop detection

To avoid triggering loop detection, you should verify the hostname of your origin server is not the same as the domain using one of the following two options:

- Create a DNS hostname (`origin.example.com`) with the appropriate A and AAAA DNS records for your origin server, and use that origin DNS hostname in your Fastly service configuration. This ensures the origin (`origin.example.com`) is different than the domain (`www.example.com`) on your service. We recommend this option. If you make changes to the DNS records for `origin.example.com` in the future, Fastly will automatically detect and use those changes.
- Use an IPv4 address instead of a DNS hostname for your origin's address within your Fastly service's configuration. If the origin server's IP

address changes in the future, you'll need to update and activate a new version of your Fastly service configuration.

Example error message

When Fastly detects a loop, an error message similar to the one displayed below will appear in the headers.

```
HTTP/1.1 503 Loop detected
Error-Reason: loop detected
Connection: close
Content-Type: text/plain
Fastly-Host: <hostname>
Fastly-FF: <hostname>
Server: Varnish
```

§ Temporarily disabling caching (/guides/debugging/temporarily-disabling-caching)

Caching can be disabled:

- at the individual URL level,
- at the browser level, and
- at the site level.

Disabling caching at the individual URL level

To disable caching at the individual URL level:

1. Create a request setting that always forces a pass (/guides/basic-configuration/how-request-settings-are-applied).
2. Add a condition to the request setting (/guides/vcl/understanding-the-different-pass-action-behaviors) that looks for specific URLs.
3. Activate the new version of your service to enable the setting.

Disabling caching at the browser level

Theoretically, all browsers should follow the stated rules of the HTTP standard. In practice, however, some browsers don't strictly follow these rules. The following combination of headers seems to force absolutely no caching with every browser we've tested.

```
Cache-Control: no-cache, no-store, private, must-revalidate, max-age=0, max-stale=0, post-check=0, pre-check=0
Pragma: no-cache
Expires: 0
```

In addition, IE8 has some odd behavior to do with the back button. Adding `Vary: *` to the headers seems to fix the problem.

IMPORTANT: If you want your content cached in Fastly but not cached on the browser, you must not add these headers on your origin server. Instead, add these as new Headers via the Content pane of your control panel and be sure the Type is set to Response (/guides/basic-configuration/responses-tutorial).

Disabling caching at the site level

You can disable caching at the site level via the Fastly UI or via custom VCL. Via the UI, create a request setting that always forces a pass (/guides/basic-configuration/how-request-settings-are-applied) and then activate the new version of your service to enable the setting. To disable caching at the site level via custom VCL (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl), add this to your Fastly VCL:

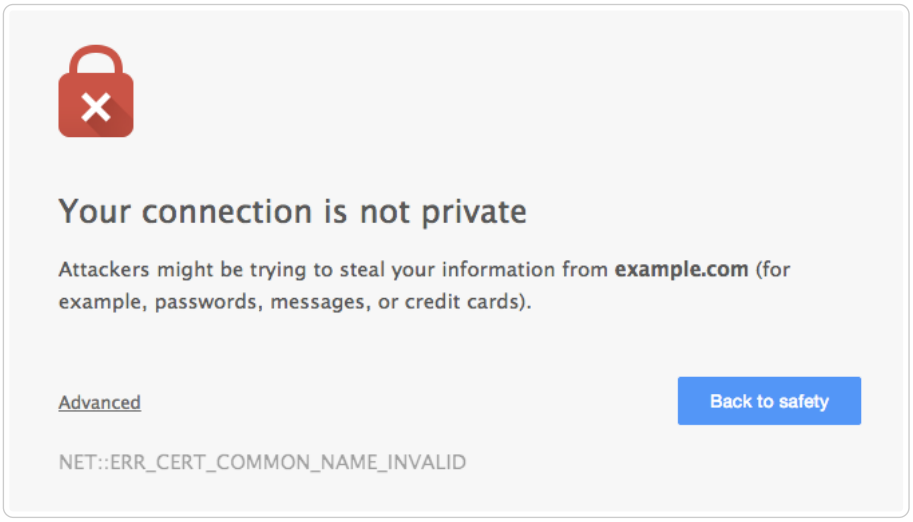
```
sub vcl_recv {
  return(pass);
}
```

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

§ TLS certificate errors (/guides/debugging/tls-certificate-errors)

"Your connection is not private"

If you've recently started testing Fastly services, you may see errors like the following:



These errors appear because your domain has not been provisioned with TLS across the Fastly network. We offer a number TLS options (/guides/securing-communications/ordering-a-paid-tls-option) that may work for you. Contact support@fastly.com (mailto:support@fastly.com) to begin the provisioning process.

If you don't want to use TLS for your site, set the CNAME DNS record for your domain to point to `global-nossl.fastly.net`. This network endpoint only accepts requests over port 80, and will not expose your users to these certificate errors.

Errors when using Wget

When connecting to a Fastly service using Wget, you may see errors along the lines of

```
ERROR: Certificate verification error for mysite.example.com: unable to get local issuer certificate
ERROR: certificate common name `*.a.ssl.fastly.net' doesn't match requested host name `mysite.example.com'.
To connect to mysite.example.com insecurely, use `--no-check-certificate'.
Unable to establish TLS connection.
```

Checking with a browser or curl will show that there really is no problem, however. The errors appear because a previous version of Wget (wget-1.12-2.fc13) that shipped with some versions of Red Hat Enterprise Linux (RHEL) was buggy and failed to check Subject Alternative Names (SAN) properly.

Upgrading Wget will correct this problem and eliminate the errors. For more information you can read this Red Hat bug report (https://bugzilla.redhat.com/show_bug.cgi?id=674186) or this Debian one (https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=409938). For more information about TLS-related issues, see our TLS guides (/guides/securing-communications/) or contact support@fastly.com (mailto:support@fastly.com) with questions.

---

§ TLS origin configuration messages (/guides/debugging/tls-origin-configuration-messages)

Hostname mismatches

- `Error: Hostname mismatch`

Why the error appears

Your origin server is serving a TLS certificate with a Common Name (CN) or list of Subject Alternate Names (SAN) that does not match the origin host or the origin's SSL hostname setting.

How to fix it

You can fix this by telling Fastly what to match against in the CN or SAN field in your origin's certificate.

1. Log in to the Fastly application.
2. Click the **configure** tab (the wrench at the top of the window).
3. From the **Service** menu, select the appropriate service.

4. Click the blue **Configure** button.
5. Click the **Hosts** pane from the list on the left.
6. Click the gear icon next to the affected host and select **TLS Options**. The TLS Options window appears.
7. In the **Certificate Hostname** field, type the hostname (either the CN or SAN), depending on which certificate was issued for the hostname. For example, if your certificate's CN field is `www.example.com`, type that value for your hostname.
8. Click **Update**.
9. Activate a new version of the service to complete the configuration changes.

When using custom VCL (/guides/vcl/uploading-custom-vcl), you can specify the hostname to match against the certificate by using the `.ssl_cert_hostname` field of your origin's definition. For example: `.ssl_cert_hostname = www.example.com;`.

Certificate chain mismatches

- `Error: unable to verify the first certificate`
- `Error: self signed certificate`
- `Error: unable to get local issuer certificate`
- `Error: self signed certificate in certificate chain`
- `Error: unable to get issuer certificate`

Why the errors appear

Your origin server is serving a certificate chain that can not be validated using any of the Certificate Authorities (CAs) that Fastly knows. This can happen for two reasons:

- Your certificate is self-signed or self-issued and you did not provide your generated CA certificate to Fastly for us to use for verification.
- Your certificate is issued by a CA that isn't in Fastly's CA certificates bundle.

How to fix them

In both cases, you can fix your configuration by adding the CA certificate that Fastly should use to verify the certificate to your service configuration:

1. Log in to the Fastly application.
2. Click the **configure** tab (the wrench at the top of the window).
3. From the **Service** menu, select the appropriate service.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Hosts** pane from the list on the left.
6. Click the gear icon next to the affected host and select **TLS Options**. The TLS Options window appears.
7. In the **TLS CA Certificate** field, copy and paste a PEM-formated CA certificate.
8. Click **Update**.
9. Activate a new version of the service to complete the configuration changes.

If you are using custom VCL, you can specify the CA for Fastly to use by setting the '.ssl_ca_cert' backend parameter to a PEM encoded CA certificate.

Alternatively, you can get a new certificate issued by a CA in Fastly's CA certificate bundle (e.g., Globalsign).

Connection failures

- `Error: Gethostbyname`
- `Error: Connection timeout`
- `Error: Connection refused`

Why each error appears and how to fix it

For `Gethostbyname` failures, the configured backend Host domain is returning NXDOMAIN. Double check that the DNS settings for your backend are correct.

For `Connection time out` failures, the connection to your server is timing out. Double check that your backend is accessible and responding in a timely fashion.

For `Connection refused` failures, the connection to your server is being refused, potentially by a firewall or network ACL. Double check that you have whitelisted the Fastly IP addresses (/guides/securing-communications/accessing-fastlys-ip-ranges) and that your backend is accessible from our network.

Certificate expirations

`Error: Certificate has expired`

The certificate your backend server is presenting Fastly has expired and needs to be reissued with an updated validity period.

If this is a self-signed certificate you can perform this update on your own by issuing a new CSR with your private key, creating the corresponding certificate, and installing it on the server.

If this is a CA signed certificate you will need to issue a new CSR with your private key, submit it to your CA, and install the signed certificate they provide you.

SSL and old TLS protocol errors

- `Error: Unknown protocol`
- `Error: SSL handshake failure`
- `Error: TLSv1 alert internal error`

Why the errors appear

Either your origin server is not configured to use TLS or it only supports older, outdated versions of the protocol. We do not support SSLv2 or SSLv3.

How to fix them

If the origin server is configured to use TLS, make sure you are using the latest version of both the server and the TLS library (e.g., OpenSSL). You may have to explicitly enable a newer protocol version. Fastly supports TLSv1, TLSv1.1 and TLSv1.2.

If the origin server is not configured to use TLS, change your service configuration to disable TLS and communicate with it on port 80 instead of port 443:

1. Log in to the Fastly application.
2. Click the **configure** tab (the wrench at the top of the window).
3. From the **Service** menu, select the appropriate service.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Hosts** pane from the list on the left.
6. Click the gear icon next to the affected host and select **TLS Options**. The TLS Options window appears.
7. From the **TLS for Connection** menu, select **No**.
8. Click **Update**.
9. Activate a new version of the service to complete the configuration changes.

RC4 cipher error

- `Error: Using RC4 Cipher`

Why the error appears

When Fastly connects to your origin server using TLS, the only cipher suite your server supports for establishing a connection is the RC4 cipher. This cipher is considered to be unsafe for general use and should be deprecated.

How to fix it

You can fix this on your origin by using the latest version of both the server and the TLS library (e.g., OpenSSL) and ensuring the cipher suites offered are tuned to best practices. You may need to explicitly blacklist the RC4 cipher.

---

§ Using GET instead of HEAD for command line caching tests (/guides/debugging/using-get-instead-of-head-for-command-line-caching-tests)

If you're testing on the command line to determine an object's caching status, then use GET instead of HEAD. For example:

```
curl -svo /dev/null www.example.com
```

By default, the results of GET requests are cached. HEAD requests are not proxied as is, but are handled locally if an object is in cache or a GET is done to the backend to get the object into the cache. Anything other than HEAD or GET requests are proxied and not cached by default. However, POSTs *can* be cached if required, and caching of that can be keyed off of the contents of the POST request body if that is below 2K. We've posted an example of how to do this (/guides/vcl/manipulating-the-cache-key#using-a-post-request-body-as-a-cache-key).

• Guides (/guides/) > Diagnostics and performance (/guides/diagnostics) > Performance tuning (/guides/performance-tuning/)

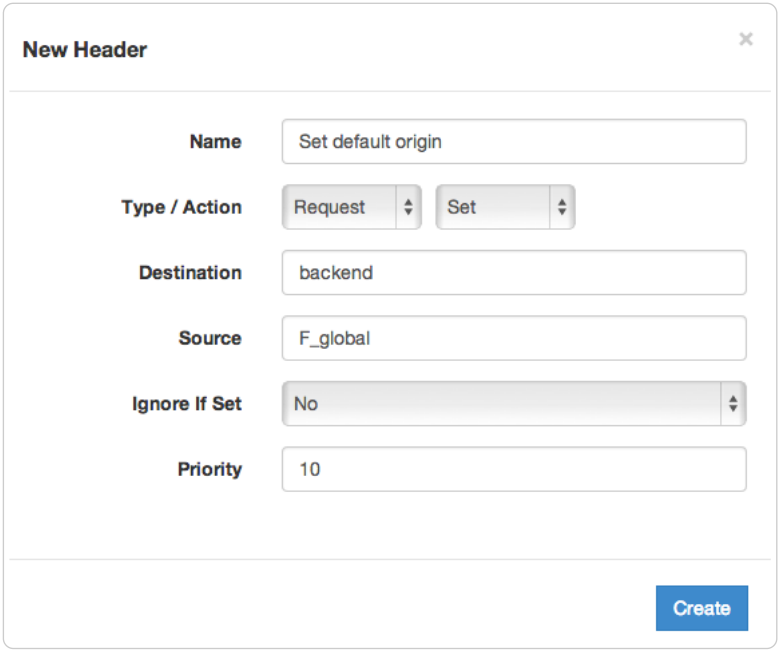
§ Changing origins based on user location (/guides/performance-tuning/changing-origins-based-on-user-location)

Fastly allows you to change origins based on user location. The VCL looks something like this:

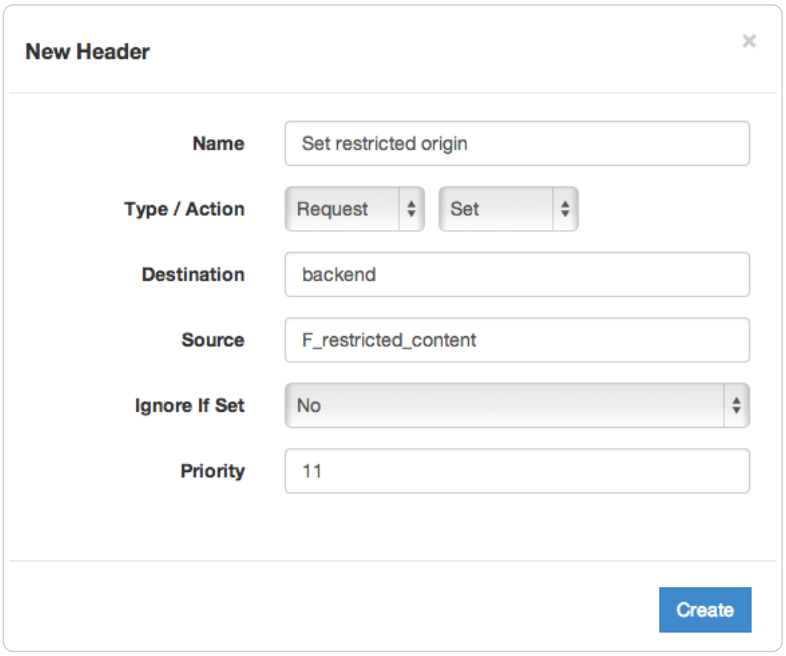
```
# default conditions
set req.backend = F_global;

# Use restricted content if the user is in Asia, France or Germany
if (geoip.continent_code == "AS" || geoip.country_code == "fr" || geoip.country_code == "de") {
  set req.backend = F_restricted_content;
}
```

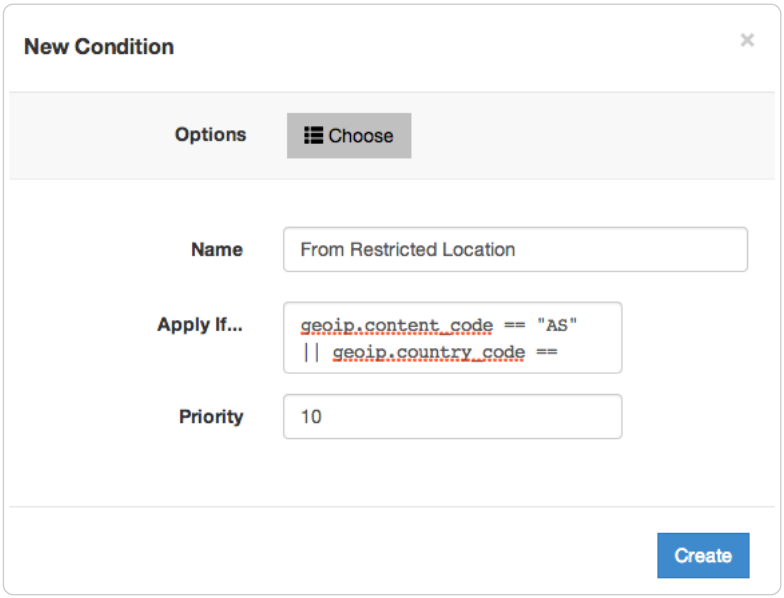
So, how does this translate to the Fastly UI? First, create a new Header (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses) under the Content pane of your configuration control panel (/guides/about-fastly-services/about-the-web-interface-controls#about-the-configuration-control-panel). The Type is Request and the Action is Set. Note that the priority is 10. The backend (here it's F_global should be the name of whatever your global origin server is. You can see the name if you look at your VCL).



Now create another header almost exactly the same with two differences: it sets the backend to be your restricted origin server and the priority is higher than the previous header.



Finally, attach a condition (/guides/conditions/) to the restricted origin header which checks the GeoIP header (/guides/vcl/geoip-related-vcl-features):

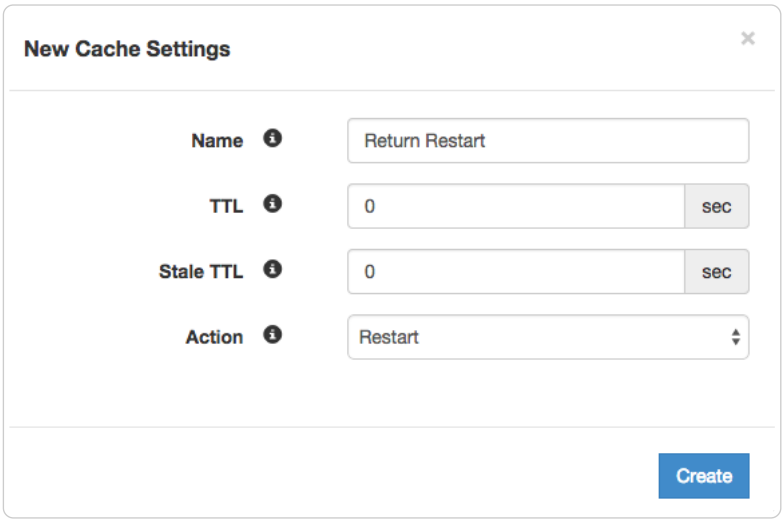


§ Checking multiple backends for a single request (/guides/performance-tuning/checking-multiple-backends-for-a-single-request)

Using a restart is a good option to check multiple backends for a single request. This can be created using a cache setting rule (/guides/tutorials/cache-control-tutorial) and request headers.

Create a new cache setting rule

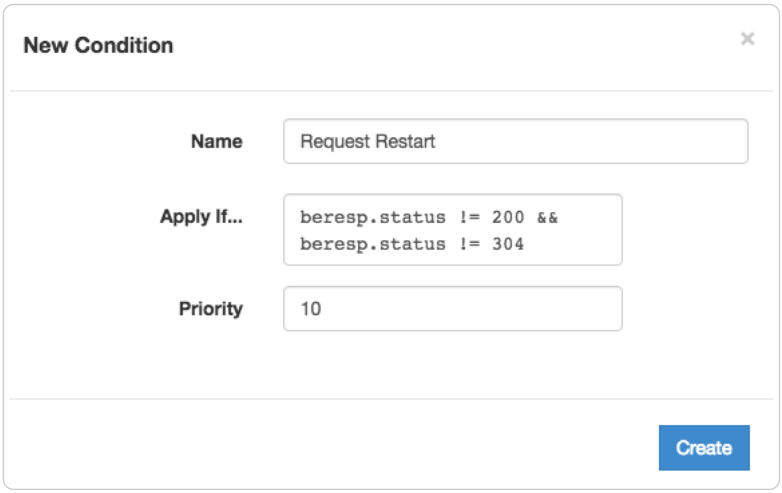
Within the Fastly application, navigate to **Settings** → **Cache Settings**. Then select the **New** button to create a cache restart within vcl_fetch.



Set the cache settings as follows:

- **Name:** `Return Restart` (or any meaningful, preferred name)
- **TTL:** `0`
- **Stale TTL:** `0`
- **Action:** `Restart`

Click **Create** to add the new cache setting (it will appear in the Cache Settings area). Then click the gear icon next to this new cache setting to add a new condition.



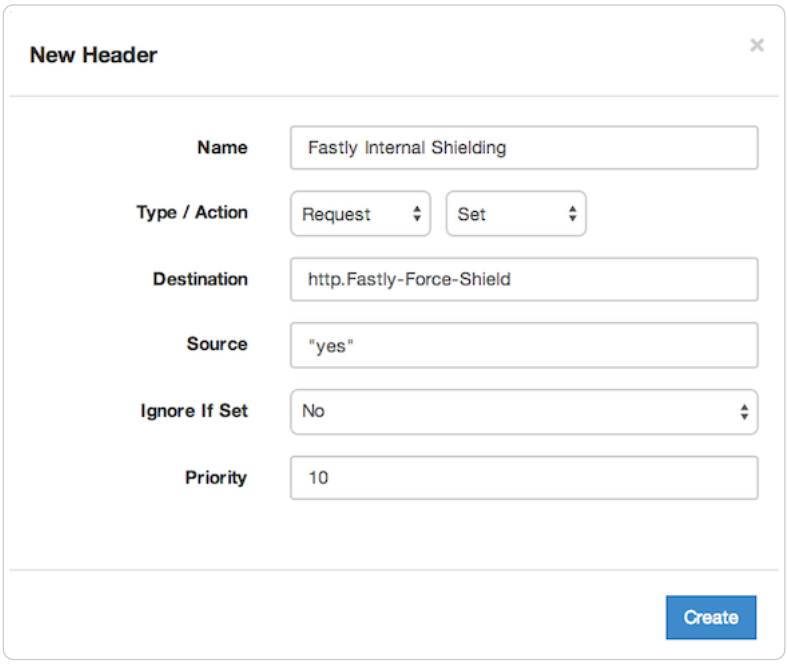
Set the new condition settings as follows:

- **Name:** `Restart Request` (or any meaningful, preferred name)
- **Apply if**: `beresp.status != 200 && beresp.status != 304`

Click **Create** to assign the condition.

Create new request headers

Within the Fastly application, navigate to **Content** → **Headers**. Then select the **New** button to create a request header within vcl_recv.

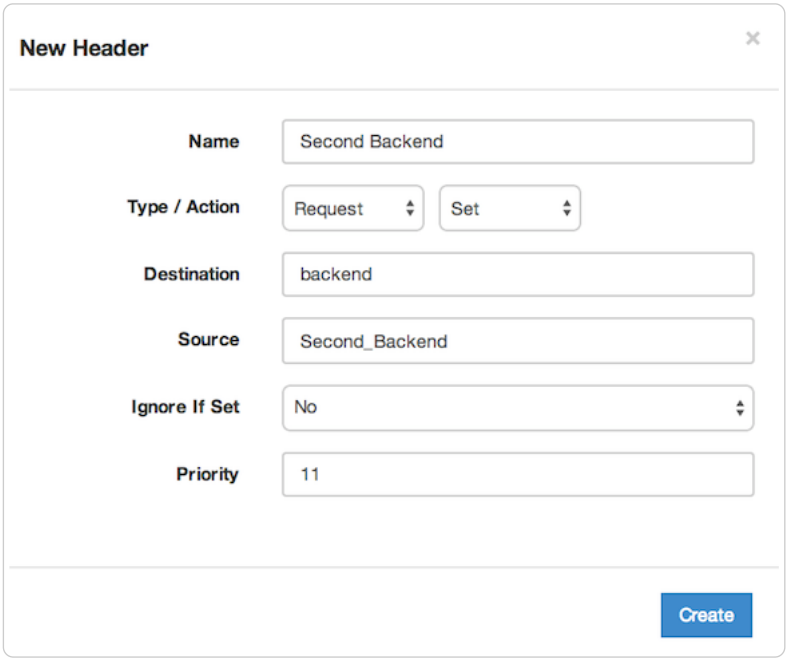


Set the header controls as follows:

- **Name:** `Fastly Internal Shielding` (or any meaningful, preferred name)
- **Type/Action:** `Request/Set`
- **Destination:** `http.Fastly-Force-Shield`
- **Source:** `"yes"`
- **Ignore If Set:** `No`
- **Priority:** `10`

Click **Create** to add the new header (it will appear in the Headers area).

Add another header to switch to the next backend.



Set the next header controls as follows:

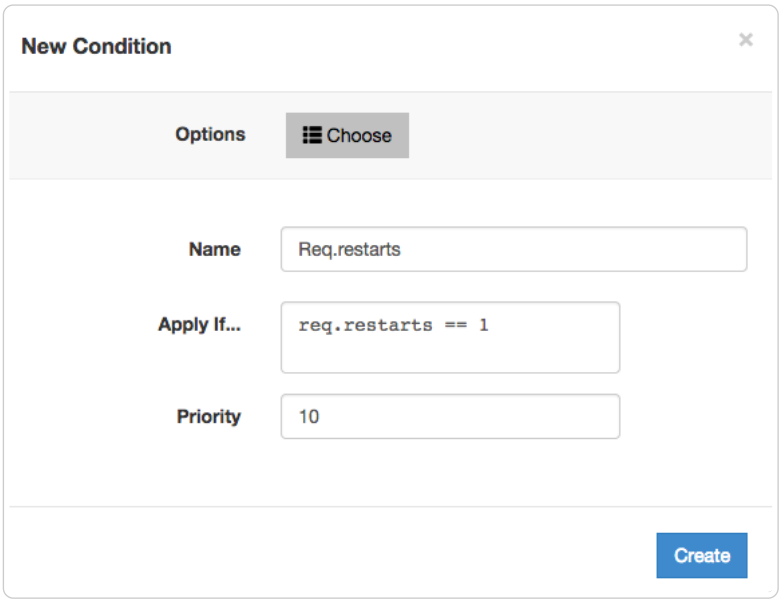
- **Name:** `Second Backend` (or any meaningful, preferred name)
- **Type/Action:** `Request/Set`
- **Destination:** `backend`
- **Source:** `Second Backend` (this should match the name of your other backend)

- **Ignore If Set:** `No`
- **Priority:** `11` (the priority is higher to set after the Fastly Internal Shielding)

Click **Create** to add the new header (it will appear in the Headers area).

Create new header conditions

Once you've created the new headers, click the gear icon next to each new header rule in the Headers area to add a new condition.



Set the new condition settings as follows:

- **Name:** `Req.request` (or any meaningful, preferred name)
- **Apply if**: `req.restarts == 1`

Click **Create** to assign the condition. Then click the **Activate** button, near the top right corner of the window, to deploy the new version of the service you just edited.

---

§ Controlling caching (/guides/performance-tuning/controlling-caching)

How long Fastly caches content

The maximum amount of time we cache content depends on a number of factors including the TTL (Time To Live) and Grace Period, how often an object gets accessed, and how busy other customers are. Setting TTL and Grace Period to a week, possibly even two weeks should be absolutely fine. For more information about controlling how long Fastly caches your resources, start with our Cache Control Tutorial (/guides/tutorials/cache-control-tutorial).

In general, we will honor any cache-control headers (/guides/about-fastly-services/how-fastlys-cdn-service-works) you send to us from your origin; however, if you do not send any cache-control headers, you will most likely reach the Default TTL limit for your service. You can change this limit in the Configuration Control Panel (/guides/about-fastly-services/about-the-web-interface-controls#about-the-configuration-control-panel) under the Settings pane.

Changing caching times for different users

You can change the caching times for different users through Surrogate-Control headers defined by the W3C (http://www.w3.org/TR/edge-arch/). If, for example, you wanted Fastly to cache something for a month (clearing with API purges, if necessary) but you also wanted to set a maximum age of a single day for users viewing that object in a browser, then you could return the HTTP header:

```
Surrogate-Control: max-age=2629744
Cache-Control: max-age=86400
```

The Surrogate Control header in this example tells Fastly to cache the object for a maximum of 2629744 seconds (one month). The Cache Control header in this example tells the browser to cache the object for a maximum of 86400 seconds (1 day).

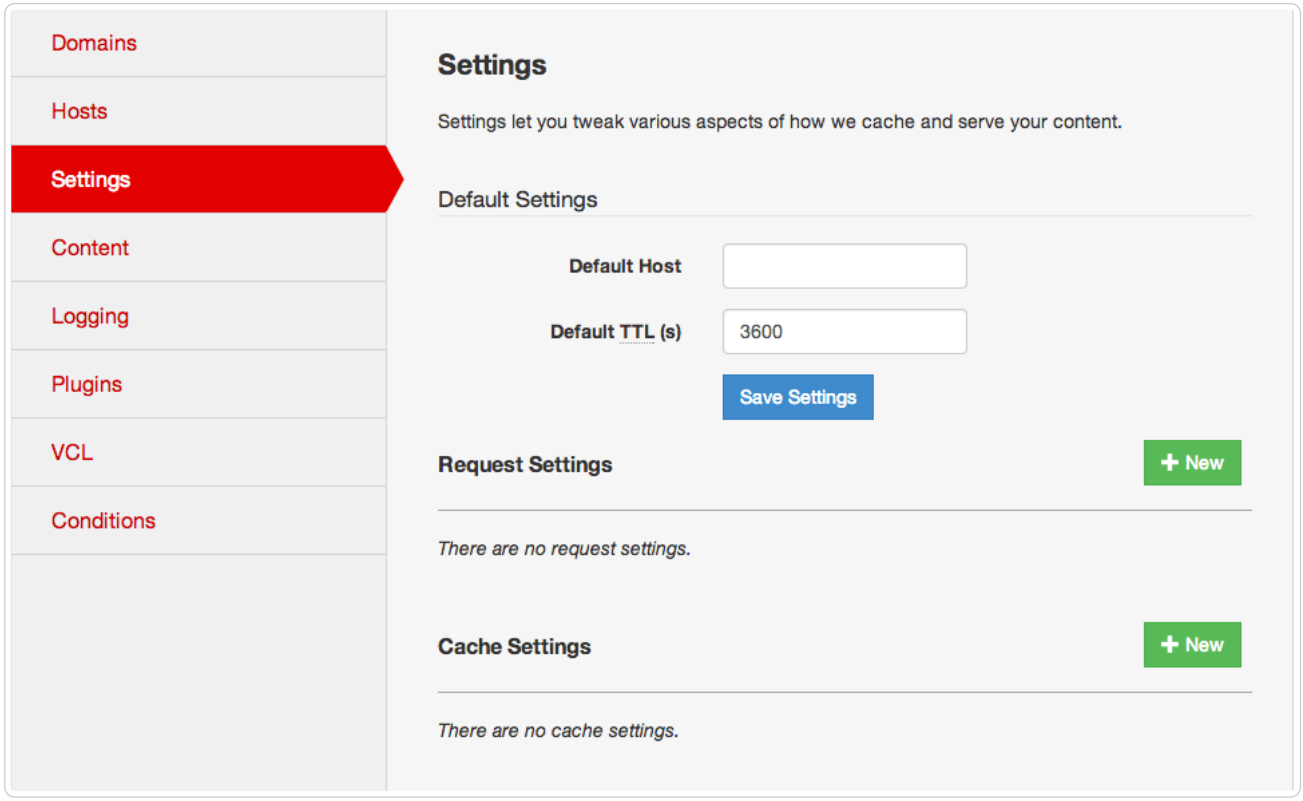
For more information about controlling caching, see our Cache Control Tutorial (/guides/tutorials/cache-control-tutorial).

Conditionally preventing pages from caching

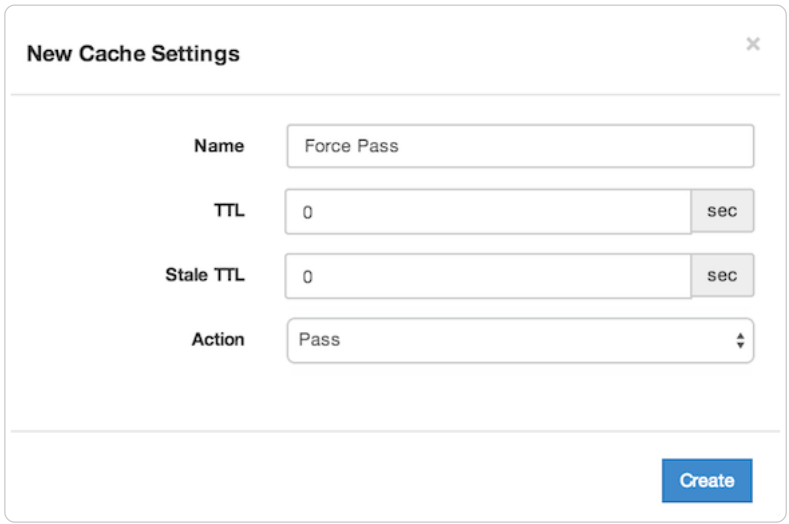
To conditionally prevent pages from caching, follow the steps below.

1. Log in to the Fastly application.
2. Click on the **configure** button (the wrench icon at the top of the application window).

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the displayed service name.
5. Click **Settings** from the section list on the left.

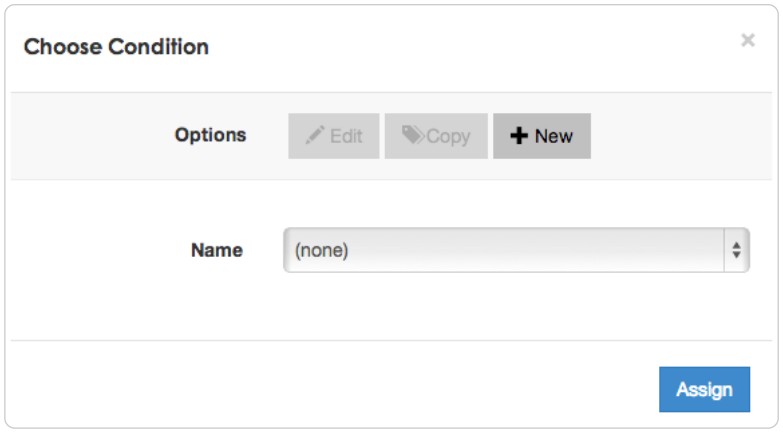


6. In the **Cache Settings** area, click the green **New** button to create a new cache setting. The New Cache Settings window appears.

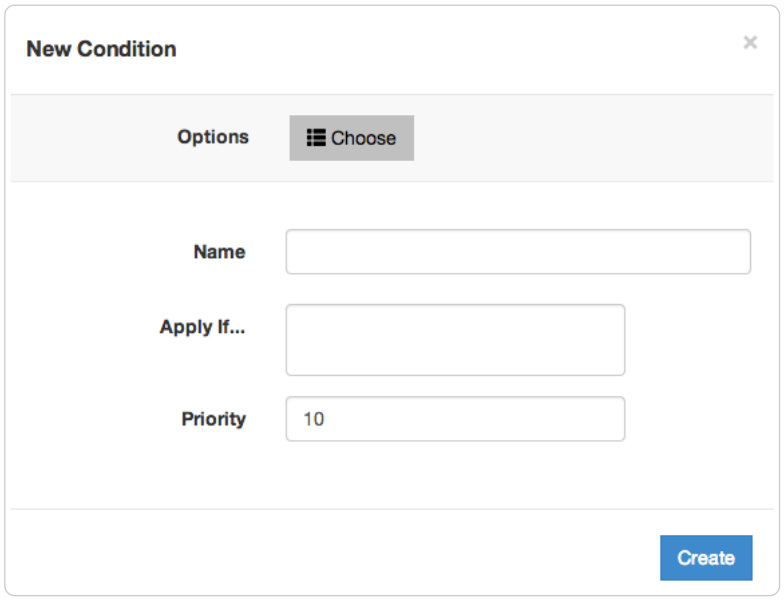


7. Create a new cache setting and click then click the **Create** button. For more information about the actions, see our descriptions of caching actions. The new cache setting you created appears in the Cache Settings area of the Settings pane on the Configuration tab.
8. Click the gear icon to the right of the newly created cache setting and then select **Conditions** from the menu.

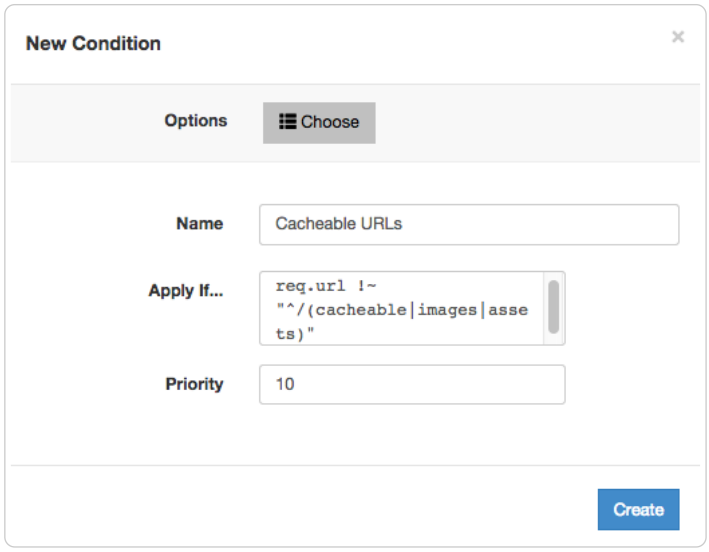
The Choose Condition window appears.



9. Click the **New** button on the **Choose Condition** window. The New Condition window appears.



10. Create a condition that matches against the URLs you want and then click the **Create** button. In this example, we set the condition to look for URLs containing '/cacheable', '/images', or '/assets'. If it finds them, the URLs should be cached. If it doesn't find them, the URLs are explicitly not cached by the Apply If statement shown below.



★ **TIP:** You can use these steps to override default caching based on a backend response (/guides/basic-configuration/overriding-caching-defaults-based-on-backend-responses).

Caching action descriptions

You can use actions to tell Fastly what to do with cached objects and what to do with additional cache configurations as a result. The following actions are available:

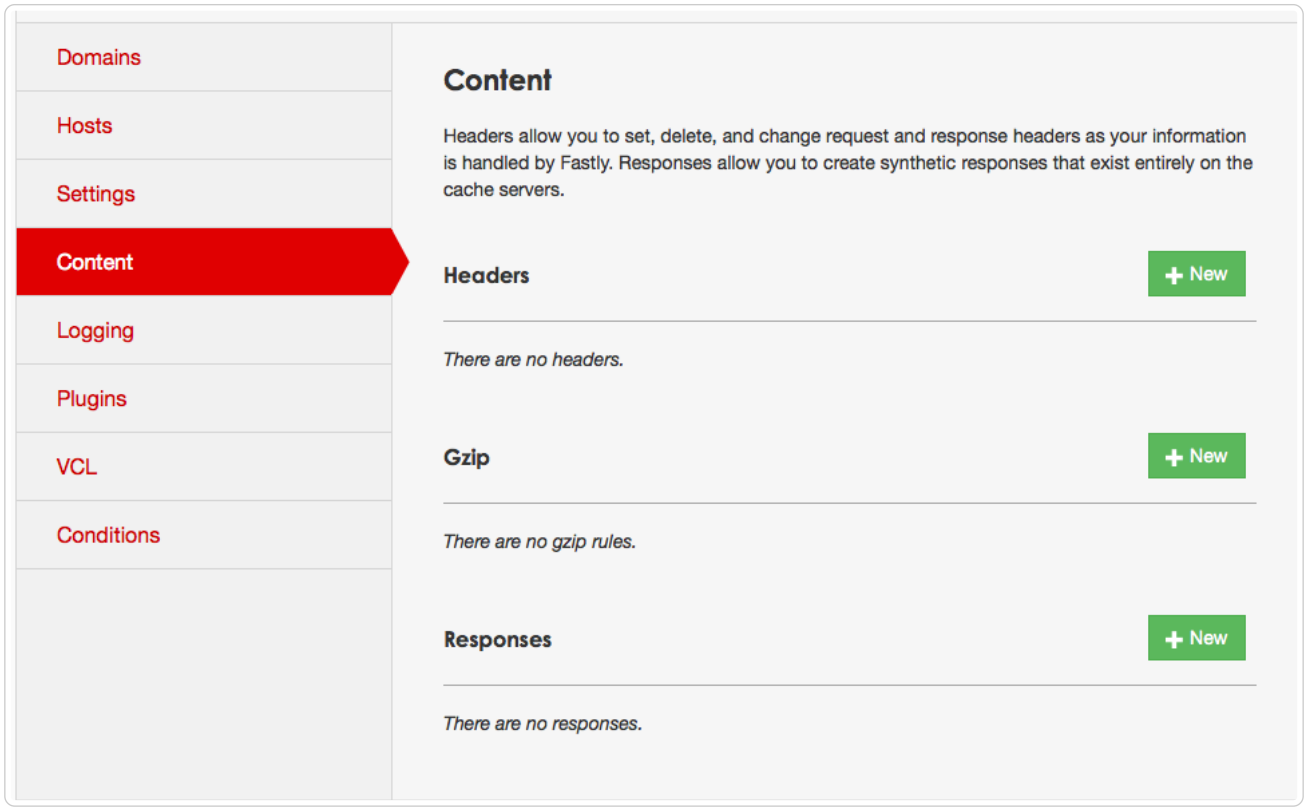
- **N/A** - Applies the cache settings within the policy and continues through to other cache configurations.
- **Deliver** - Delivers the object to the client. Usually returned from `vcl_fetch`.
- **Pass** - Passes the request and subsequent response to and from the origin server. It won't be cached. Pass can be returned from `vcl_recv`.
- **Restart** - Restarts processing of the request. You can restart the processing of the whole transaction. Changes to the `req` object are retained.

§ Enabling cross-origin resource sharing (CORS) (/guides/performance-tuning/enabling-cross-origin-resource-sharing)

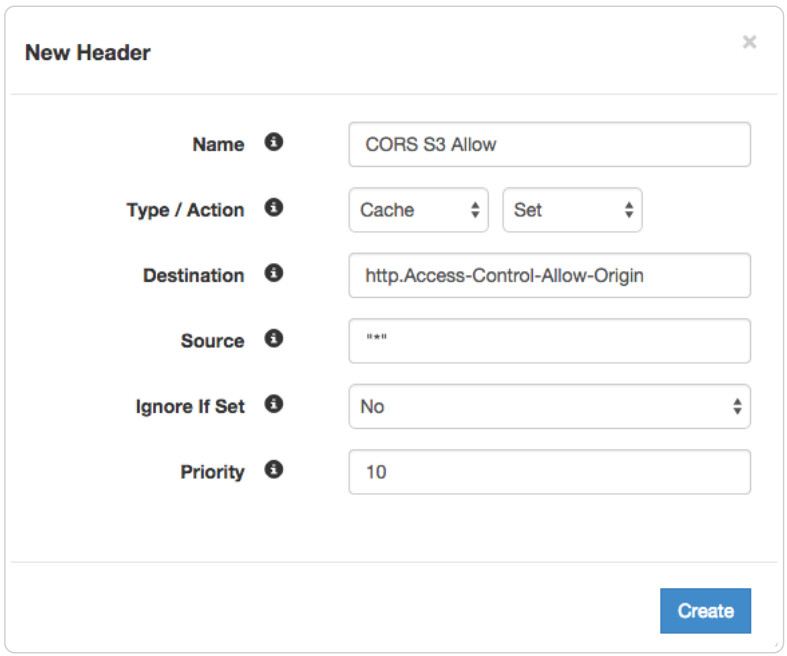
We recommend enabling CORS (Cross-Origin Resource Sharing (http://docs.aws.amazon.com/AmazonS3/latest/dev/cors.html)) when using Amazon S3 (/guides/integrations/amazon-s3) as your backend server. To enable CORS, set up a custom HTTP header for your service by following the steps below.

1. Log in to the Fastly application.
2. Click the **configure** tab (the wrench icon).

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button.
5. Click the **Content** pane. The Content controls appear.



6. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.



7. Fill out the **New Header** fields as follows:
   - In the **Name** field, type a descriptive name for the new header (e.g., `CORS S3 Allow`).
   - From the **Type/Action** menus, select **Cache** and **Set**.
   - In the **Destination** field, type `http.Access-Control-Allow-Origin`.
   - In the **Source** field, type `"*"`.
   - Leave the **Ignore If Set** menu and the **Priority** field set to their default values.
8. Click **Create**. The new header appears in the Headers area.

IMPORTANT: Objects already cached won't have this header applied until you purge them (/guides/purging/).

Test it out

Running the command `curl -I your-hostname.com/path/to/resource` should include similar information to the following in your header:

```
Access-Control-Allow-Origin: http://your-hostname.tld
Access-Control-Allow-Methods: GET
Access-Control-Expose-Headers: Content-Length, Connection, Date...
```

§ Enabling global POPs (/guides/performance-tuning/enabling-global-pops)

The sun never sets on the Fastly empire, but how can you take full advantage? Simply set your CNAME record (/guides/basic-setup/adding-cname-records) to `global-nossl.fastly.net`. You'll now have access to all of our worldwide POPs (https://www.fastly.com/network-map) as they come online. We don't restrict POP access. Instead, you control it.

How to check if your CNAME is set to global.prod.fastly.net

Run the following command in your terminal:

```
$ host www.example.com
```

Your output should appear similar to the following:

```
www.example.com is an alias for global-nossl.fastly.net.
global-nossl.fastly.net has address 23.235.46.204
```

If you don't see `global-nossl.fastly.net` in your output, then your CNAME isn't properly set. We link to instructions for setting your CNAME (/guides/basic-setup/adding-cname-records) for a number of popular providers.

Instead of using the above command in your terminal, you can also use various online DNS checking tools, such as the OpenDNS Cache Check (https://cachecheck.opendns.com/).

Limiting POP use to the United States and European Union

If you're not using TLS, simply set your CNAME record (/guides/basic-setup/adding-cname-records) to `a.prod.fastly.net` instead of `global-nossl.fastly.net`. If you're using TLS, see our guide on CNAME records (/guides/basic-setup/adding-cname-records) to find the appropriate entry.

§ Failover configuration (/guides/performance-tuning/failover-configuration)

This guide describes how to configure a failover origin server. A failover (backup) server ensures you can maintain availability of your content if your primary server is not available.

Before you begin

Before you configure failover, keep in mind the following:

- To configure a failover origin server you must make sure you have health checks (/guides/basic-configuration/health-checks-tutorial) configured for your primary server. If you configure your failover server but don't configure health checks (/guides/basic-configuration/health-checks-tutorial) on the primary server, the failover won't work properly if your primary server stops responding.
- Many customers configure load balancing at the same time they configure failover functionality. Our guide on configuring load balancing (/guides/performance-tuning/load-balancing-configuration) can show you how.

Configuring a failover origin server

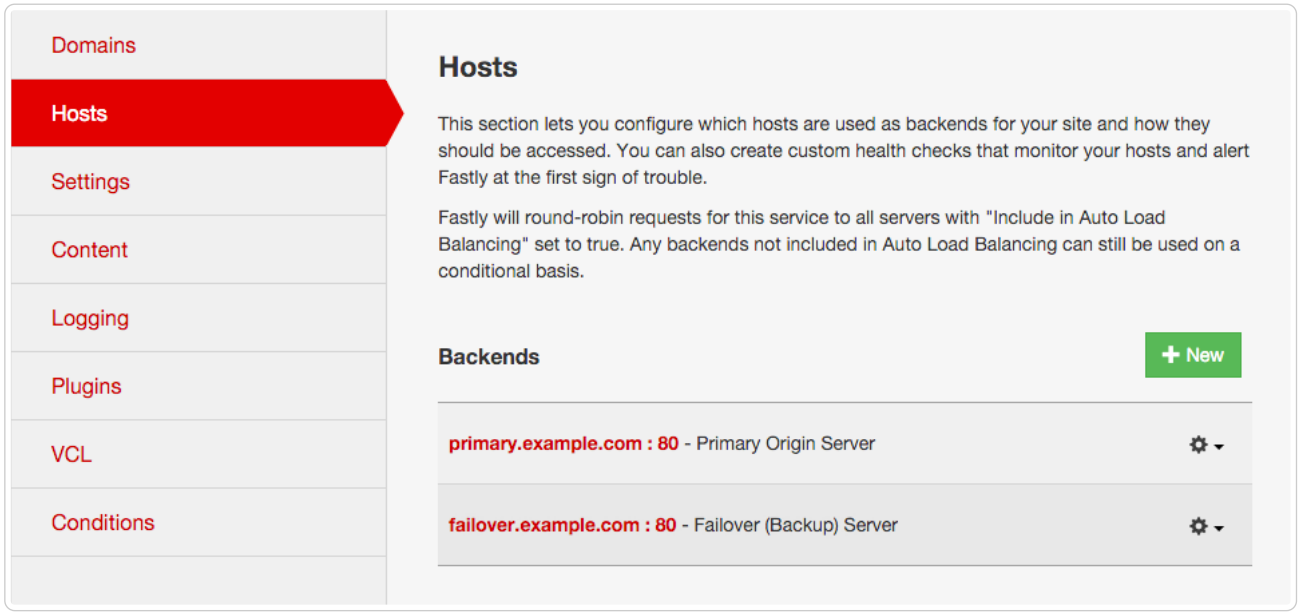
Once you've confirmed health checks are configured, you must:

1. Turn off automatic load balancing on both the primary origin server and the server that will become your failover.
2. Create headers that configure both the primary and failover origin servers.
3. Create a header condition that specifies exactly when to use the failover server.

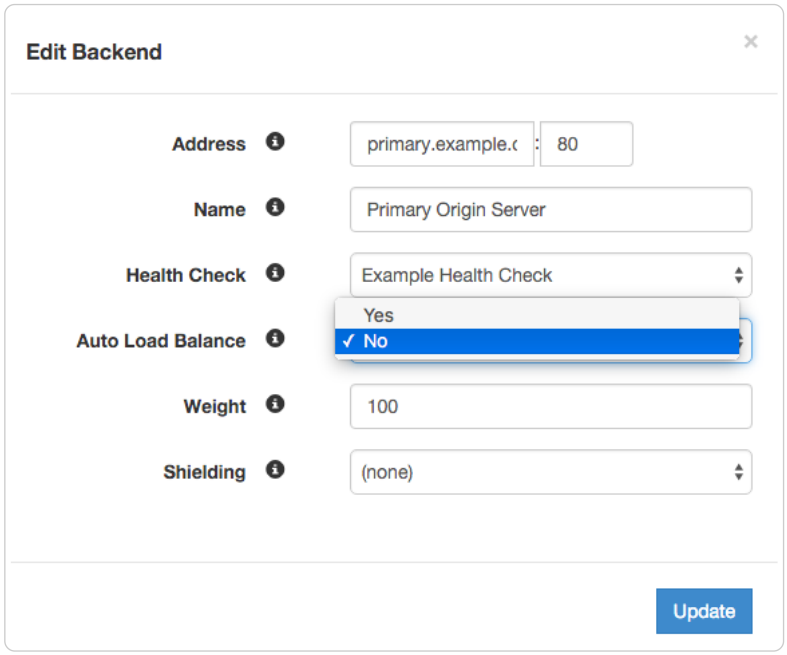
Turn off automatic load balancing

To configure a failover origin server you must turn off automatic load balancing for both the server that will act as your primary origin server and the server that will become your failover origin server.

1. Log in to the Fastly application.
2. Click the **configure** tab (the wrench icon).
3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button.
5. Click the **Hosts** pane from the list on the left.



6. In the **Backends** area, click the gear icon next to the name of the origin server you want to configure.
7. Click **Edit**. The Edit Backend window appears.



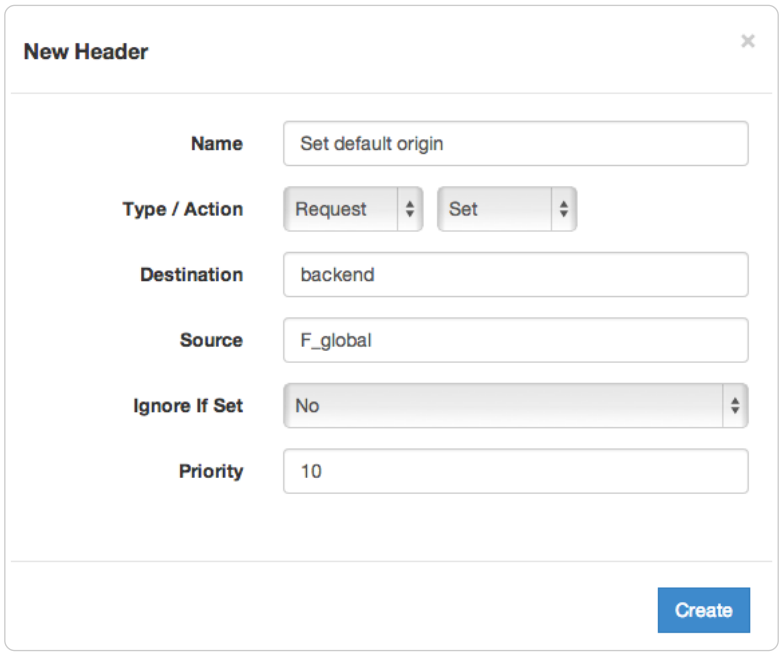
8. From the **Auto Load Balance** menu, select **No**.
9. Click **Update** to apply the changes.

Configure the primary and failover origin servers

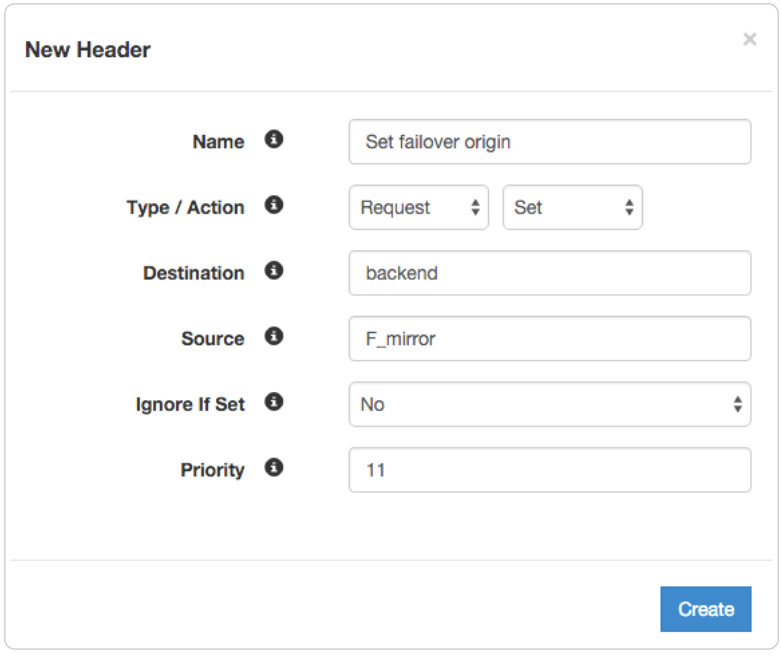
Once you've turned off automatic load balancing, create two new request headers, one each for your primary and failover servers.

1. Click the **Content** pane from the list on the left.

2. In the **Headers** area, click the **New** button to create the first request header. The New Header window appears.



3. Fill out the **New Header** window as follows:
   - In the **Name** field, type a descriptive name for the header.
   - From the **Type/Action** menus, select **Request** and **Set**.
   - In the **Destination** field, type the name of the header that will be affected by the selected action.
   - In the **Source** field, type where the new content for the header comes from.
   - Leave the **Ignore If Set** and **Priority** controls at their default settings.
4. Click **Create** to create the first header. A new header appears in the Headers area that sets the first server as the primary origin server.
5. In the **Headers** area, click the **New** button to create a second request header. The New Header window appears.



6. Fill out the **New Header** window as follows:
   - In the **Name** field, type a descriptive name for the header.
   - From the **Type/Action** menus, select **Request** and **Set**.
   - In the **Destination** field, type the name of the header that will be affected by the selected action.
   - In the **Source** field, type where the new content for the header comes from.

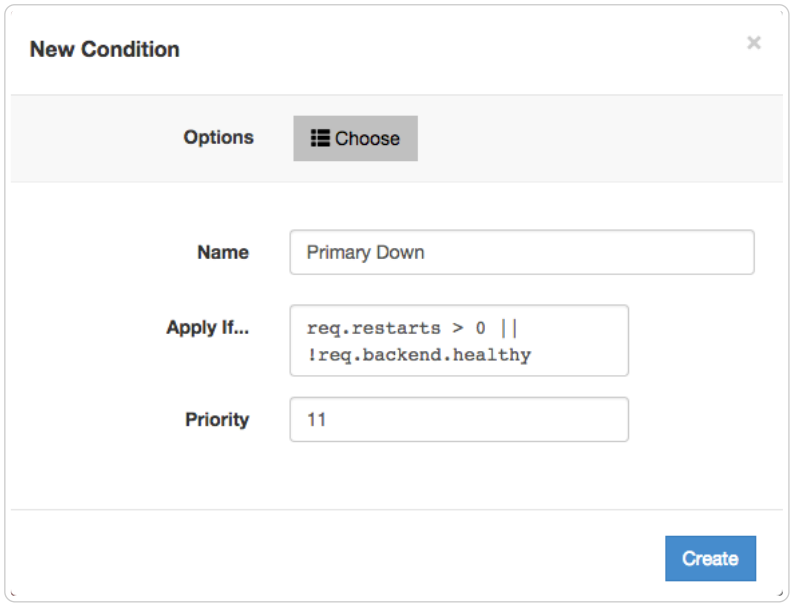
- Leave the **Ignore If Set** control at the default setting.
- In the **Priority** field, type a number at least one higher than the priority you set on the primary server's request header. For example, if you left the first header's priority set to the default, `10`, you would set the second header's priority to `11` or higher.

7. Click **Create** to create the second header. A new header appears in the Headers area that sets the second server as the failover for your primary origin server.

Specify when to use the failover server

Once you've configured your primary and failover servers, create an associated header condition that specifies exactly when the failover server should be used.

1. In the **Headers** area, click the gear icon next to the new header you just created for the failover origin server, and select **Request Conditions**.
2. Click the **New** button to create a new condition. The New Condition window appears.



3. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a descriptive name for the new condition (for example, `Primary Down`).
   - In the **Apply If** field, type the appropriate request condition that will be applied. For example, `req.restarts > 0 || !req.backend.healthy` would tell the system only to use the failover server if the number of restarts is more than 0 or the origin is unhealthy.
   - In the **Priority** field, type `11`.
4. Click **Create** to create the new condition for the header.
5. Activate the new configuration for your service.

---

§ Generating HTTP redirects at the edge (/guides/performance-tuning/generating-http-redirects-at-the-edge)

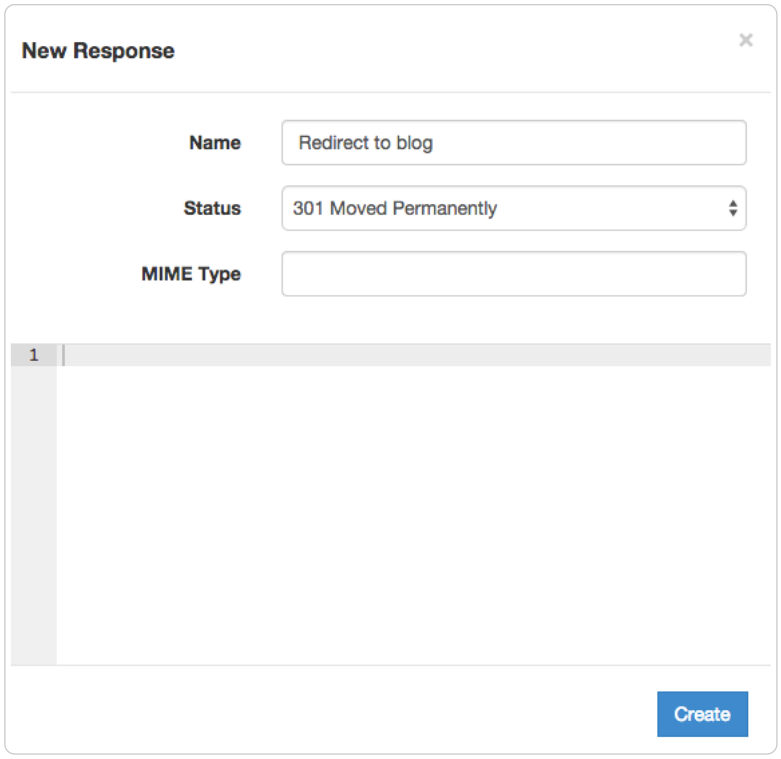
When users request information from your origin servers, you may want to redirect them for various reasons. For example, you may want to redirect them to either a secure or non-secure version of a web page, or to pages that have been moved or updated since the last time they were requested.

You can send these redirects from the edge rather than having to go to origin by creating a synthetic response with the appropriate redirect status code and then creating a content rule with the proper Location header.

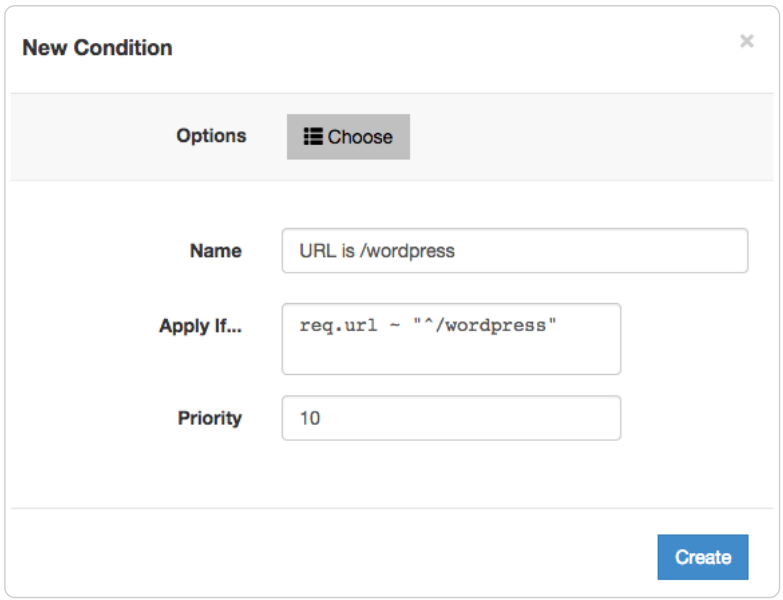
Create a new response and condition

To generate redirects at the edge, start by creating a new response with the appropriate status code and a new condition describing when the response can be applied.

1. Click the **Content** section.
2. In the **Responses** area, click the **New** button to create a new response. The New Response window appears.



3. Fill out the fields of the **New Response** window as follows:
   - In the **Name** field type a meaningful name for your response (e.g., `Redirect to blog`).
   - From the **Status** menu, select the HTTP status code that should be included in the header of the response (e.g., `301 Moved Permanently` or `302 Moved Temporarily` for redirections).
   - Leave the **MIME Type** field blank.
4. Click **Create** to create the new response.
5. Click the gear icon to the right of the new response you just created and select **Request Conditions** from the menu. The Choose Condition window appears.
6. In the **Options** section, click the **New** button. The New Condition window appears.



7. Fill out the fields of the **New Condition** window as follows:
   - In the **Name** field type a meaningful name for your condition (e.g., `URL is /wordpress`).
   - In the **Apply If** field, type the logical expression to execute in VCL to determine if the condition resolves as True or False (e.g., `req.url ~ "^/wordpress"`).

- Leave the **Priority** set to 10.

8. Click **Create** to create the new condition.

Create a new header and condition

Complete the creation of a synthetic redirect by creating a new header and condition that modifies that response by adding the location header based on the status code and the matching URL. This ensures the redirect only applies when both of those are true.

1. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.
2. Fill out the fields of the **New Header** window as follows:
   - In the **Name** field type a meaningful name for your header (e.g., `Location for wordpress redirect`).
   - From the **Type** and **Action** menus, select **Response** and **Set** respectively.
   - In the **Destination** field type `http.location`.
   - In the **Source** field, type the source location of the new content (e.g., `"http://www.my-site.com/new-location/of/item"`).
   - Leave the **Ignore If Set** and **Priority** fields at their default settings.
3. Click **Create** to create the new header.
4. Click the gear icon to the right of the new header you just created and select **Response Conditions** from the menu. The Choose Condition window appears.
5. In the **Options** section, click the **New** button. The New Condition window appears.
6. Fill out the fields of the **New Condition** window as follows:
   - In the **Name** field type a meaningful name for your condition (e.g., `Set location for blog redirect`).
   - In the **Apply If** field, type the logical expression to execute in VCL to determine if the condition resolves as True or False (e.g., `req.url ~ "^/wordpress" && resp.status == 301`). The `resp.status` needs to match the response code generated in the response above.
   - Leave the **Priority** set to 10.
7. Click **Create** to create the new condition.

NOTE: These responses use a custom status number >500. They will appear as errors in the real-time analytics graph (/guides/about-fastly-services/about-the-web-interface-controls#about-the-analytics-dashboard) even though they are desired behavior.

§ Improving caching performance with large files (/guides/performance-tuning/improving-caching-performance-with-large-files)

Fastly provides two features to enhance performance specifically for large files up to 5GB: Streaming Miss and Large File Support.

Streaming Miss

When fetching an object from the origin, Streaming Miss ensures the response is streamed back to the client immediately and is written to cache only after the whole object has been fetched. This reduces the first-byte latency, which is the time that the client must wait before it starts receiving the response body. The larger the response body, the more pronounced the benefit of streaming.

NOTE:

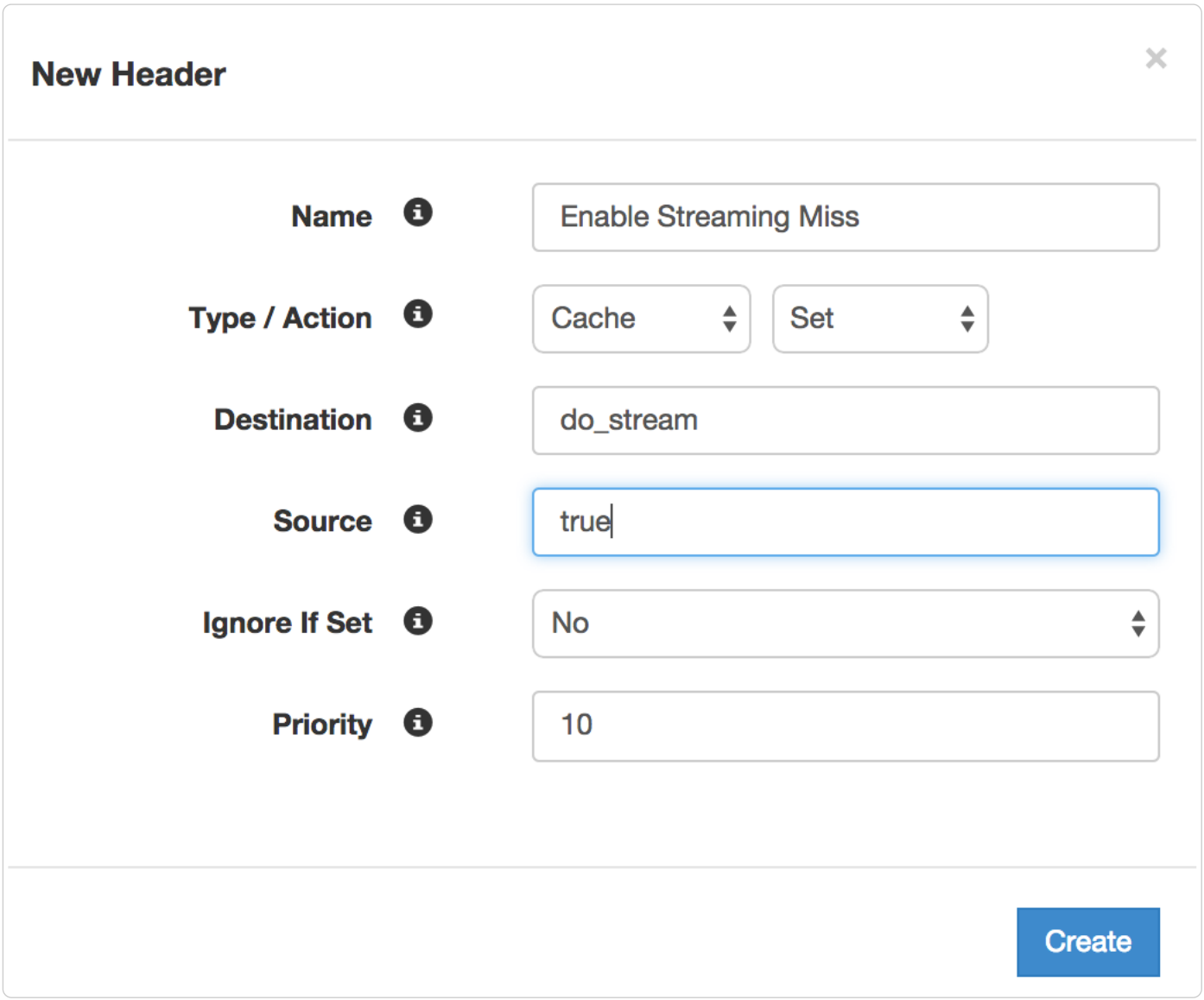
If you enable Streaming Miss, be aware that if an error occurs while transferring the response body, Fastly cannot send an error because the headers are already sent to the client. All we can do is close the connection, truncating the response.

Configuration

Configuration is simple. In VCL, simply set `beresp.do_stream` to true in `vcl_fetch`:

```
sub vcl_fetch {
  ...
  set beresp.do_stream = true;
  ...
  return(deliver);
}
```

The same can be achieved by creating a new header of type `Cache`, action `Set`, destination `do_stream` and source `true` (this can, of course, be controlled with conditions (/guides/conditions/)).



Limitations

There are several limitations to using Streaming Miss.

Origins cannot use TLS

Streaming Miss does not currently support HTTPS (TLS) origin servers. The content can be served to the client over HTTPS, but it cannot be fetched with Streaming Miss over HTTPS. Objects fetched from HTTPS origins are therefore limited to the non-Streaming Miss size of 2GB.

Streaming Miss is not available to HTTP/1.0 clients

If an HTTP/1.0 request triggers a fetch, and the response header from the origin does not contain a Content-Length field, then Streaming Miss will be disabled for the fetch and the fetched object will be subject to the non-streaming-miss object size limit.

If an HTTP/1.0 request is received while a Streaming Miss for an object is in progress, the HTTP/1.0 request will wait for the response body to be downloaded before it will receive the response header and the response body, as if the object was being fetched without Streaming Miss.

Cache hits are not affected. An HTTP/1.0 client can receive a large object served from cache, just like an HTTP/1.1 client.

Streaming Miss is not compatible with on-the-fly gzip compressing of the fetched object

Streaming Miss can handle large files whether or not they are compressed. However, on-the-fly compression of objects that are not already compressed is not compatible with Streaming Miss. If the VCL sets `beresp.gzip` to true, Streaming Miss will be disabled.

Streaming Miss is not compatible with ESI (Edge-Side Includes)

Responses that are processed through ESI cannot be streamed. Responses that are included from an ESI template cannot be streamed. When ESI is enabled for the response or when the response is fetched using `<esi:include>`, then Streaming Miss will be disabled and the fetched object will be subject to the non-streaming-miss object size limit of 2GB.

Large File Support

Large File Support is automatically enabled for all clients — there's no need to manually configure anything. You should, however, be aware that there are maximum file sizes and several failure modes.

Maximum file size

If Streaming Miss is enabled, then the maximum size is slightly below 5GB (specifically 5,368,578,048 bytes). With Streaming Miss disabled, the maximum size is still higher than the previous maximum, but is limited to a little under 2GB (specifically 2,147,352,576 bytes).

Failure modes

There are several failure modes you may encounter while using Large File Support.

What happens when the maximum object size limit is exceeded?

If the response from the origin has a Content-Length header field which exceeds the maximum object size, Fastly will immediately generate a 503 response to the client unless specific VCL is put in place to act on the error.

If no Content-Length header field is returned, Fastly will start to fetch the response body. If while fetching the response body we determine that the object exceeds maximum object size, we will generate a status 503 response to the client (again, unless specific VCL is in place to act on the error).

If no Content-Length header field is present and Streaming Miss is in effect, Fastly will stream the content back to the client. However, if while streaming the response body Fastly determines that the object exceeds the maximum object size, it will terminate the client connection abruptly. The client will detect a protocol violation, because it will see its connection close without a properly terminating 0-length chunk.

What happens when an origin read fails?

A failure to read the response header from the origin, regardless of Streaming Miss, causes a 503 response (which can be acted on in VCL).

If reading the response body from the origin fails or times out, the problem will be reported differently depending on whether Streaming Miss is in effect for the fetch. Without Streaming Miss, a 503 response will be generated. With Streaming Miss, however, it is already too late to send an error response since the header will already have been sent. In this case, Fastly will again abruptly terminate the client connection and the client will detect a protocol violation. If the response was chunked, it will see its connection close without a properly terminating 0-length chunk. If Content-Length was known, it will see the connection close before the number of bytes given.

Incidentally, this is the reason why HTTP/1.0 clients cannot be supported by Streaming Miss in the cases when the Content-Length is not yet known or available. Without the client receiving a Content-Length and without support for chunking, the client cannot distinguish the proper end of the download from an abrupt connection breakage anywhere upstream from it.

---

§ Load-balancing configuration (/guides/performance-tuning/load-balancing-configuration)

This guide describes how to automatically load balance between two or more origin servers. Load balancing distributes requests across multiple servers to optimize resource use and avoid overloading any single resource.

Before you begin

Before you configure load balancing, keep in mind the following:

- To prevent errors when shielding is enabled (/guides/performance-tuning/shielding#enabling-shielding), all backends in the automatic load balancing group must use the same shielding location.
- Conditions on your origin server can directly change how automatic load balancing behaves. Be sure to review conditions behavior to ensure automatic load balancing works properly.
- Many customers configure failover at the same time they configure load balancing functionality. Our guide on configuring failover (/guides/performance-tuning/failover-configuration) can show you how.

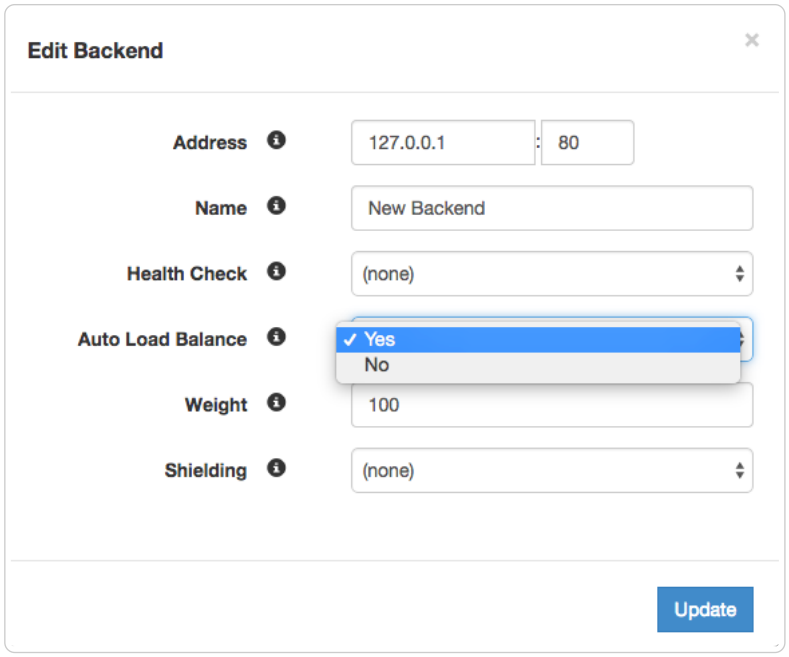
Setting up load balancing between origins

If you want to randomly load balance between two or more origin servers, enable the Fastly automatic load balancing feature.

Enabling load balancing

1. Log in to the Fastly application.
2. Click the **configure** tab (the wrench icon at the top of the application window).
3. Select the appropriate service from the **Service** menu.

4. Click the **Configure** button to the right of the displayed service name.
5. Click the **Hosts** pane from the section list on the left.
6. In the **Backends** area, click the gear icon to the right of the origin server you want to edit.
7. Select **Edit** from the menu. The Edit Backend window appears.

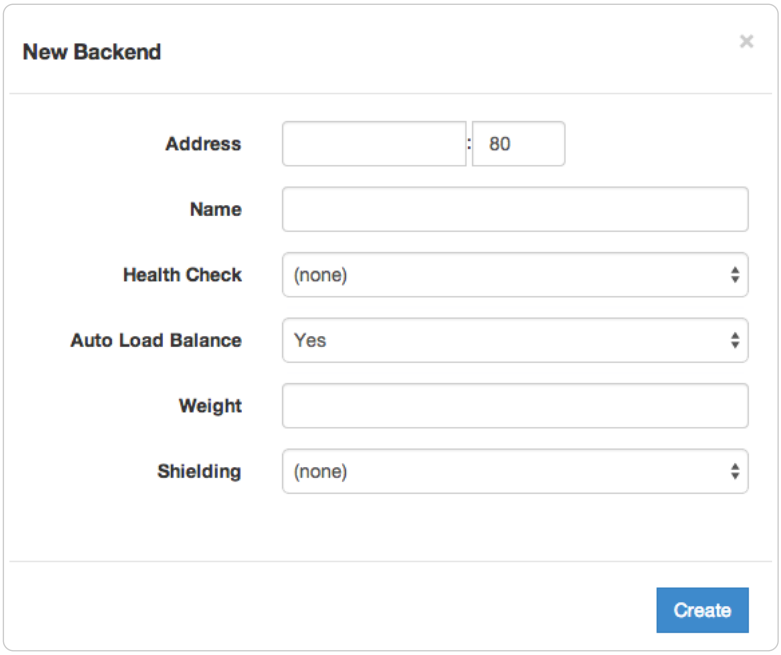


8. Change the setting for **Auto Load Balance** to **Yes** and click **Update**.

You can repeat steps 7 and 8 for each origin server you want to include in the automatic load balancing group.

How Weight affects load balancing

The Weight setting on the New Backend or Edit Backend window allows you to specify the percentage of the total traffic that is sent to specific origins servers. This setting only appears on the windows when you set Auto Load Balance to Yes.



When you specify a whole number in the Weight field, you specify the percentage of the total traffic to send to a specific origin server. Each origin server receives the percentage (<weight>/<total>) of the total traffic equal to the number you specify.

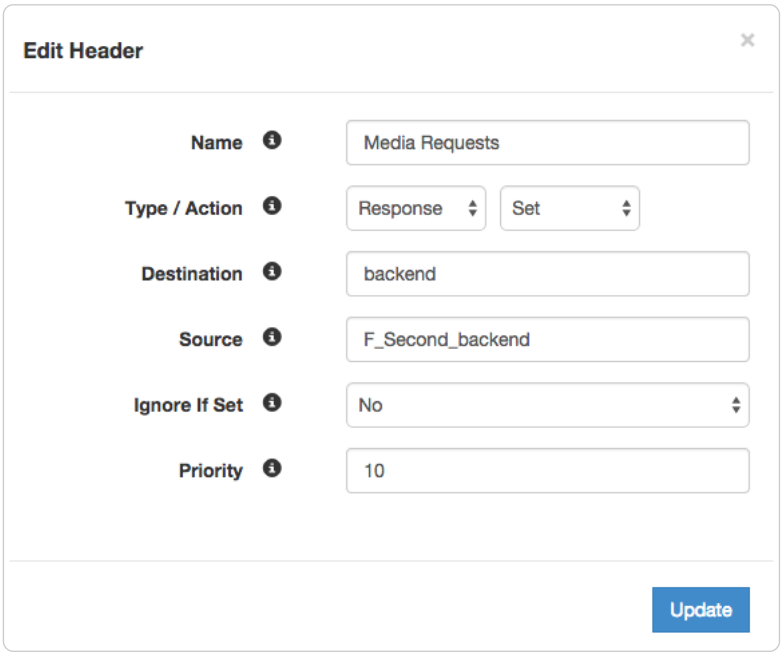
For example, if you have two origin servers, A and B, setting the Weight field to 50 on both splits the traffic between them equally. Each origin server receives 50 percent of your total traffic.

If you increase the Weight setting on origin server A to 55 and decrease the Weight setting on origin server B to 45, the percentage of traffic changes to 55 percent and 45 percent respectively.

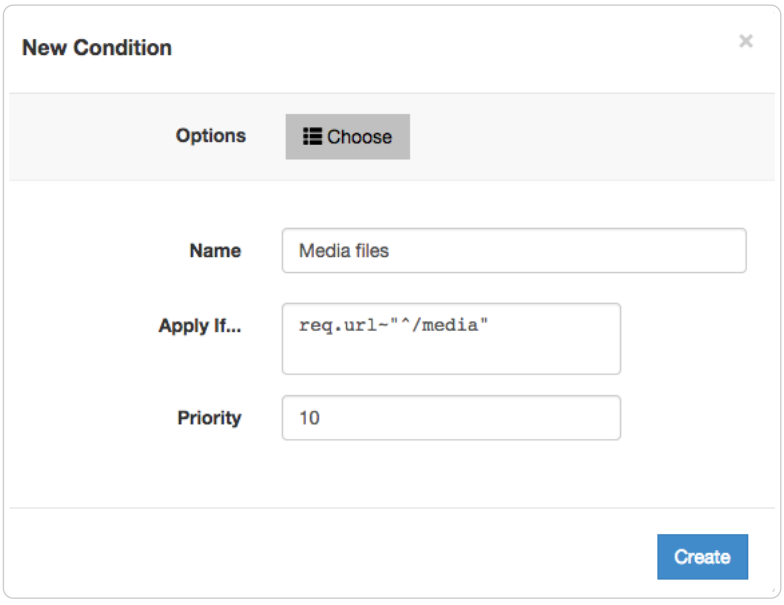
How conditions affect load balancing

When you set conditions (/guides/conditions/) on origin servers you can potentially change how automatic load balancing works. The load balancing autodirector groups servers together based on like conditions, giving you the flexibility to effectively create subsets of the autodirector by assigning a condition to one group of origins and another condition to another set of origins. If each group of origin servers has a different condition that affects load balancing, the auto load function will not randomly load balance between the different servers.

Conditions can also be assigned to a server through a header. For example, you have three servers called F_Fastly, F_Second_backend, and F_Third_backend and want all URLs with a certain prefix to default to the second server.



After the header is created, click the gear icon to the right of the header name and select Request Conditions, then add a new condition to apply if the URL matches the desired prefix.



The generated VCL below illustrates the autodirector set for all three servers. Within the section **sub vcl_recv**, the default origin server is set to the autodirector and, if the media condition is met, requests are forwarded to the second server.

```
director autodirector_ random {
  {
    .backend = F_Second_backend;
    .weight  = 100;
  }{
    .backend = F_Third_Backend;
    .weight  = 100;
  }{
    .backend = F_Fastly;
    .weight  = 100;
  }
}

sub vcl_recv {
#--FASTLY RECV CODE START
  if (req.restarts == 0) {
    if (!req.http.X-Timer) {
      set req.http.X-Timer = "S" time.start.sec "." time.start.usec_frac;
    }
    set req.http.X-Timer = req.http.X-Timer ",VS0";
  }

  # default conditions
  set req.backend = autodirector_;

  # end default conditions

  # Request Condition: Media files Prio: 10
  if (req.url ~ "^/media") {

    # Header rewrite Media Requests : 10
    set req.backend = F_Second_backend;
  }
  #end condition

#--FASTLY RECV CODE END
}
```

§ Maintaining separate HTTP and HTTPS requests to origin servers (/guides/performance-tuning/maintaining-separate-http-and-https-requests-to-backend-servers)

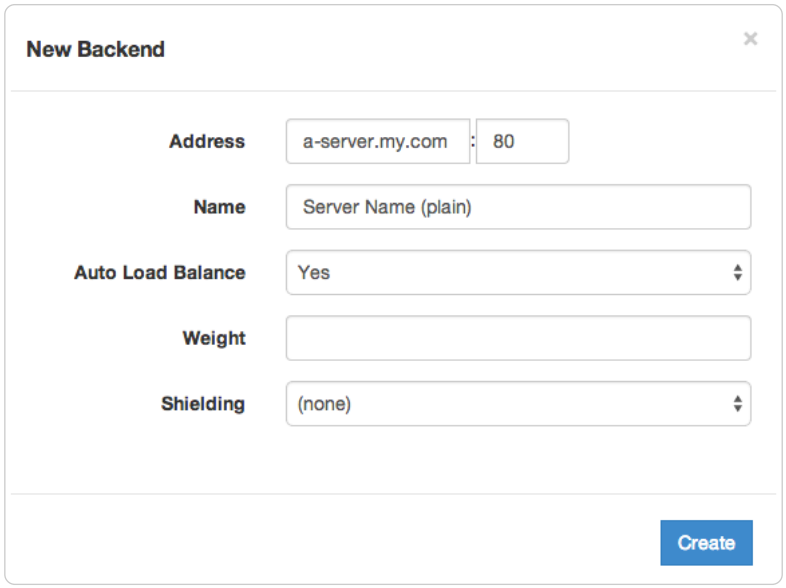
It is common to use the same origin web application to serve both HTTP and HTTPS requests and let the application determine which actions to take to secure communications (/guides/securing-communications/) depending on the incoming protocol. Fastly allows users to set this up to preserve this functionality within their servers. To set Fastly up to send HTTP requests to the non-secure service and HTTPS requests to the secure service, configure two origins, one each for the secure and non-secure ports, then set up the conditions under which requests will be sent there.

Create multiple origins

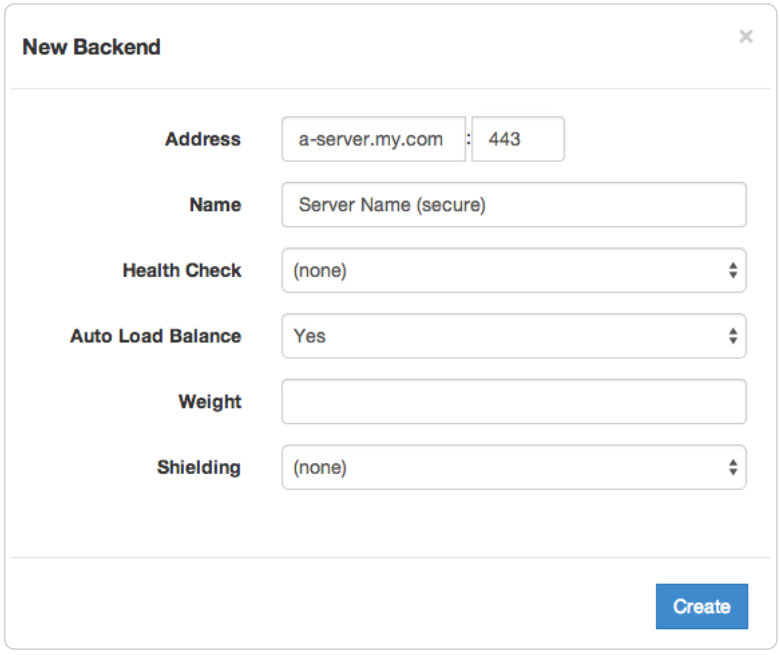
Begin by configuring the same origin address with a different port as a separate origin by following the steps below.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Hosts** section.
6. In the **Backends** area click the **New** button to create a new backend for your non-secure server. The New Backend window appears.



7. Fill out the **New Backend** window as follows:
   - In the **Address** field, type the address of your non-secure server (for example, `a-server.my.com`).
   - In the **Port** field type `80`.
   - In the **Name** field, type the name of your non-secure server (for example, `Server Name (plain)`).
   - Leave the **Health Check**, **Auto Load Balance**, **Weight**, and **Shielding**. controls set to their default values.
8. Click the **Create** button. The non-secure HTTP server appears in the Backends area.
9. In the **Backends** area click the **New** button a second time to create a second, new backend, this time for your secure server.



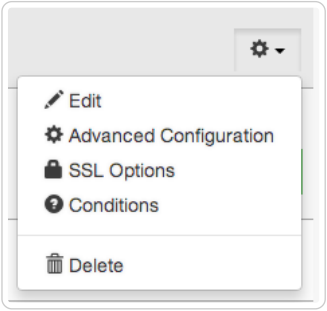
10. Fill out the **New Backend** window as follows:
    - In the **Address** field, type the address of your secure server (for example, `a-server.my.com`).
    - In the **Port** field type `443`.
    - In the **Name** field, type the name of your secure server (for example, `Server Name (secure)`).
    - Leave the **Health Check**, **Auto Load Balance**, **Weight**, and **Shielding**. controls set to their default values.
11. Click the **Create** button. The secure HTTP server appears in the Backends area.

Conditionally send traffic to origins

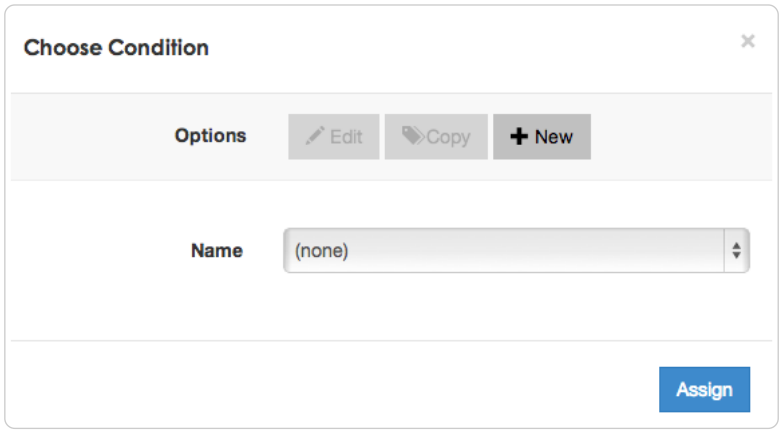
To conditionally determine which server receives secure and non-secure requests, Fastly relies on the presence or absence of a specific header when the backend is selected. When an incoming connection is received over TLS, Fastly sets the `'req.http.fastly-ssl'` header to determine which server to use.

Set a condition for this header on each origin by following the steps below.

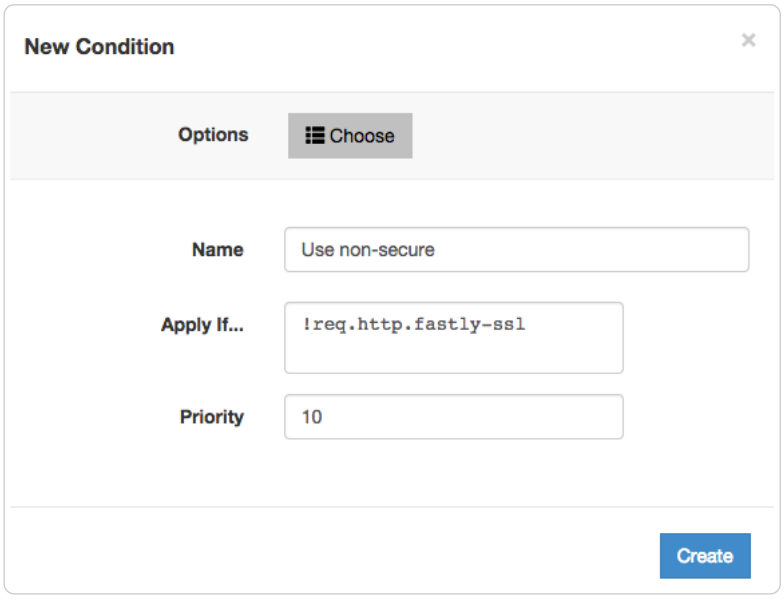
1. In the **Backends** area of the **Hosts** section, click the gear icon next to the name of the non-secure server.



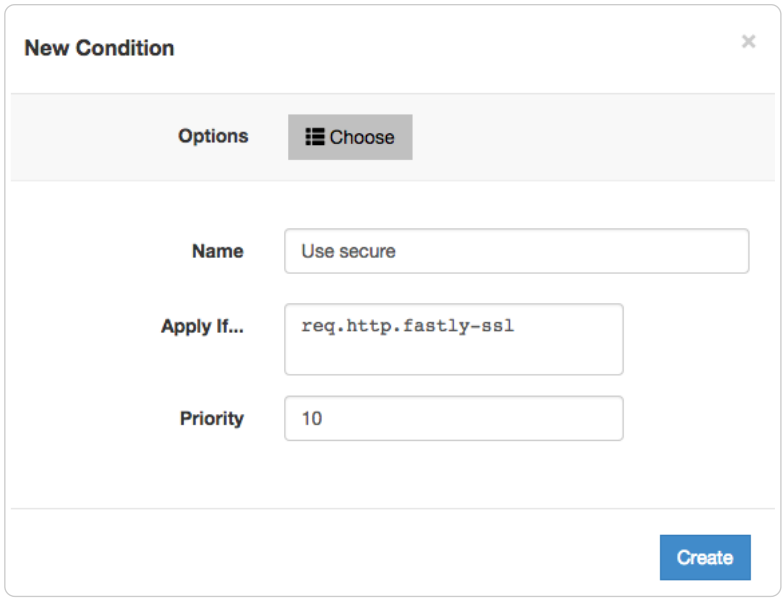
2. From the menu that appears, select **Conditions**. The Choose Condition window appears.



3. Click the **New** button. The New Condition window appears.

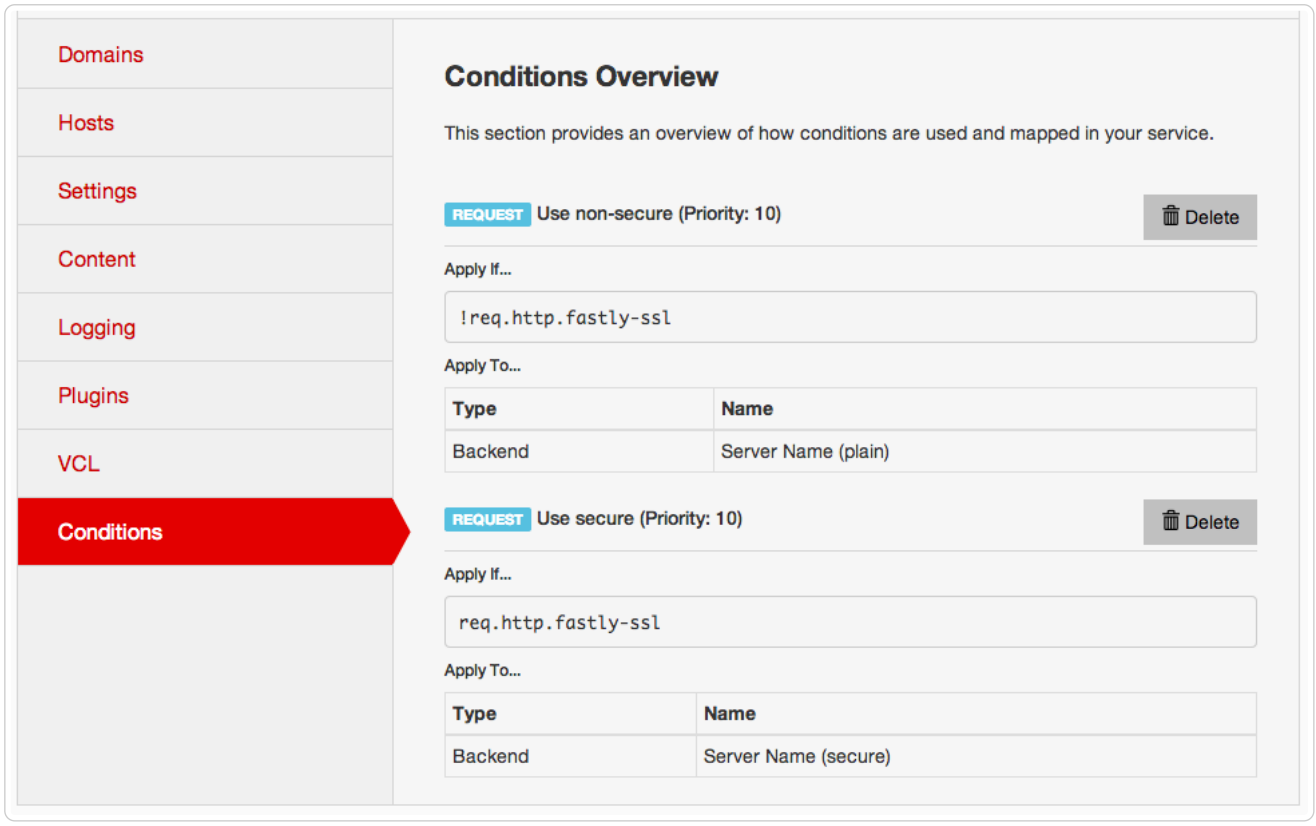


4. Fill out the **New Condition** window as follows:
   - In the **Name** field, type the name of the condition specifying use of the non-secure server (for example, `Use non-secure`).
   - In the **Apply If** field, type `!req.http.fastly-ssl`.
   - Leave the **Priority** field set to its default value.
5. Click the **Create** button to create the new condition.
6. In the **Backends** area of the **Hosts** section, click the gear icon next to the name of the secure server.
7. From the menu that appears, select **Conditions**. The Choose Condition window appears.
8. Click the **New** button. The New Condition window appears.



9. Fill out the **New Condition** window as follows:
   - In the **Name** field, type the name of the condition specifying use of the secure server (for example, `Use secure`).
   - In the **Apply If** field, type `req.http.fastly-ssl`.
   - Leave the **Priority** field set to its default value.
10. Click the **Create** button to create the new condition.

> ★ **TIP:** You can view all conditions and the criteria under which they're applied, including the ones you just created, by examining the Conditions Overview for your service.



11. Deploy the new origins and their conditions by activating the new version of your service.

§ Making query strings agnostic (/guides/performance-

tuning/making-query-strings-agnostic)

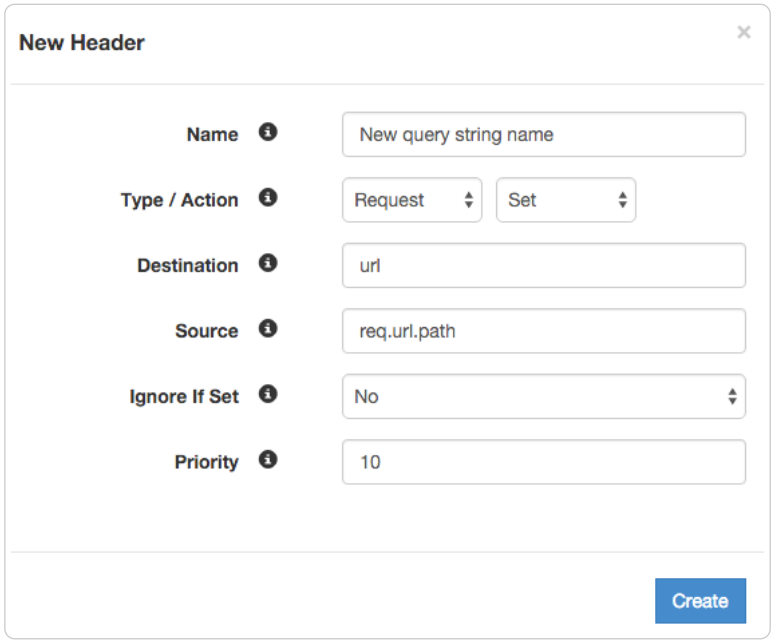
Under normal circumstances, Fastly would consider these URLs as different objects that are cached separately:

- http://example.com
- http://example.com?asdf=asdf
- http://example.com?asdf=zxcv

It is possible, however, to have them all ignore the query string and return the same cached file.

1. Log in the Fastly application and click the **configure** tab (wrench icon).

2. From the **Service** menu, select the appropriate service and then click the blue **Configure** button. The main controls for your selected service appear.
3. Click **Content** from the section list on the left.
4. In the **Headers** area, click the **New** button. The New Header window appears.



5. Fill out the **New Header** window as follows:
   - Set the **Name** field to whatever you want (e.g., "Chop Off Querystring")
   - Select **Request** from the **Type** menu.
   - Select **Set** from the **Action** menu.
   - In the **Destination** field type `url`.
   - In the **Source** field type `req.url.path`
   - Select **No** from the **Ignore If Set** menu.
   - Set the **Priority** field to whatever priority you want.
6. Click the **Create** button to create the new header. The new header you created appears in the list of headers on the Content page.

The request will be sent to the origin as a URL without the query string.

For more information about controlling caching, see our Cache Control Tutorial (/guides/tutorials/cache-control-tutorial).

§ Request collapsing (/guides/performance-tuning/request-collapsing)

This guide describes Fastly's Request Collapsing feature, frequently used when creating advanced service configurations.

NOTE: This guide requires advanced knowledge of Varnish and the VCL language (/guides/vcl/guide-to-vcl).

The basics

Request Collapsing causes simultaneous cache misses within a single Fastly data center to be "collapsed" into a single request to an origin server. While the single request is being processed by the origin, the other requests wait in a queue for it to complete. Two types of Request Collapsing exist:

1. Collapsing on a single cache server
2. Collapsing within the data center between cache servers

Each cache server will automatically queue duplicate requests for the same hash and only allow one request to origin. You can disable this behavior by setting `req.hash_ignore_busy` to `true` in `vcl_recv`.

Within a data center, not every cache stores every object. Only two servers in each data center will store an object: one as a primary and one as a backup. Only those two servers will fetch the object from origin.

How it works

In Fastly's version of Varnish, VCL subroutines often run on different caches during a request. For a particular request, both an edge cache and a shield cache will exist (though a single cache can, in some cases, fulfill both of these roles). The edge cache receives the HTTP request from the client and determines via a hash which server in the data center is the shield cache. If this cache determines it is the shield cache and has the object in cache, it fulfills both the edge cache and the shield cache roles.

Certain VCL subroutines run on the edge cache and some on the shield cache:

- Edge Cache: *`vcl_recv`*, *`vcl_deliver`*, *`vcl_log`*, *`vcl_error`*
- Shield Cache: *`vcl_miss`*, *`vcl_hit`*, *`vcl_pass`*, *`vcl_fetch`*, *`vcl_error`*

Determining if a cache is an edge or a shield

The `fastly_info.is_cluster_edge` VCL variable will be true if the cache currently running the VCL is the edge cache and false if it is the shield cache.

Caveats

Keep in mind the following limitations when using the Request Collapsing feature:

1. Any `req.http.*` headers are not transferred from the shield cache back to the edge cache. Remember this when writing advanced configurations that use headers to keep track of state. If you set a `req.http.*` header in any of the subroutines that run on the shield cache, expect that the change will not persist on the edge cache.
2. A single, slow request to origin can sometimes cause a great many other requests for the same object to hang and fail. Because many requests for a single object are being collapsed down to one, they all succeed or fail based on the request that reaches the origin.

---

§ Routing assets to different origins (/guides/performance-tuning/routing-assets-to-different-origins)

Some customers have assets stored on multiple origin servers and want to route various requests to specific, different servers based on criteria they supply (e.g., asset type, file directory, host header). Fastly offers customers the power to set conditions on their origins, which simply adds an if statement block to your VCL.

Basic setup: Create conditions for each origin

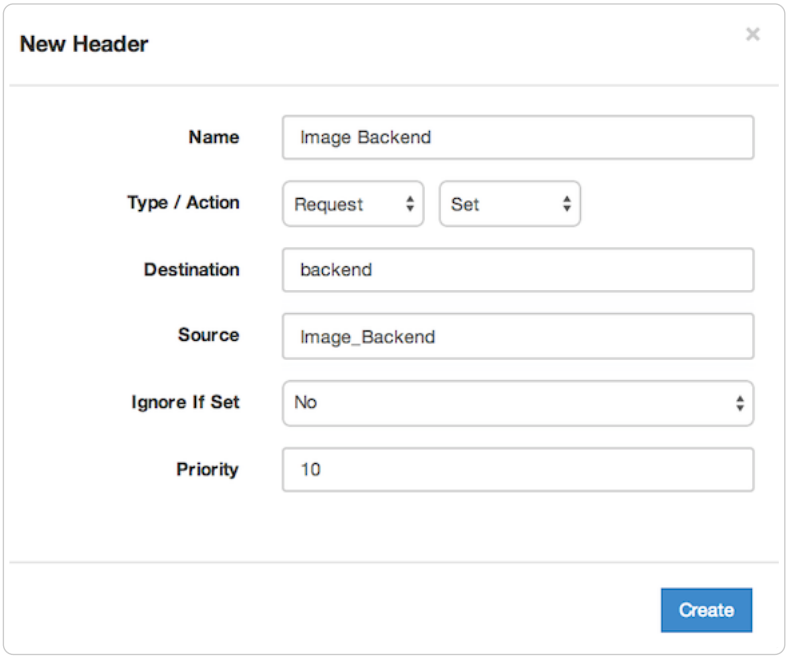
To add a condition on your origin, navigate to **Hosts** → **Backends** within the Fastly application. Click the gear icon next to a specific origin, and select **Conditions** from the menu that appears to create the new condition you wish to apply to that origin. Here are some example criteria and the conditions that would be placed in the **Apply If** field of the **New Condition** you create:

Criteria	Sample Conditions
hosts	`req.http.host ~ "www.example.com"`
content-type / url	`req.url ~ ".(jpg|png|gif)($|\?)"`

Backup Setup: Create a Header

What if you have a condition already assigned to your origin? Although you can group request conditions on the origin with an 'and' or 'or' clause, there can only ever be one condition rule attached to that origin. If you want to separate your request conditions instead of grouping them, you can use header rules to route assets to different origins instead.

Within the Fastly application, navigate to **Content** → **Headers**. Then select the **New** button to create the first header that will set the default origin.

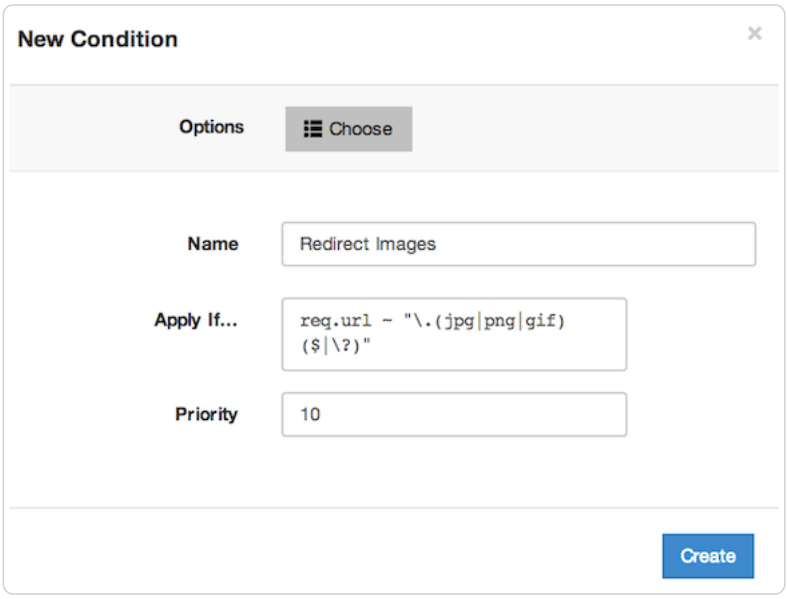


Set the first header's controls as follows:

- **Name:** `Image Backend` (or any meaningful, preferred name)
- **Type/Action:** Request/Set
- **Destination:** `backend`
- **Source:** `Image_Backend` (This should match the name of your global origin server. You can see the exact name if you look at your VCL. Click on the VCL button at the top of the page.)
- **Ignore If Set:** `No`
- **Priority:** `10`

Once you've set the controls, click **Create** to add the first new header.

Next click the gear icon to add a condition. Select **New** to add a new condition.



Enter the condition settings as follows:

- **Name:** `Redirect Images` (or any meaningful, preferred name)
- **Apply if**: `req.url ~ "\.(jpg|png|gif)($|\?)"`

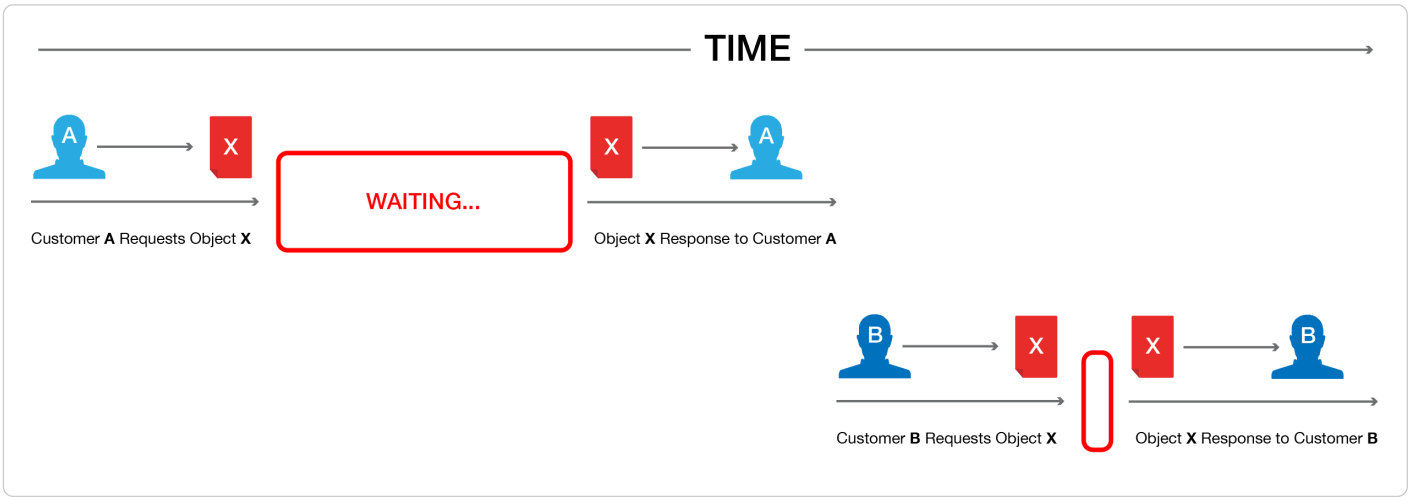
Once you have finished, click **Create** to assign the condition to your header. Then click the **Activate** button, near the top right corner of the window, to deploy the new version of the service you just edited.

★ **TIP:** Learn more about conditions in our Conditions subcategory (/guides/conditions/).

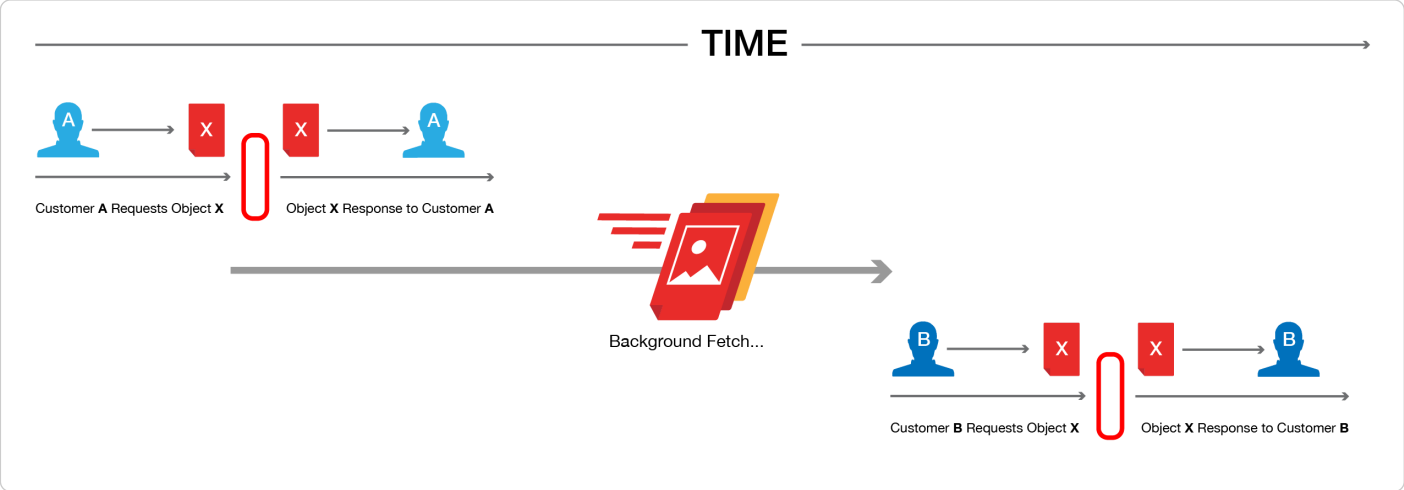
§ Serving stale content (/guides/performance-tuning/serving-stale-content)

Serving old content while fetching new content

Certain pieces of content can take a long time to generate. Once the content is cached it will be served quickly, but the first user to try and access it will pay a penalty.



This is unavoidable if the cache is completely cold, but if this is happening when the object is in cache and its TTL is expired, then Fastly can be configured to show the stale content while the new content is fetched in the background.



Fastly builds on the behavior proposed in RFC 5861 (http://tools.ietf.org/html/rfc5861) "HTTP Cache-Control Extensions for Stale Content" by Mark Nottingham, which is under consideration for inclusion in Google's Chrome browser (https://www.mnot.net/blog/2014/06/01/chrome_and_stale-while-revalidate).

Usage

To activate this behavior simply add a `stale-while-revalidate` or `stale-if-error` statement to either the Cache-Control or Surrogate-Control headers in the response from your origin server. For example:

```
Cache-Control: max-age=600, stale-while-revalidate=30
```

will cache some content for 10 minutes and, at the end of that 10 minutes, will serve stale content for up to 30 seconds while new content is being fetched.

Similarly, this statement:

```
Surrogate-Control: max-age=3600, stale-if-error=86400
```

instructs the cache to update the content every hour (3600 seconds) but if the origin is down then show stale content for a day (86400 seconds).

Alternatively, these behaviors can be controlled from within VCL by setting the following variables in vcl_fetch:

```
set beresp.stale_while_revalidate = 30s;
```

```
set beresp.stale_if_error = 86400s;
```

Interaction with grace

Stale-if-error works exactly the same as Varnish's grace variable such that these two statements are equivalent:

```
set beresp.grace = 86400s;
```

```
set beresp.stale_if_error = 86400s;
```

However, if a grace statement is present in VCL it will override any `stale-while-revalidate` or `stale-if-error` statements in any Cache-Control or Surrogate-Control response headers.

Setting `beresp.stale_if_error` either via header or via VCL does nothing on its own. In order to serve stale, follow the instructions below.

Serving stale content on errors

In certain situations where your origin server becomes unavailable, you may want to serve stale content. These instructions provide an advanced configuration that allows all three possible origin failure cases to be handled using VCL. These instructions require the ability to upload custom VCL.

> **IMPORTANT:** The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

In the context of Varnish, there are three ways an origin can fail:

- The origin can be marked as unhealthy by failing healthchecks.
- If Varnish cannot contact the origin for any reason, a 503 error will be generated.
- The origin returns a valid HTTP response, but that response is not one we wish to serve to users--for instance a 503.

The custom VCL shown below handles all three cases. If the origin is unhealthy, the default serve stale behavior is triggered by `stale-if-error`. In between the origin failing and being marked unhealthy, Varnish would normally return 503s. The custom VCL allows us to instead either serve stale if we have a stale copy, or to return a synthetic error page. The error page can be customized. The third case is handled by intercepting all 5XX errors in `vcl_fetch` and either serving stale or serving the synthetic error page.

> ⚠ **WARNING:** Do not purge all (/guides/purging/single-purges#purging-all-content) cached content if you are seeing 503 errors (/guides/debugging/common-503-errors). Purge all overrides stale-if-error and increases the requests to your origin server, which could result in additional 503 errors.

Although not strictly necessary, healthchecks (/guides/basic-configuration/health-checks-tutorial) should be enabled in conjunction with this VCL. Without healthchecks enabled, all of the functionality will still work, but serving stale or synthetic responses will take much longer while waiting for an origin to timeout. With healthchecks enabled, this problem is averted by the origin being marked as unhealthy.

The custom VCL shown below includes the Fastly standard boilerplate. Before uploading this to your service, be sure to customize or remove the following values to suit your specific needs:

- `if (beresp.status >= 500 && beresp.status < 600)` should be changed to include any HTTP response codes you wish to serve stale/synthetic for.
- `set beresp.stale_if_error = 86400s;` controls how long content will be eligible to be served stale and should be set to a meaningful amount for your configuration. If you're sending `stale_if_error` in Surrogate-Control or Cache-Control from origin, remove this entire line.
- `set beresp.stale_while_revalidate = 60s;` controls how long the `stale_while_revalidate` feature will be enabled for an object and should be set to a meaningful amount for your configuration. This feature causes Varnish to serve stale on a cache miss and fetch the newest version of the object from origin in the background. This can result in large performance gains on objects with short TTLs, and in general on any cache miss. Note that `stale_while_revalidate` overrides `stale_if_error`. That is, as long as the object is eligible to be served stale while revalidating, `stale_if_error` will have no effect. If you're sending `stale_while_revalidate` in Surrogate-Control or Cache-Control from origin, remove this entire line.
- `synthetic {"<!DOCTYPE html>Your HTML!</html>"};` is the synthetic response Varnish will return if no stale version of an object is available and should be set appropriately for your configuration. You can embed your HTML, CSS, or JS here. Use caution when referencing external CSS and JS documents. If your origin is offline they may be unavailable as well.

```
sub vcl_recv {
  if (req.http.Fastly-FF) {
    set req.max_stale_while_revalidate = 0s;
  }

#FASTLY recv

  if (req.request != "HEAD" && req.request != "GET" && req.request != "FASTLYPURGE") {
    return(pass);
  }

  return(lookup);
}

sub vcl_fetch {
  /* handle 5XX (or any other unwanted status code) */
  if (beresp.status >= 500 && beresp.status < 600) {

    /* deliver stale if the object is available */
    if (stale.exists) {
      return(deliver_stale);
    }

    if (req.restarts < 1 && (req.request == "GET" || req.request == "HEAD")) {
      restart;
    }

    /* else go to vcl_error to deliver a synthetic */
    error 503;
  }

  /* set stale_if_error and stale_while_revalidate (customize these values) */
  set beresp.stale_if_error = 86400s;
  set beresp.stale_while_revalidate = 60s;

#FASTLY fetch

  if ((beresp.status == 500 || beresp.status == 503) && req.restarts < 1 && (req.request == "GET" || req.request == "HEAD")) {
    restart;
  }

  if (req.restarts > 0) {
    set beresp.http.Fastly-Restarts = req.restarts;
  }

  if (beresp.http.Set-Cookie) {
    set req.http.Fastly-Cachetype = "SETCOOKIE";
    return(pass);
  }

  if (beresp.http.Cache-Control ~ "private") {
    set req.http.Fastly-Cachetype = "PRIVATE";
    return(pass);
  }

  /* this code will never be run, commented out for clarity */
  /* if (beresp.status == 500 || beresp.status == 503) {
     set req.http.Fastly-Cachetype = "ERROR";
     set beresp.ttl = 1s;
     set beresp.grace = 5s;
     return(deliver);
  } */

  if (beresp.http.Expires || beresp.http.Surrogate-Control ~ "max-age" || beresp.http.Cache-Control ~ "(s-maxage|max-age)") {
    # keep the ttl here
  } else {
    # apply the default ttl
    set beresp.ttl = 3600s;
  }

  return(deliver);
}

sub vcl_hit {
#FASTLY hit
```

```
  if (!obj.cacheable) {
    return(pass);
  }
  return(deliver);
}

sub vcl_miss {
#FASTLY miss
  return(fetch);
}

sub vcl_deliver {
  if (resp.status >= 500 && resp.status < 600) {
    /* restart if the stale object is available */
    if (stale.exists) {
      restart;
    }
  }

#FASTLY deliver
  return(deliver);
}

sub vcl_error {
#FASTLY error

  /* handle 503s */
  if (obj.status >= 500 && obj.status < 600) {

    /* deliver stale object if it is available */
    if (stale.exists) {
      return(deliver_stale);
    }

    /* otherwise, return a synthetic */

    /* include your HTML response here */
    synthetic {"<!DOCTYPE html><html>Please replace this text with the error page you would like to serve to clients if your origin
 is offline.</html>"};
    return(deliver);
  }

}

sub vcl_pass {
#FASTLY pass
}
```

Additional reading

- RFC 5861 (http://tools.ietf.org/html/rfc5861)
- Mark Nottingham's blog post (https://www.mnot.net/blog/2014/06/01/chrome_and_stale-while-revalidate)
- Chrome discussion (https://groups.google.com/a/chromium.org/forum/#!msg/chromium-dev/zchogDvIYrY/ZqWSdt3LJdMJ)

§ Shielding (/guides/performance-tuning/shielding)

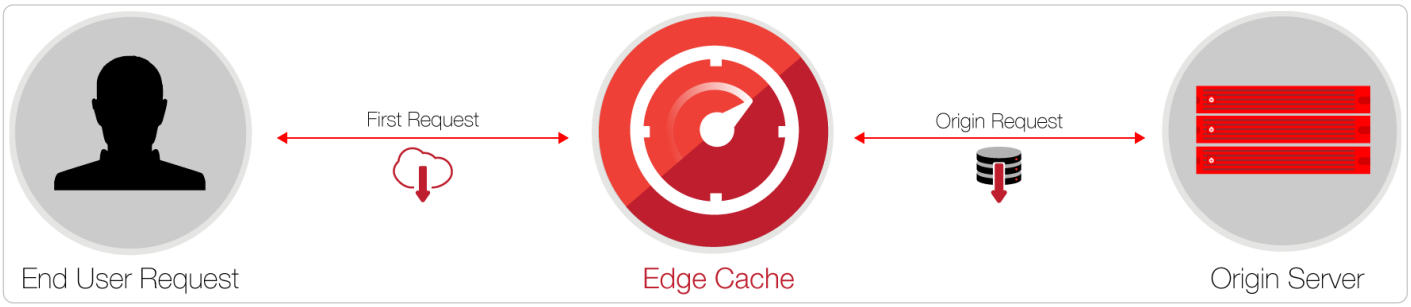
Fastly's shielding service feature (/guides/about-fastly-services/about-fastlys-origin-shielding-features) allows you to designate a specific POP as a shield node to your origins. Once designated, all requests to your origin will go through that data center, increasing the chances of getting a HIT (/guides/performance-tuning/understanding-cache-hit-and-miss-headers-with-shielded-services) for a given resource. If a different POP doesn't have a specific object, it will query the shield (provided it's not down for maintenance) instead of your origins.

How shielding works

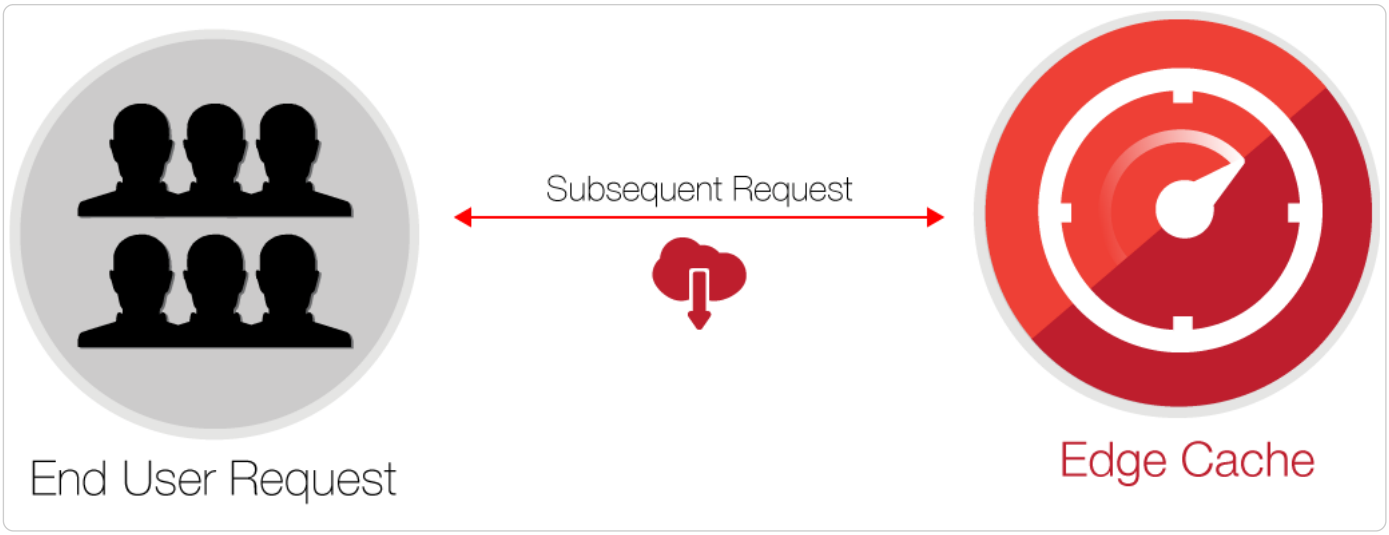
When a user requests brand new content from a customer's server and that content has never been cached by any Fastly POP (/guides/about-fastly-services/fastly-pop-locations), this is what happens to their request when shielding is and is not enabled.

Without shielding enabled

Without shielding enabled, when the first request for content arrives at POP A, the POP does not have the content cached. It passes the request along to a customer's origin server to get the content. Once the content is retrieved, POP A caches it and sends it on to the user.



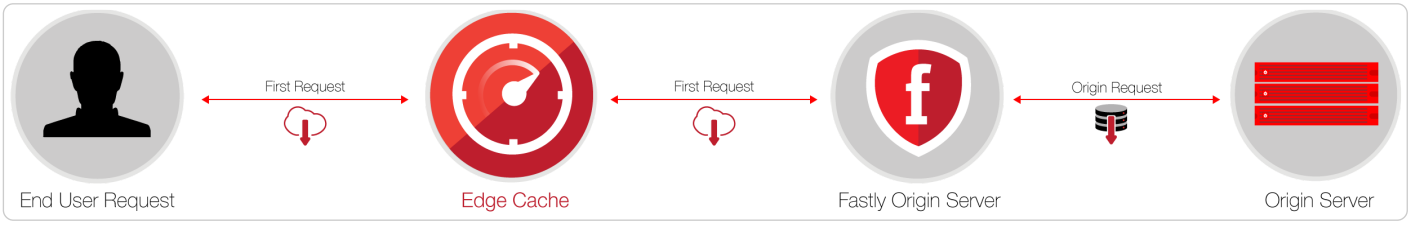
When a second request for that same content arrives at POP A, the content is already cached. No request goes to the customer's origin server. It's merely sent from the cached copy.



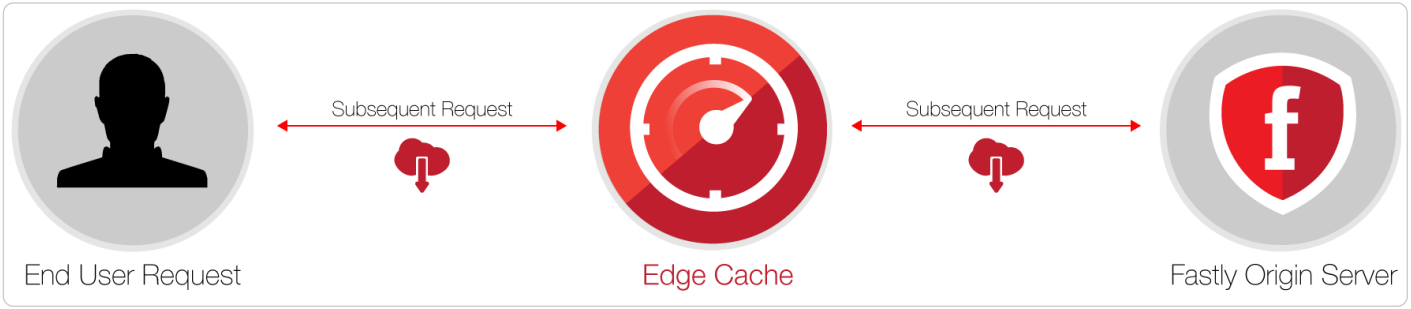
If the second request were to arrive at POP B instead of POP A, however, the request would once again be passed along to the customer's origin server. It would then be cached and passed back to the end user, just like POP A did when that info was first requested.

With shielding enabled

With shielding enabled (/guides/performance-tuning/shielding), when the first request for content arrives at the POP A, that POP does not have the content cached. It passes the request along to the shield POP, which also doesn't have the content cached. The shield POP passes the request along to the customer's origin server. It then caches the content that's retrieved and passes it along to POP A. POP A then passes the content along to the user.



When a second request for that same content arrives at POP A, the content is already cached, so no request goes to the shield POP or the customer's origin server.



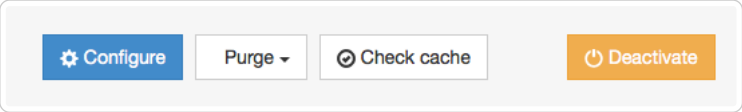
If the second request were to arrive at POP B instead of POP A, however, the request would be passed along to the shield POP. That shield POP already has a cached copy from the first request to POP A. No future requests for the content would be passed along to the customer's origin server until the shield POP's cached copy of it expires.

Enabling shielding

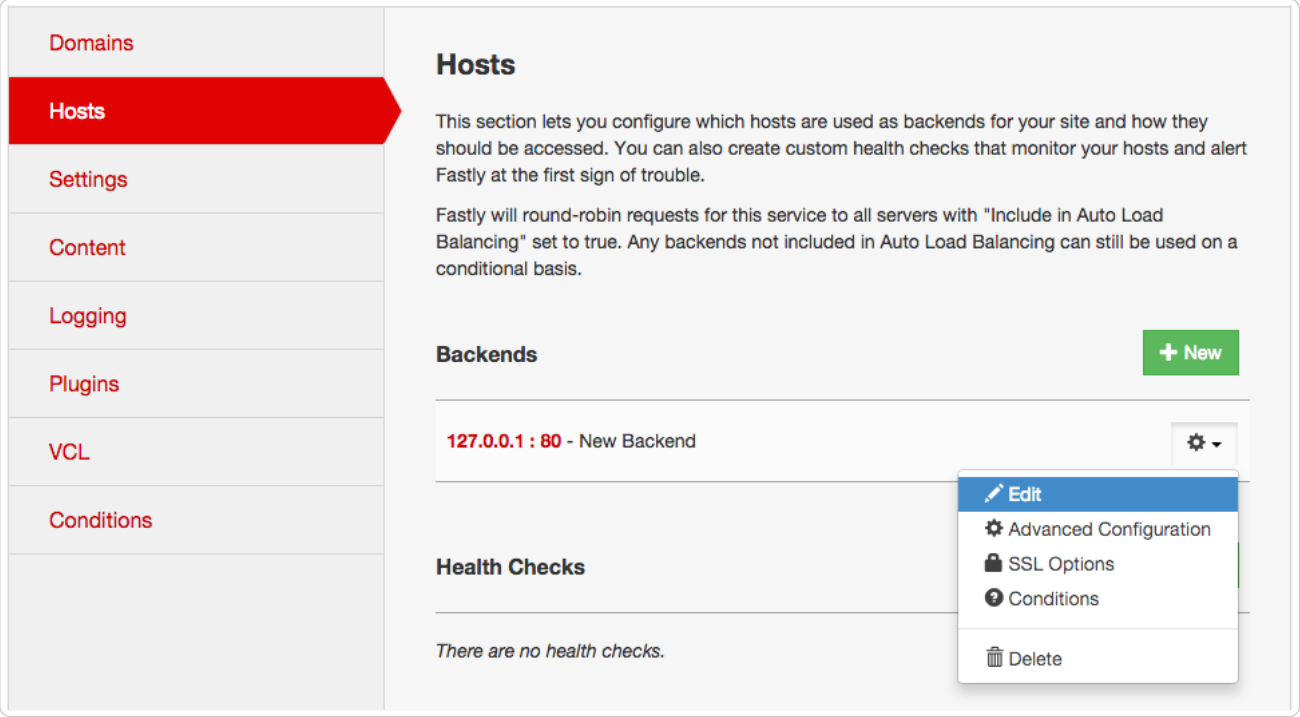
IMPORTANT: If you are using Google Cloud Storage as your origin, you need to follow the steps in our GCS setup guide (/guides/integrations/google-cloud-storage) instead of the steps below.

Enable shielding with these steps:

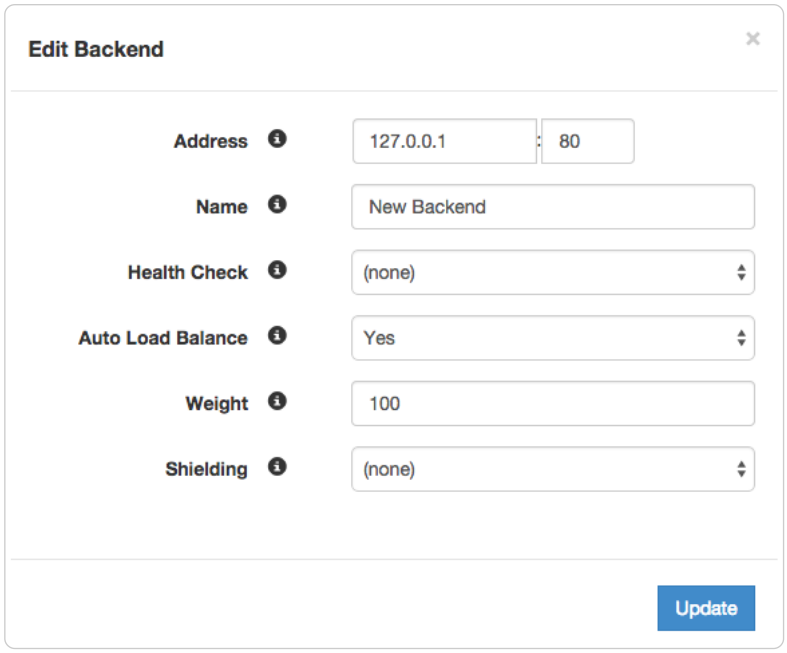
1. Read the caveats of shielding information below for details about the implications of and potential pitfalls involved with enabling shielding for your organization.
2. Log in to the Fastly application and click the **configure** button (wrench icon).
3. From the **Service** menu, select the service for which shielding will be enabled and then click the blue **Configure** button (with the gear icon) to the right of the service name.



4. Click **Hosts** from the section list on the left.
5. In the **Backends** area, click the gear icon to the right of the appropriate backend and then select **Edit** from the menu.



The Edit Backend window appears.

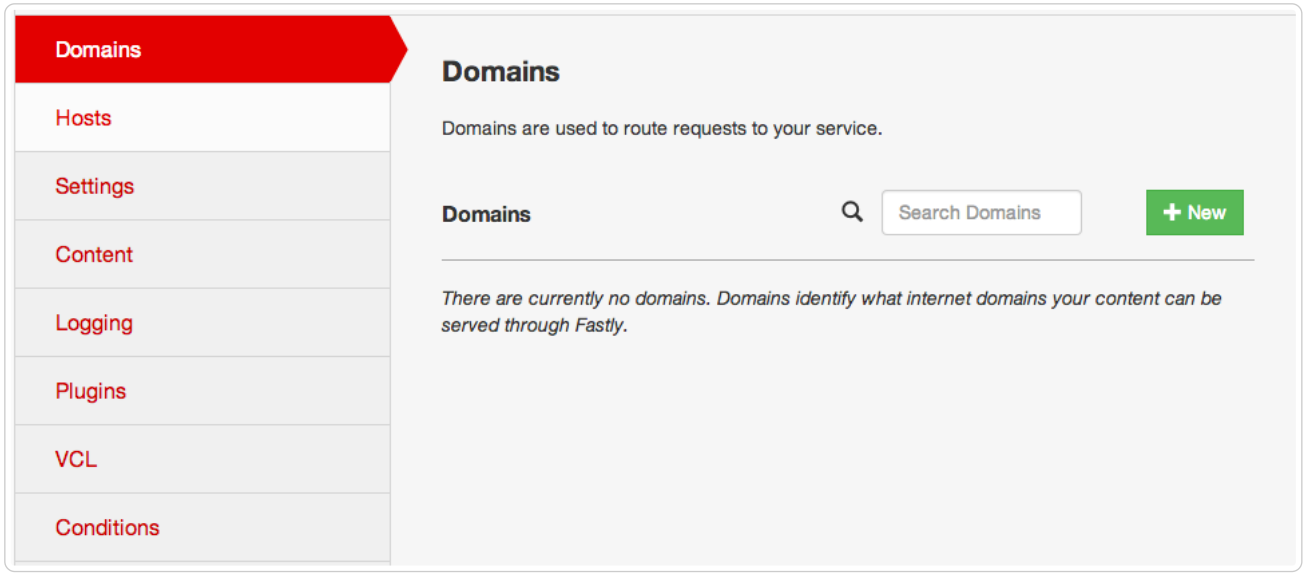


6. From the **Shielding** menu, select the data center to use as your shield and then click the blue **Update** button to save your changes. Generally, we recommend selecting a data center close to your backend. Doing this allows faster content delivery because we optimize requests between the edge POP (the one close to your user) and the shield POP (the one close to your server).

   NOTE: With multiple backends, each backend will have its own shield defined. This allows flexibility if your company has backends selected geographically and different shield POPs are desired.

7. If you have changed the default host or have added a header to change the host, add the modified hostname to your list of domains.

   Do this by clicking **Domains** from the section list to the left and checking to make sure the host in question appears in the **Domains** section of the page. If it isn't included, add it via the **New** button.



   With shielding enabled, queries from other POPs appear as incoming requests to the shield. If the shield doesn't know about the modified hostname, it doesn't know which service to match the request to. Including the origin's hostname in the domain list eliminates this concern.

8. Deploy a new version of your service to activate shielding.

Caveats of shielding

Shielding not only impacts traffic and hit ratios, it affects configuration and performance. When you configure shielding, be aware of the following caveats.

Traffic and hit ratio caveats

Inbound traffic to a shield will be billed as regular traffic, including requests to populate remote POPs. Enabling shielding will incur some additional Fastly bandwidth charges, but will be offset by savings of your origin bandwidth (and origin server load). Pass-through requests will not go directly to the origin, they will go through the shield first.

Global HIT ratio calculation may seem lower than the actual numbers. Shielding is not taken into account when calculating the global hit ratio. If an edge node doesn't have an object in its cache, it reports a miss. Local MISS/Shield HIT gets reported as a miss and a hit in the statistics, even though there is no call to the backend. It will also result in one request from the edge node to the shield. Local MISS/Shield MISS will result in two requests, because we will subsequently fetch the resource from your origin. For more information about caching with shielding see our article Understanding Cache HIT and MISS with Shielding Services (/guides/performance-tuning/understanding-cache-hit-and-miss-headers-with-shielded-services).

Configuration caveats

You will be unable to manually define backends using VCL. Shielding at this level is completely dependent on backends being defined as actual objects through the user interface or API. Other custom VCL (/guides/vcl/uploading-custom-vcl) will work just fine.

You can only set one shield total if automatic load balancing is selected via the user interface. If you've selected auto load balancing, you can only select one shield total. You must use custom VCL to use multiple shields when auto load balancing is set.

Enabling sticky load balancing and shielding at the same time requires custom VCL. Sticky load balancers use `client.identity` to choose where to send the session. The `client.identity` defaults to the IP request header. That's fine under normal circumstances, but if you enable shielding, the IP will be the original POP's IP, not the client's IP. Thus, to enable shielding and a custom sticky load balancer, you want to use the following:

```
if (req.http.fastly-ff) {
  set client.identity = req.http.Fastly-Client-IP;
}
```

Performance caveats

You'll need to use caution when changing the host header before it reaches the shield. Fastly matches a request with a host header. If the host header doesn't match to a domain within the service an error of 500 is expected. Also, purging conflicts can occur if the host header is changed to a domain that exists in a different service.

For example, say Service A has hostname `a.example.com` and Service B has hostname `b.example.com`. If Service B changes the host header to `a.example.com`, then the edge will think the request is for Service B but the shield will think the request is for Service A.

When you purge an object from Service B and not from Service A, the shield will serve the old object that you wanted to purge to the edge, since the purge went out to Service B and not Service A. You will want to purge the object from both Service A and Service B. However, this opens the door for confusion and error.

VCL gets executed twice: once on the shield node and again on the edge node. Changes to `beresp` and `resp` can affect the caching of a URL on the shield and edge. Consider the following examples.

Say you want Fastly to cache an object for one hour (3600 seconds) and then ten seconds on the browser. The origin sends `Cache-Control: max-age=3600`. You unset `beresp.http.Cache-Control` and then reset `Cache-Control` to `max-age=10`. With shielding enabled, however, the result will not be what you expect. The object will have `max-age=3600` on the shield and reach the edge with `max-age=10`.

A better option in this instance would be to use `Surrogate-Control` and `Cache-Control` response headers. `Surrogate-Control` overrides `Cache-Control` and is stripped after the edge node. The `max-age` from `Cache-Control` will then communicate with the browser. The response headers would look like this:

```
Surrogate-Control: max-age=3600
Cache-Control: max-age=10
```

Another common pitfall involves sending the wrong `Vary` header to an edge POP. For example, there's VCL that takes a specific value from a cookie, puts it in a header, and that header is then added to the `Vary` header. To maximize compatibility with any caches outside of your control (such as with shared proxies as commonly seen in large enterprises), the `Vary` header is updated in `vcl_deliver`, replacing the custom header with `Cookie`. The code might look like this:

```
vcl_recv {
  # Set the custom header
  if (req.http.Cookie ~ "ABtesting=B") {
    set req.http.X-ABtesting = "B";
  } else {
    set req.http.X-ABtesting = "A";
  }
  ...
}

...

sub vcl_fetch {
  # Vary on the custom header
  if (beresp.http.Vary) {
    set beresp.http.Vary = beresp.http.Vary ", X-ABtesting";
  } else {
    set beresp.http.Vary = "X-ABtesting";
  }
  ...
}

...

sub vcl_deliver {
  # Hide the existence of the header from downstream
  if (resp.http.Vary) {
    set resp.http.Vary = regsub(resp.http.Vary, "X-ABtesting", "Cookie");
  }
}
```

When combined with shielding, however, the effect of the above code will be that edge POPs will have `Cookie` in the `Vary` header, and thus will have a terrible hit rate. To work around this, amend the above VCL so that `Vary` is only updated with `Cookie` when the request is not coming from another Fastly cache. The `Fastly-FF` header is a good way to tell. The code would look something like this (including the same `vcl_recv` from the above example):

```
# Same vcl_recv from above code example

sub vcl_fetch {
  # Vary on the custom header, don't add if shield POP already added
  if (beresp.http.Vary !~ "X-ABtesting") {
    if (beresp.http.Vary) {
      set beresp.http.Vary = beresp.http.Vary ", X-ABtesting";
    } else {
      set beresp.http.Vary = "X-ABtesting";
    }
  }
  ...
}

...

sub vcl_deliver {
  # Hide the existence of the header from downstream
  if (resp.http.Vary && !req.http.Fastly-FF) {
    set resp.http.Vary = regsub(resp.http.Vary, "X-ABtesting", "Cookie");
  }
}
```

§ Tracking your origin's name, IP, and port (/guides/performance-tuning/tracking-your-origins-name-ip-and-port)

Fastly provides three values captured in `vcl_fetch` that allow you to see and track origin information:

- `beresp.backend.name`
- `beresp.backend.port`
- `beresp.backend.ip`

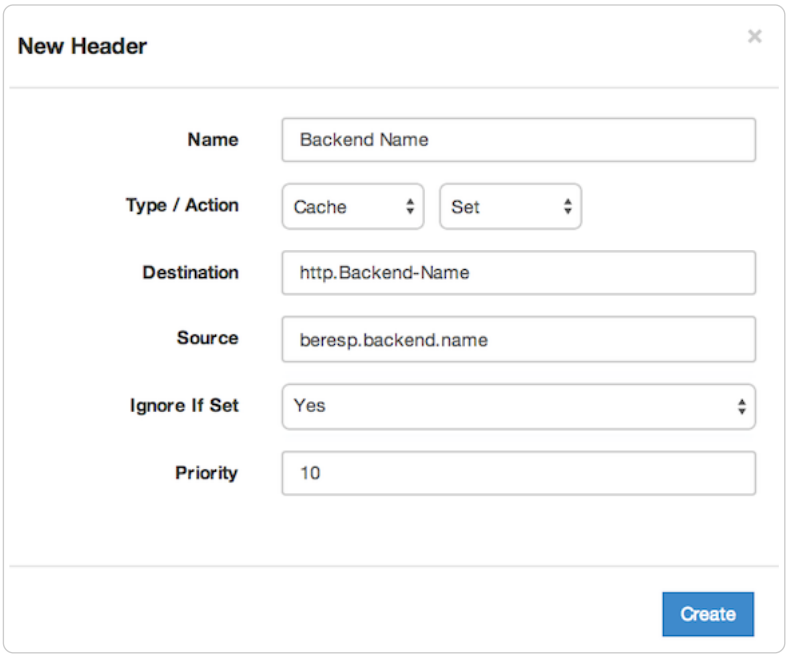
While these three values are immensely useful, you may want to use this information within `vcl_deliver` for things like response information or remote log streaming. You can do this by:

1. Creating cache headers that capture the origin information.
2. Adding a response header to the log format to capture the response output.

Capturing the Origin Information

To track your origin's name, IP, and port, you need to create two separate headers: one that captures the origin name and another that captures the origin's IP and port (e.g., 80, 443).

Create the header that captures the origin name by launching the Fastly application and navigating to **Content** → **Headers**. Then, click the **New** button to display the New Header window.

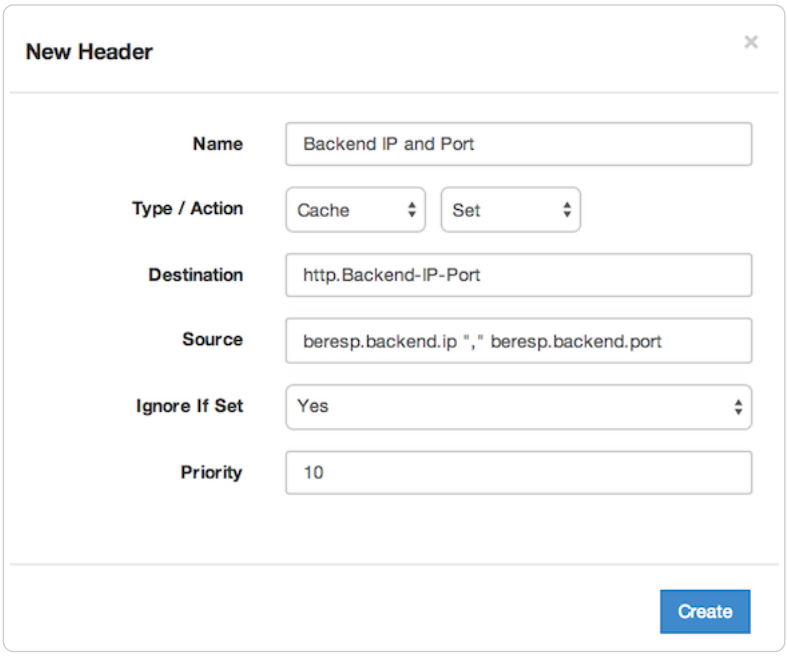


Set this first header's controls as follows:

- **Name:** Backend Name (or preferred name)
- **Type/Action**: Cache/Set
- **Destination:** `http.Backend-Name` (or preferred header variable)
- **Source:** `beresp.backend.name`
- **Ignore If Set:** Yes
- **Priority:** `10`

Once you've set the controls, click **Create** to add the first new header.

Create the second header to capture the IP and port for your origin by navigating again to **Content** → **Headers.** Then, click the **New** button to display the New Header window a second time.



Set the second header's controls as follows:

- **Name:** Backend IP and Port (or preferred name)
- **Type/Action**: Cache/Set
- **Destination:** `http.Backend-Name` (or preferred header variable)
- **Source:** `beresp.backend.ip "," beresp.backend.port`
- **Ignore If Set:** Yes
- **Priority:** 10

Once you've set the controls, click **Create** to add the second new header.

Adding a Response Header to the Log Format

The values captured in a header within `vcl_fetch` will flow to `vcl_deliver`. For example, there will exist a resp.http.Backend-Name header in `vcl_deliver` that corresponds to beresp.http.Backend-Name in `vcl_fetch`. By default, the response header will be included in the response output.

Unfortunately, with remote log streaming, you cannot add the `vcl_fetch` header, beresp.http.Header-Name, to the log format. However, you can add its cousin in `vcl_deliver`, resp.http.Header-Name.

Add resp.http.Header-Name to the log format that you configured by following the instructions in the Remote Log Streaming guide (/guides/streaming-logs/setting-up-remote-log-streaming). Using the example above, you would add resp.http.Backend-Name and resp.http.Backend-IP-Port.

Important Notes

Regarding Shielding

Notice within the example the field **Ignore If Set** is set to **Yes**. With shielding, the VCL is executed twice, once on the shield and again on the edge node. This setting will display the original information from the origin without overriding it on the edge node.

You can also set the field **Ignore If Set** to **No** with shielding enabled. In this scenario, the edge node captures the shield's information within `beresp.backend.name`, `beresp.backend.ip`, and `beresp.backend.port`.

If remote log streaming is configured, remember it is executed twice. Thus the first log (from the shield node) will have the origin's information and the second log (from the edge node) will have the shield's information.

Regarding Security

For security purposes, you may want to track the information in logging but not display all or some of it in the response. This is possible but requires custom VCL to strip the information after sending the log line from the edge node.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Don't forget to read our guide to using custom VCL (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl) before you begin. Remember to include the entire boilerplate if you do not intend to override the Fastly default settings.

Then add the following snippet within `vcl_deliver` with the headers you want to strip. Using our example above, we continue to send the value of the origin's name within the response. We strip the origin's IP and port from the response information.

```
sub vcl_deliver {
#FASTLY deliver

  if (!req.http.Fastly-FF) {
    unset resp.http.Backend-IP-Port;
  }

  return(deliver);
}
```

§ Understanding cache HIT and MISS headers with shielded services (/guides/performance-tuning/understanding-cache-hit-and-miss-headers-with-shielded-services)

Here's some help deciphering cache hit and miss headers when you have shielding enabled (/guides/performance-tuning/shielding). Let's look at the following requests for the same object if you had run a curl in your terminal (for example, `curl -svo /dev/null www.example.com`) to return the Fastly headers.

The first request for an object using the above curl might produce output something like this:

```
X-Served-By: cache-iad2120-IAD, cache-sjc3120-SJC
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
```

For this first request, the two cache-nodes in `X-Served-By` show that shielding is turned on, with `cache-iad2120-IAD` serving as the cache node and `cache-sjc3120-SJC` serving as the local delivering node. The `X-Cache: MISS, MISS` indicates that the requested object was neither in the shield cache (a MISS) nor the local delivering node (also a MISS). The `X-Cache-Hits` reflects that same MISS information because it displays `0, 0`.

The second request for an object using the above curl might produce output something like this:

```
X-Served-By: cache-iad2120-IAD, cache-sjc3120-SJC
X-Cache: MISS, HIT
X-Cache-Hits: 0, 1
```

This second time, we hit the same local cache-node (`cache-sjc3120-SJC`) and got a HIT. The MISS listed for `cache-iad2120-IAD` reflects the state of that node the last time it queried that node for that object (not its current state), which at that time was a MISS, but in fact, the object resides in that cache now because it was requested from origin.

Waiting a minute or two and requesting the same object a third time using the above curl might produce output something like this:

```
X-Served-By: cache-iad2120-IAD, cache-sjc3120-SJC
X-Cache: HIT, MISS
X-Cache-Hits: 1, 0
```

This third request shows a new cache (`cache-sjc3120-SJC`) being selected, MISSing, and retrieving the object from the shield node (`cache-iad2120-IAD`), which had the object (a HIT). In total, there should have only been a single request for this object to the origin.

Keep in mind that if the closest delivering cache node exists in the shield data center, you will only see a single server and HIT data such as:

```
X-Served-By: cache-iad2120-IAD
X-Cache: HIT
X-Cache-Hits: 1
```

After a purge of the object, requesting the object again via the above curl will produce results similar to the first request scenario. For example:

```
X-Served-By: cache-iad2120-IAD, cache-sjc3120-SJC
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
```

§ Understanding the X-Timer header (/guides/performance-

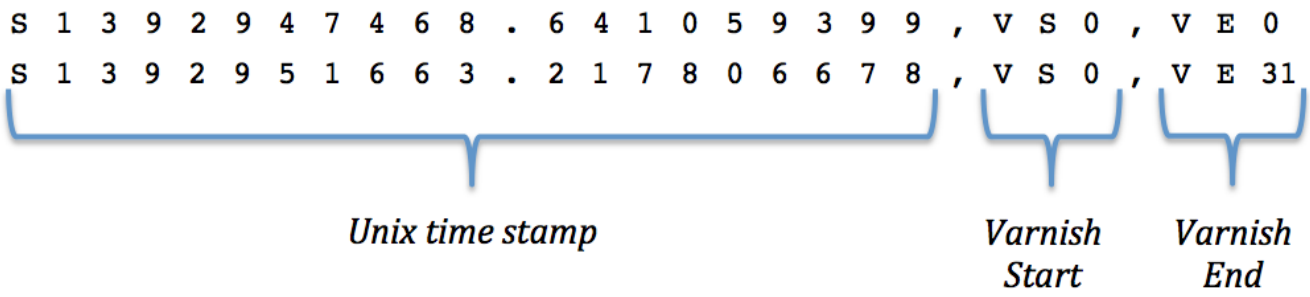
tuning/understanding-the-xtimer-header)

If you look at the raw headers returned with a response from a Fastly cached asset, you will notice some extra headers tacked on. One in particular is X-Timer. This header provides timing information about the journey of a request from end to end.

Here are two examples of X-Timer headers:

- `S1392947468.641059399,VS0,VE0` (a cache HIT)
- `S1392951663.217806578,VS0,VE31` (a cache MISS)

Let's break these headers down into their parts, separated by commas, and examine what each part means.



The first section of the header, starting with `S`, represents a Unix timestamp (https://en.wikipedia.org/wiki/Unix_time) of the start of the request on our edges.

The next section, `VS` or "varnish start," represents the start of the varnish part of the request's journey. This should always be 0 (we've got to start counting somewhere).

And the last section, `VE` or "varnish end," represents the sum of the length of the trip. For cache HITs, the length of the trip will nearly always be 0 (not actually zero, but less than a millisecond is rounded down). For cache MISSs, the number represents the number of milliseconds it took to retrieve the data from your origin server and send the response back to the requester. In the example above, it took 31ms to retrieve the data.

§ Using edge side includes (ESI) (/guides/performance-tuning/using-edge-side-includes)

Fastly supports the following edge side includes (ESI) language (http://www.w3.org/TR/esi-lang) elements:

- include
- comment
- remove

We don't support the following ESI language elements:

- inline
- choose | when | otherwise
- try | attempt | except
- vars
- ESI Variables

• Guides (/guides/) > Diagnostics and performance (/guides/diagnostics) > Security (/guides/security/)

§ Penetration testing your service behind Fastly (/guides/security/penetration-testing-your-service-behind-fastly)

We understand the need for our customers to validate the security of their service behind Fastly.

IMPORTANT: Penetration tests that interfere with or disrupt the integrity or performance of Fastly services violate our acceptable use policy (https://www.fastly.com/acceptable-use). You must respond immediately to any communication from Fastly regarding your test to help ensure your testing does not adversely affect other customers or the Fastly network.

To perform security testing of your Fastly service configurations, create a Customer Support ticket by contacting Fastly via email at support@fastly.com (mailto:support@fastly.com) at least two (2) business days before you begin any security testing. In your ticket, include these details:

- the IDs of the services (/guides/account-management-and-security/finding-and-managing-your-account-info#finding-your-service-id) that will be tested
- the source IP address of the test
- the date of the test
- the start and end time of the test, including the time zone
- the contact information for the individual or third party performing the test, including a phone number and e-mail address
- whether or not the security test is likely to lead to significantly increased traffic volume

The following requirements apply to any security testing you perform:

- Only test Fastly services you own or are authorized by the owner to test. You may not perform tests against other customers without explicit permission or against Fastly-owned resources.
- Do not begin testing until after Fastly has responded affirmatively to your ticket and authorized your request.
- Update the ticket if either the scope or timeframe of your testing changes.
- If you discover vulnerabilities in the Fastly platform during your test, update the ticket with your findings as soon as possible so we can address them.

Fastly maintains programs for security (/guides/compliance/security-program) and technology compliance (/guides/compliance/technology-compliance). To perform an independent audit of these programs, contact sales@fastly.com (mailto:sales@fastly.com) to discuss purchase of Compliance Services (/guides/customer-support/compliance-services).

★ **TIP:** We welcome security professionals researching potential vulnerabilities in our network under our guidelines for reporting a security issue (https://www.fastly.com/security/report-security-issue).

• Guides (/guides/) > Migrations and integrations (/guides/mig-int) > Migrations (/guides/migrations/)

§ WordPress plugins: Migrating to Purgely from Fastly's original plugin (/guides/migrations/migrating-to-purgely-from-fastlys-original-plugin)

Purgely (https://wordpress.org/plugins/purgely/) is an open source WordPress plugin written by Condé Nast that integrates Fastly services into the WordPress platform. We recommend customers use Purgely over Fastly's original WordPress plugin (https://wordpress.org/plugins/fastly/) released in 2011, because Purgely takes advantage of newer WordPress and Fastly features, including Fastly's surrogate key (/guides/purging/getting-started-with-surrogate-keys) purging. Purgely also takes fewer steps to configure and delivers content to users faster than the original plugin.

IMPORTANT: You can continue to use Fastly's original WordPress plugin (https://wordpress.org/plugins/fastly/) without service interruption. However, we do not plan to continue its development.

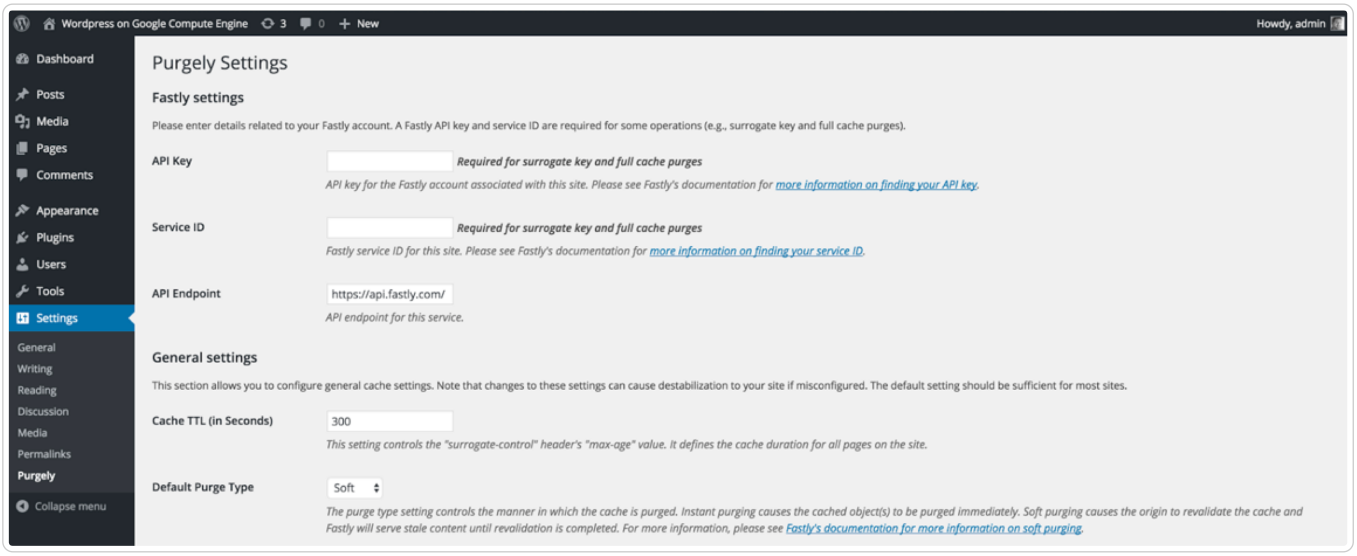
Recommended migration path

To migrate from Fastly's WordPress plugin to the Purgely plugin, follow these steps.

1. Find your Fastly API key and Service ID in one of the following ways:
   - Look in the Fastly application (/guides/account-management-and-security/finding-and-managing-your-account-info).
   - Look in the Fastly WordPress plugin settings in the Plugins menu (https://codex.wordpress.org/Plugins_Screen) of your WordPress instance.
2. Uninstall Fastly's WordPress plugin from your WordPress instance by following the Uninstalling Plugins

(https://codex.wordpress.org/Managing_Plugins#Uninstalling_Plugins) instructions.

3. Install the Purgely plugin using one of the following the methods:
   - **GUI installation (recommended).** Follow the Automatic Plugin Installation (https://codex.wordpress.org/Managing_Plugins#Automatic_Plugin_Installation) instructions to add it as a new plugin by searching for it in the listed plugins and then clicking **Install Now** on the plugin's **Details** screen.
   - **Command line installation.** Install the WordPress command line tool (https://codex.wordpress.org/wp-cli) and then run the install command by typing `wp plugin install --activate purgely`.
   - **Manual installation.** Follow the Manual Plugin Installation (https://codex.wordpress.org/Managing_Plugins#Manual_Plugin_Installation) instructions to upload Purgely to your `/wp-content/plugins/` directory and then activate it from the Plugins menu (https://codex.wordpress.org/Plugins_Screen) in the WordPress interface.
4. From the WordPress **Plugins** menu, click the **Settings** link in the navigation bar.
5. Click **Purgely**. The Purgely settings window appears.



6. Configure the Purgely plugin by doing the following:
   - In the **API Key** field, type your Fastly API key.
   - In the **Service ID** field, type your Fastly service ID.
7. Click the **Save** button.

★ **TIP:** For other Purgely configuration settings, see the configuration instructions on the Purgely plugin installation tab (https://wordpress.org/plugins/purgely/installation/).

• Guides (/guides/) > Migrations and integrations (/guides/mig-int) > Integrations (/guides/integrations/)

§ Acquia Cloud (/guides/integrations/acquia-cloud)

To use Acquia Cloud as an origin you must sign up for both an Acquia Cloud subscription and Fastly services and connect the two.

Sign up for an Acquia Cloud subscription

1. Using a web browser, navigate to the Acquia Cloud signup page (https://www.acquia.com/free).
2. Select the Acquia Cloud Free option. The account subscription form appears.
3. Fill out the form and click **Create** to sign up for a subscription and start Acquia's automated site creation process.

   The automated portion of the Acquia subscription process can take three to five minutes to complete. You'll know the entire process ends successfully when you see the checkmark appear next to the word "Done."

Check for domain alias conflicts

Ensure that your new site domain does not have a conflicting alias by running the `host` command on a command line. For example, the `host` command for the Test-Example-2 domain would be:

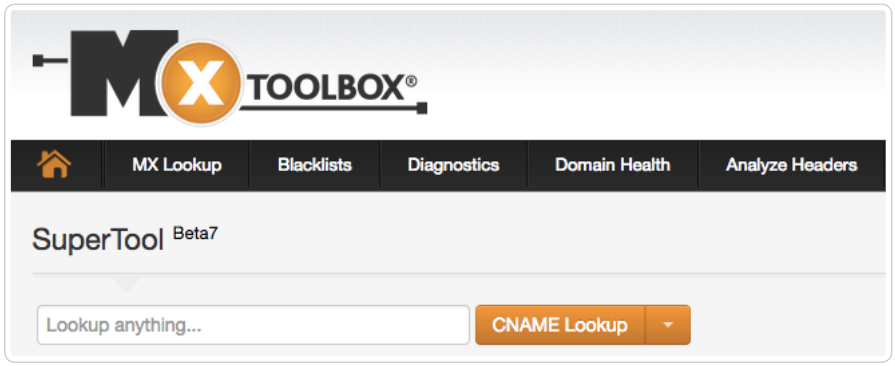
`host test-example-2.devcloud.acquia-sites.com`.

and would produce the following sample output:

```
test-example-2.devcloud.acquia-sites.com has address 127.0.0.1
```

Determine your CNAME

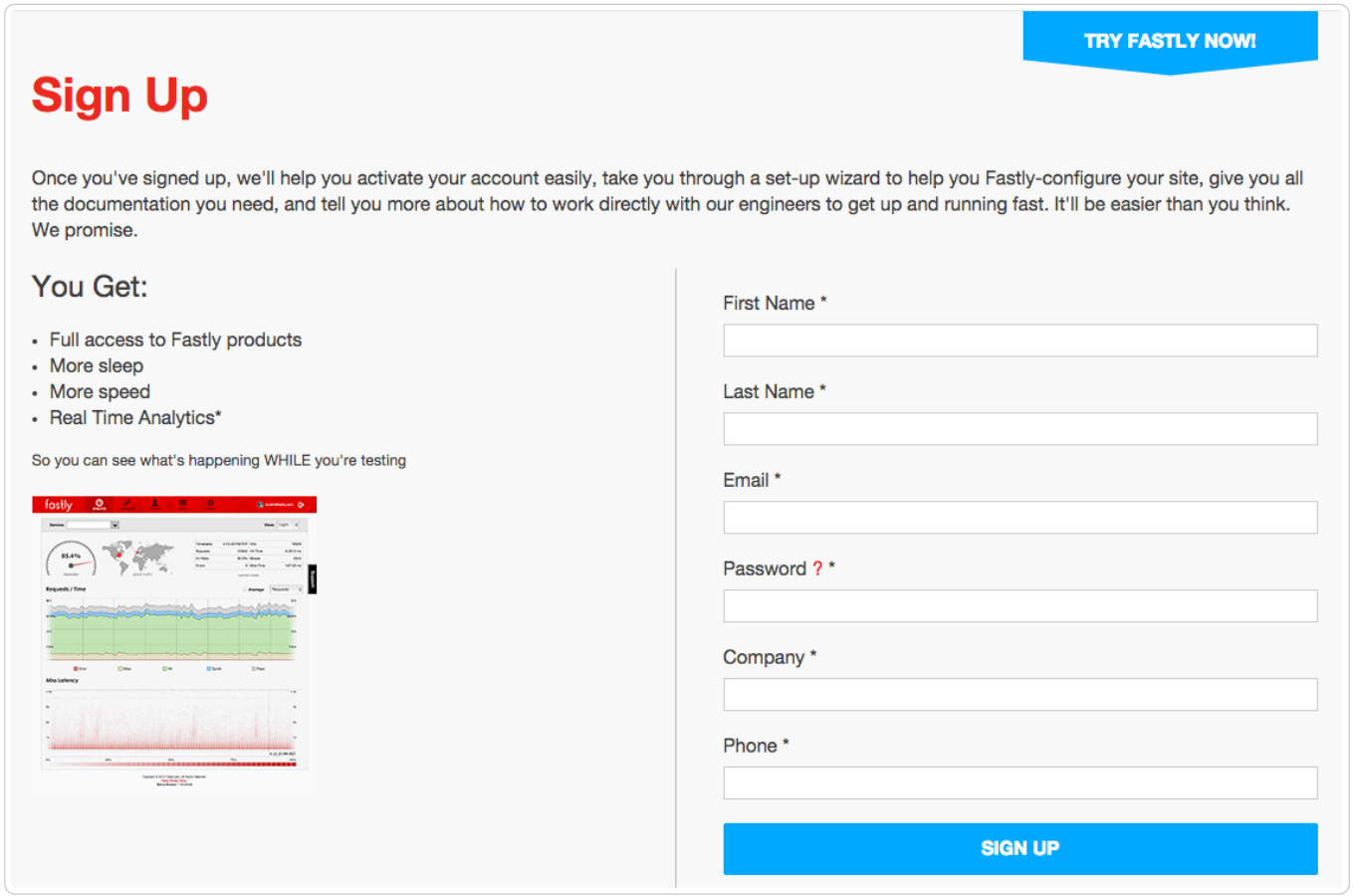
1. Using a web browser, navigate to the MX Toolbox SuperTool (http://mxtoolbox.com/SuperTool.aspx).



2. In the **Lookup anything** field, type the name of your website domain.
3. From the menu to the right of the field, select **CNAME Lookup**. The domain name and canonical name appear below the field.
4. Save this information to use when you sign up for Fastly services.

Sign up for Fastly CDN services

1. Using a web browser, navigate to the Fastly signup page (https://www.fastly.com/signup) and sign up for a Fastly account.



The confirmation page appears and the system sends a confirmation email to the address you specified during signup.

2. Verify your new Fastly account by clicking on the verification link sent to the email address used to sign up for Fastly service.

3. Log in to the Fastly application and complete your account configuration (/guides/basic-setup/sign-up-and-create-your-first-service) using the Fastly Quick Start guide that appears.

Complete the integration

Once you have completed the signup and configuration steps, please send an email to acquia@fastly.com (mailto:acquia@fastly.com) to complete the integration. Fastly will need to know:

- the email address associated with your new Fastly account and
- whether or not you require TLS (/guides/securing-communications/ordering-a-paid-tls-option) for your customer-facing domain.

Keep in mind that Fastly and Acquia both run a Varnish Cache (https://varnish-cache.org). In order to properly configure your service, Fastly needs to make a few modifications during the final setup process to ensure compatibility between the two. In this final setup process, Fastly runs a script (https://gist.github.com/vvuksan/66cc45e09812fbf90808) to automatically provision and configure your Fastly account. This script will:

- create an Acquia-specific service within your Fastly account,
- add your Acquia origin to the new service,
- add your end-user-facing domain to the new service, and
- add caching configurations to your service to optimize content delivery.

As soon as the configuration on Fastly's side is complete, you will receive email notification that you can change your CNAME records (/guides/basic-setup/cname-instructions-for-most-providers) to point at the appropriate corresponding Fastly endpoints. As soon as the CNAME process is complete, you'll be ready to start using Acquia Cloud as an origin for Fastly services.

§ Amazon S3 (/guides/integrations/amazon-s3)

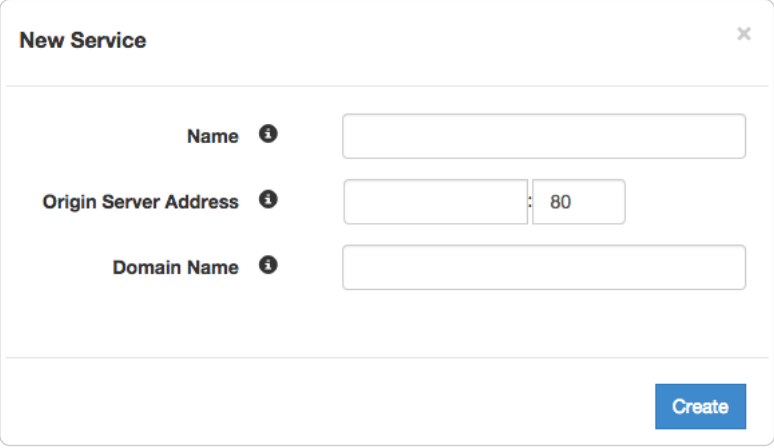
Amazon S3 (http://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html) can be used as an origin or as a private bucket.

Using Amazon S3 as an origin

To make your S3 data buckets available through Fastly follow the steps below.

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Click the green **New Service** button at the top right of the window. The New Service window appears.



4. Fill out the **New Service** window as follows:
   - In the **Name** field, type any descriptive name for your service.
   - In the **Origin Server Address** field, type `s3.amazonaws.com`. If you are using a non-standard S3 region (anything other than us-east), you must include that region in the server address field (e.g., `s3.us-west-2.amazonaws.com`). In the port field, you can type either `80` for HTTP

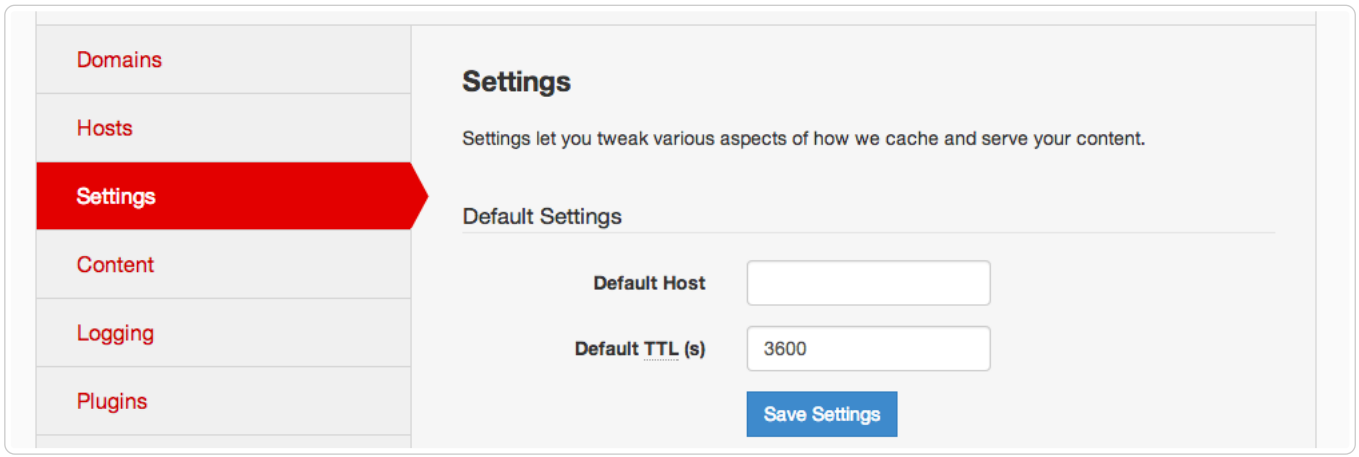
or `443` for HTTPS.

- In the **Domain Name** field, type the hostname you want to use as the URL (e.g., `cdn.example.com`).

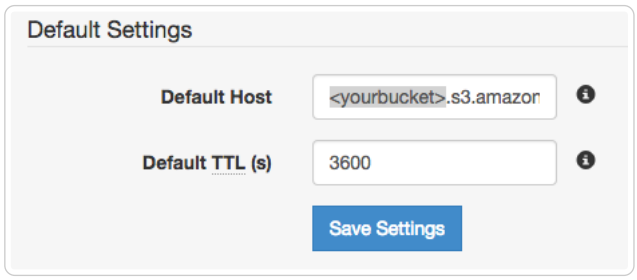
5. Click the **Create** button. A new service appears in the list of services available.

Now that the service is created, you will need to set the Default Host to `<yourbucket>.s3.amazonaws.com` by following the steps below:

1. Click the **configure** tab to access the control panel.
2. Select the new service you just created from the **Service** menu.
3. Click the blue **Configure** button to the right of the service name.
4. Click the **Settings** pane from the list on the left. The Settings controls appear.



5. In the **Default Host** field of the **Default Settings** area, type the hostname of your S3 bucket. For example, `<yourbucket>.s3.amazonaws.com`.



6. Click **Save Settings** and deploy your changes. Your service should be active within few seconds.

Testing your results

By default, we create DNS mapping called **yourdomain.global.prod.fastly.net**. In the example above, it would be `cdn.example.com.global.prod.fastly.net`. Please test, and if you are satisfied with the results, create a DNS alias for the domain name you specified (e.g., CNAME `cdn.example.com` to `global-nossl.fastly.net`).

Fastly will cache any content without an explicit `Cache-Control` header for 1 hour. You can verify whether you are sending any cache headers using `curl`. For example:

```
$ curl -I opscode-full-stack.s3.amazonaws.com

HTTP/1.1 200 OK
x-amz-id-2: ZpzRp7IWc6MJ8NtDEFGH12QBdk2CM1+RzVOngQbhMp2f2ZyalkFsZd4qPaLMkSlh
x-amz-request-id: ABV5032583242618
Date: Fri, 18 Mar 2012 17:15:38 GMT
Content-Type: application/xml
Transfer-Encoding: chunked
Server: AmazonS3
```

In this example no cache control headers are set so default TTL will be applied.

Enhanced cache control

If you need more control over how different types of assets are cached (e.g., Javascript files, images) media check out our Amazon S3 configuration (/guides/tutorials/cache-control-tutorial) in our Cache Control tutorial.

Using an Amazon S3 private bucket

To use an Amazon S3 private bucket with Fastly, follow the instructions below.

Before you begin

Be sure you've already made your S3 data buckets available to Fastly by pointing to the right S3 bucket and setting your origin to port 443. This needs to be done before authenticating.

Be sure you've got the AWS access key ID, AWS secret key ID, and AWS Bucket name on hand. The Amazon S3 Authorization header takes the following form:

```
Authorization: AWS `_AWSAccessKeyId_`:`_Signature_`
```

From your developer Amazon account you will need the following information:

1. The **AWS access key ID** and **AWS secret access key**. The AWS secret access key is issued when you register. If you do not have or remember your AWS secret access key, create a new AWS access key ID. The AWS secret access key will be displayed before disappearing again.
2. Your **AWS Bucket** name.

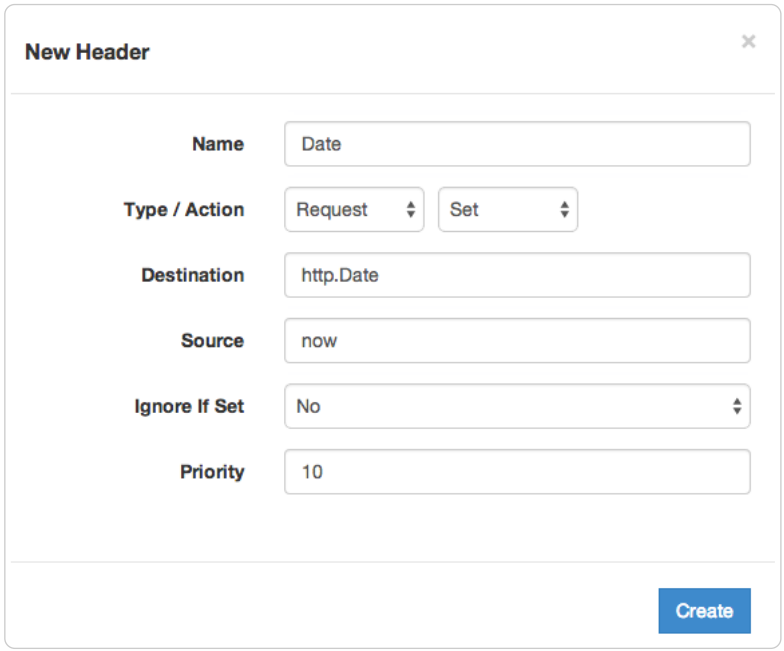
Setting up Fastly to use an Amazon S3 private bucket

In order to use an Amazon S3 private bucket with Fastly, create two headers (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses), a Date header (for use with the authorization Signature) and an Authorization header.

Create a Date header

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Content** pane from the list on the left.
6. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.



7. Fill out the **New Header** window as follows:

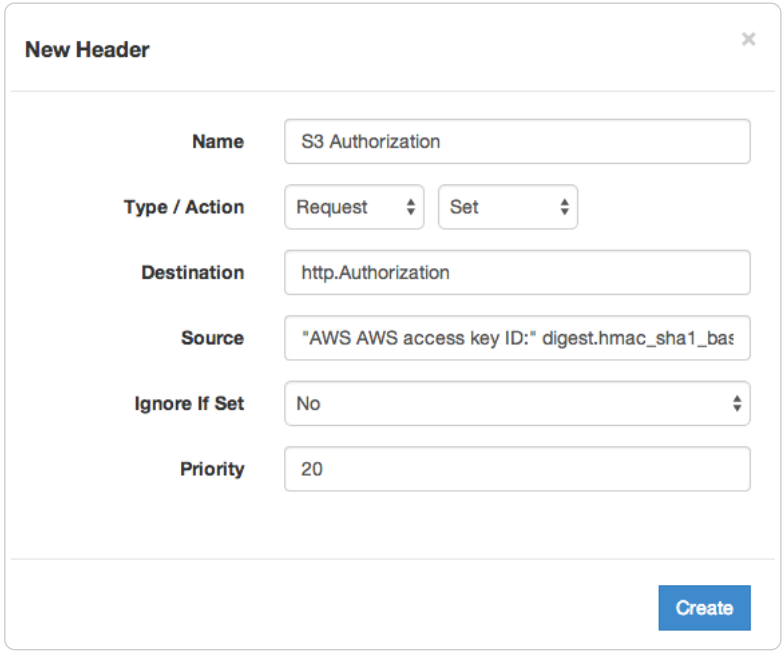
- In the **Name** field, type `Date`.
- From the **Type/Action** menus, select **Request** and **Set**.
- In the **Destination** field type `http.Date`.
- In the **Source** field type `now`.
- From the **Ignore if Set** menu, select **No**.
- In the **Priority** field, type `10`.

8. Click the **Create** button. A new Date header appears in the Headers area of the Content section. You will use this later within the Signature of the Authorization header.

Create an Authorization header

Next create the Authorization header with the specifications listed below.

1. Click the **New** button again to create another new header.



2. Fill out the **New Header** window as follows:
   - In the **Name** field, type `S3 Authorization`.
   - From the **Type/Action** menus, select **Request** and **Set**.
   - In the **Destination** field type `http.Authorization`.
   - From the **Ignore if Set** menu, select **No**.
   - In the **Priority** field, type `20`.
3. In the **Source** field, type the header authorization information using the following format:

```
"AWS <AWS access key ID>:" digest.hmac_sha1_base64("<AWS secret key ID>", if(req.request == "HEAD", "GET", req.request) LF LF
 LF req.http.Date LF "/<AWS Bucket name>" req.url.path)
```

   replacing `<AWS access key ID>`, `<AWS secret key ID>`, and `<AWS Bucket name>` with the information you gathered before you began. For example:

```
"AWS JKCAUEFV2ONFFOFMSSLA:" digest.hmac_sha1_base64("P2WPSu68Bfl89j72vT+bXYZB7SjlOwhT4whqt27", if(req.request == "HEAD", "GE
T", req.request) LF LF LF req.http.Date LF "/test123" req.url.path)
```

4. Click the **Create** button. A new Authorization header appears in the Headers area of the Content section.

A detailed look at the Source field

So what's going on in the Source field of the Authorization header? Here's the basic format:

`AWS<Access Key><Signature Function><key><message>`

It tells us the following:

Element	Description
`AWS`	A constant placed before the access key. It's always AWS.
`access key`	The access key ID from your Amazon developer's account. We used `JKCAUEFV2ONFFOFMSSLA` in this example.
`signature function`	The algorithm used to validate the key and message of the signature. We used `digest.hmac_sha1_base64(<key>, <message>)` in this example.
`key`	The secret key ID from your Amazon developer's account. We used `P2WPSu68BfI89j72vT+bXYZB7SjIOwhT4whqt27` in this example.
`message`	The UTF-8 encoding of the StringToSign. See the table below for a break down of each portion of the message.

The message that's part of the Source field in the Authorization header takes on this basic format:

`<HTTP-verb></n><Content-MD5>/n<Content-Type></n><Date></n><CanonicalizedAmzHeader></n><CanonicalizedResource>`

It tells us the following:

Element	Description
`HTTP-verb`	The REST verb. We use `req.request` in this example. We rewrite HEAD to GET because Varnish does this internally before sending requests to origin.
`/n`	A newline indicator constant. It's always /n.
`Content-MD5`	The content-md5 header value, used as a message integrity check. It's often left blank. We use `LF` (line feed) in this example.
`Content-Type`	The content-type header value, used to specify the MIME-type. It's often left blank. We use `LF` in this example.
`Date`	The date and time stamp. We use `req.http.Date` (which we created first as a separate header in the steps above).
`CanonicalizedAmzHeader`	The x-amz headers, which customize your S3 implementation. It's often left blank. We use `LF` in this example.
`CanonicalizedResource`	Your Amazon private bucket name. We use `"/test123"` in this example.

Following redirects to S3 objects and caching S3 responses

With custom VCL, Fastly can follow redirects to S3 objects and cache the s3 response as well as the 301 or 302 response separately.

IMPORTANT: The ability to upload custom VCL code (/guides/vcl/uploading-custom-vcl) is disabled by default when you sign up. To enable this ability for your account, contact support@fastly.com (mailto:support@fastly.com) and request it.

Once the ability to upload custom VCL has been enabled, be sure to read our "How do I mix and match Fastly VCL with custom VCL? (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl)" instructions. It's important to include the entire VCL boilerplate if you do not intend to override the Fastly default settings.

To configure Fastly to follow redirects to S3 objects, insert the following VCL snippets in your custom VCL:

Within vcl_recv

```
sub vcl_recv {

  if (req.http.redir != "true") {
    set req.backend = Main_Origin;
  } else {
    set req.backend = s3_backend;
    set req.http.host = "s3.amazonaws.com";
  }

#FASTLY recv

  if (req.request != "HEAD" && req.request != "GET" && req.request != "FASTLYPURGE") {
    return(pass);
  }

  return(lookup);

}
```

Within vcl_deliver

```
sub vcl_deliver {

  if (resp.status == 302 || resp.status == 301) {
    set req.http.redir = "true";
    set req.url = regsub(resp.http.Location, "http://s3.amazonaws.com/(.*)$", "/\1");
    set req.http.Fastly-Force-Shield = "yes";
    restart;
  }

#FASTLY deliver

  return(deliver);
}
```

Be sure to set the `Main_Origin` and `s3_backend` to the actual name of your backends in the service to which you're applying these redirects. You can find the exact names by reviewing your VCL; simply click on the VCL button at the top of the page while viewing the service.

Once you added these VCL snippets to your custom VCL, upload the VCL file and then activate the new version of your service to apply the changes.

---

§ Google Cloud Storage (/guides/integrations/google-cloud-storage)

Google Cloud Storage (https://cloud.google.com/storage/) (GCS) can be used as an origin server with your Fastly services once you set up and configure your GCS account and link it to a Fastly service. It can also be configured to use private content. This speeds up your content delivery and reduces your origin's workload and response times with the dedicated links between Google and Fastly's POPs.

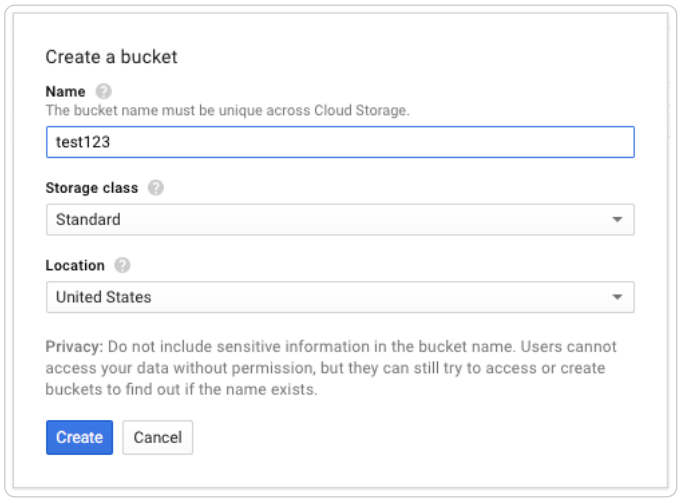
★ **TIP:** Fastly offers a Cloud Accelerator integration discount to any customer who uses GCS. Take advantage of this discount by emailing salesgcp@fastly.com (mailto:salesgcp@fastly.com) once you've completed the steps below. If you already have GCS configured as your origin and would like to take advantage of this discount, contact support (mailto:support@fastly.com).

Using GCS as an origin server

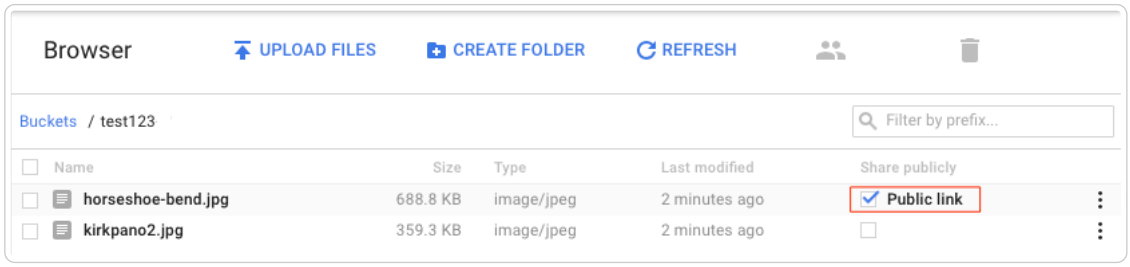
To make your GCS data available through Fastly follow the steps below.

Set up and configure your GCS account

1. Sign up for Google Cloud Storage (https://cloud.google.com/storage/docs/signup) and start the basic setup.
2. Create a bucket (https://cloud.google.com/storage/docs/getting-started-console) to store your origin's data and remember the name. You'll need the bucket name to connect your GCS account to your Fastly service.



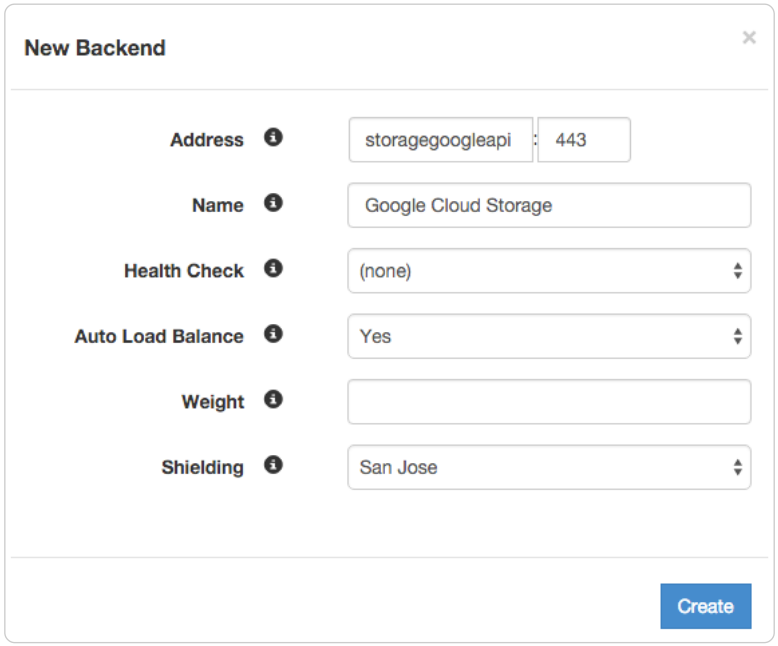
3. Add a file to the bucket and then make the file public.



Create a new origin in your Fastly service for your GCS account

Link your GCS account to a Fastly service following the steps below.

1. Log in to the Fastly application.
2. Create a new service (/guides/basic-setup/working-with-services) if you don't already have one set up.
3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Hosts** pane from the list on the left.
6. In the **Backends** area click the **New** button to create a new backend. The New Backend window appears.



7. Fill out the **New Backend** window as follows:
   - In the **Address** field, type the address of your secure server (for example, `origin.example.com`).
   - In the **Port** field type `443`.
   - In the **Name** field, type the name of your server (for example, `Google Cloud Storage`).
   - Leave the **Health Check**, **Auto Load Balance**, and **Weight** controls set to their default values.
   - From the **Shielding** menu, select an available interconnect location from the list of shielding locations. See our information on interconnect locations below for more details.
8. Click the **Create** button. The server appears in the Backends area.

Interconnect locations

The following interconnects allow you to establish direct links with Google's edge network when you choose your shielding location. By selecting one of the locations listed below, you will be eligible to receive discounted pricing (https://cloud.google.com/interconnect/cdn-interconnect#pricing) from Google CDN Interconnect for traffic traveling from Google Cloud Platform to Fastly's network. Most customers select interconnects closest to their origin.

Interconnects exist in the following locations within North America:

- Atlanta

- Chicago
- Dallas-Forth Worth
- Los Angeles
- New York City
- San Jose
- Washington DC

Interconnects outside of North America exist in:

- Amsterdam
- Frankfurt
- Hong Kong
- London
- Singapore
- Tokyo

Review our caveats of shielding (/guides/performance-tuning/shielding#caveats-of-shielding) and select an interconnect accordingly.

Set the Cache-Control header for your GCS bucket

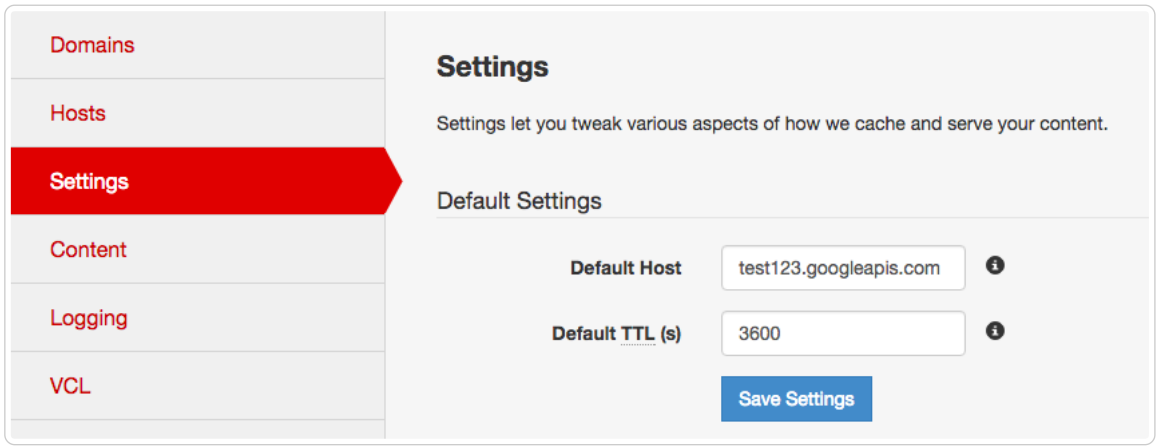
GCS performs its own caching, which may complicate efforts to purge cache with the Fastly application. To avoid potential problems, we recommend using the gsutil (https://cloud.google.com/storage/docs/gsutil_install) command line utility to set the Cache-Control header for one or more files in your GCS bucket:

```
gsutil setmeta -h "Cache-Control: max-age=0, s-maxage=86400" gs://bucket/*.html
```

Replace `bucket` in the example above with your GCS bucket's name. Note that `max-age` should instruct GCS to cache your content for zero seconds, and Fastly to cache your content for one day. See Google's setmeta docs (https://cloud.google.com/storage/docs/gsutil/commands/setmeta) for more information.

Set the default host for your service to your GCS bucket

1. Click **Settings** from the list on the left. The Settings controls appear.



2. In the **Default Host** field of the **Default Settings** area, type the name of the default host for this service.

   The name you type should match the name of the bucket you created in your GCS account and will take the format `<your bucket name>.storage.googleapis.com`. In this example, our bucket name is `test123`, so our **Default Host** name would be `test123.storage.googleapis.com`.

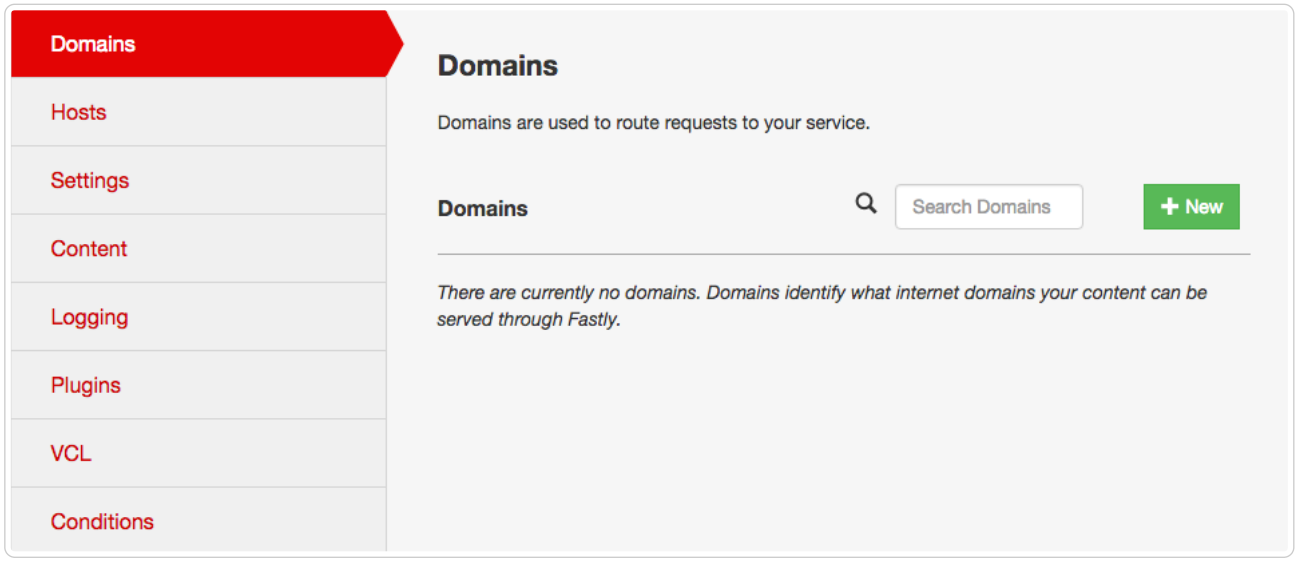
3. Decide how to change the default TTL for your GCS bucket, if at all, keeping the following in mind:

   - Your GCS account controls the default TTL for your GCS content. GCS currently sets the default TTL to 3600 seconds. Changing the default TTL via the **Default TTL (s)** field will not override the default setting in your GCS account.
   - To override the default TTL set by GCS from within the Fastly application, create a new cache setting (/guides/performance-tuning/controlling-caching) and enter the TTL there.
   - To override the default TTL in GCS, download the gsutil tool (https://cloud.google.com/storage/docs/gsutil_install) and then change the cache-control headers (https://cloud.google.com/storage/docs/gsutil/addlhelp/WorkingWithObjectMetadata#cache-control) to delete the

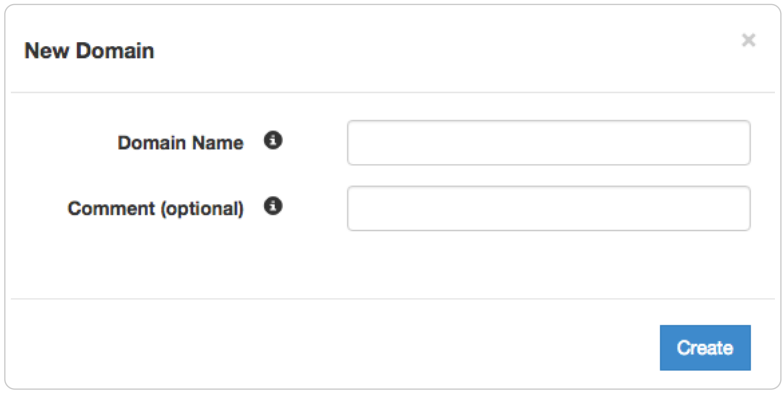
default TTL or change it to an appropriate setting.

Create new domains for GCS to respond to

1. Click **Domains** from the list on the left. The Domains controls appear.



2. Click the **New** button in the **Domains** area. The New Domain window appears.



3. In the **Domain Name** field, type the name users will type in their browsers to access your site, then click the **Create** button to create the new domain.
4. Because GCS responds to different hostnames than your Fastly service, create a second domain by following the domain creation steps immediately above.
5. In the **Domain Name** field of the second domain you create, type the same value as the default host you created earlier (e.g., `<your bucket name>.storage.googleapis.com`).

   Shielding POPs need this additional domain so they can route requests correctly. (See Caveats of Shielding (/guides/performance-tuning/shielding#caveats-of-shielding) for more information.)
6. Deploy the new origins by activating (/guides/basic-setup/working-with-services#deactivating-and-reactivating-services) the new version of your service.

Once you have deployed, you can use `http://<domain>.global.prod.fastly.net/<filename>` to access the files you uploaded.

Using GCS with private objects

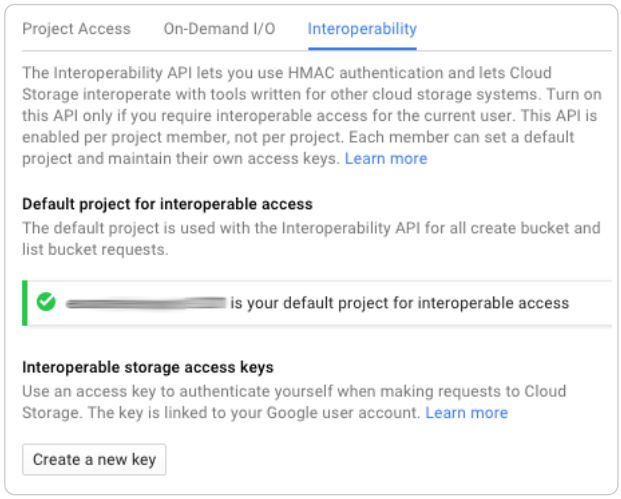
To use Fastly with GCS private objects, be sure you've already made your GCS data available to Fastly by pointing to the right GCS bucket, then follow the steps below.

Setting up interoperable access

By default, GCS authenticates requests using OAuth2, which Fastly does not support. To access private objects on GCS, your project must have HMAC authentication (https://cloud.google.com/storage/docs/gsutil/addlhelp/CredentialTypesSupportingVariousUseCases#supported-credential-types) enabled and interoperable storage access keys (an "Access Key" and "Secret" pair) created. Do this by following the steps below.

1. Open the Google Cloud Platform console and select the appropriate project.
2. Click **Settings**. The Settings appear with the Project Access controls highlighted.
3. Click the **Interoperability** tab. The Interoperability API access controls appear.
4. If you have not set up interoperability before, click **Enable interoperability access**.
5. Click **Make** `<PROJECT-ID>` **your default project** for interoperable access.

   If that project already serves as the default project, that information appears instead.



6. Click **Create a new key**. An access key and secret code appear.

7. Save the access key and secret code that appear. You'll need these later when you're creating an authorization header.

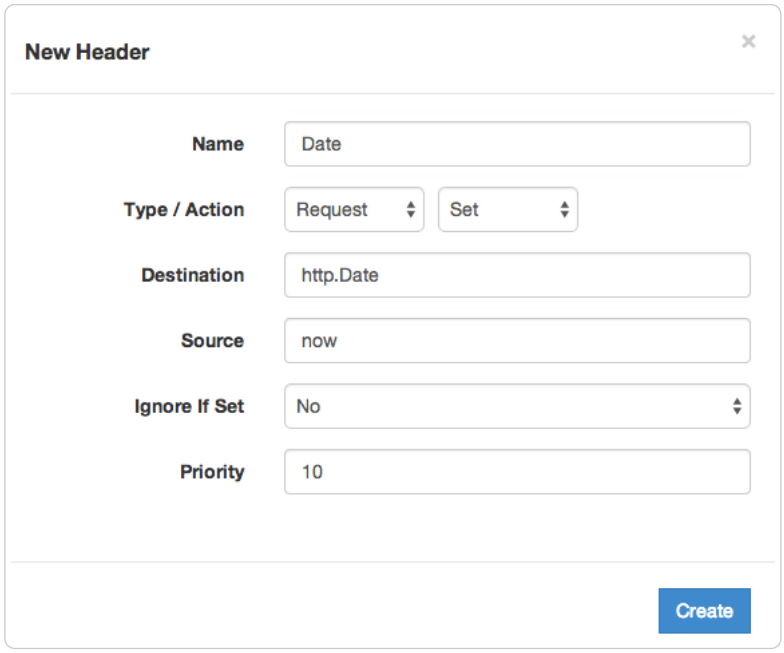
Setting up Fastly to use GCS private content

To use GCS private content with Fastly, create two headers (/guides/basic-configuration/adding-or-modifying-headers-on-http-requests-and-responses), a Date header (required Authorization Signature) and an Authorization header.

Create a Date header

1. Log in to the Fastly application.
2. Click the **configure** tab to access the control panel.

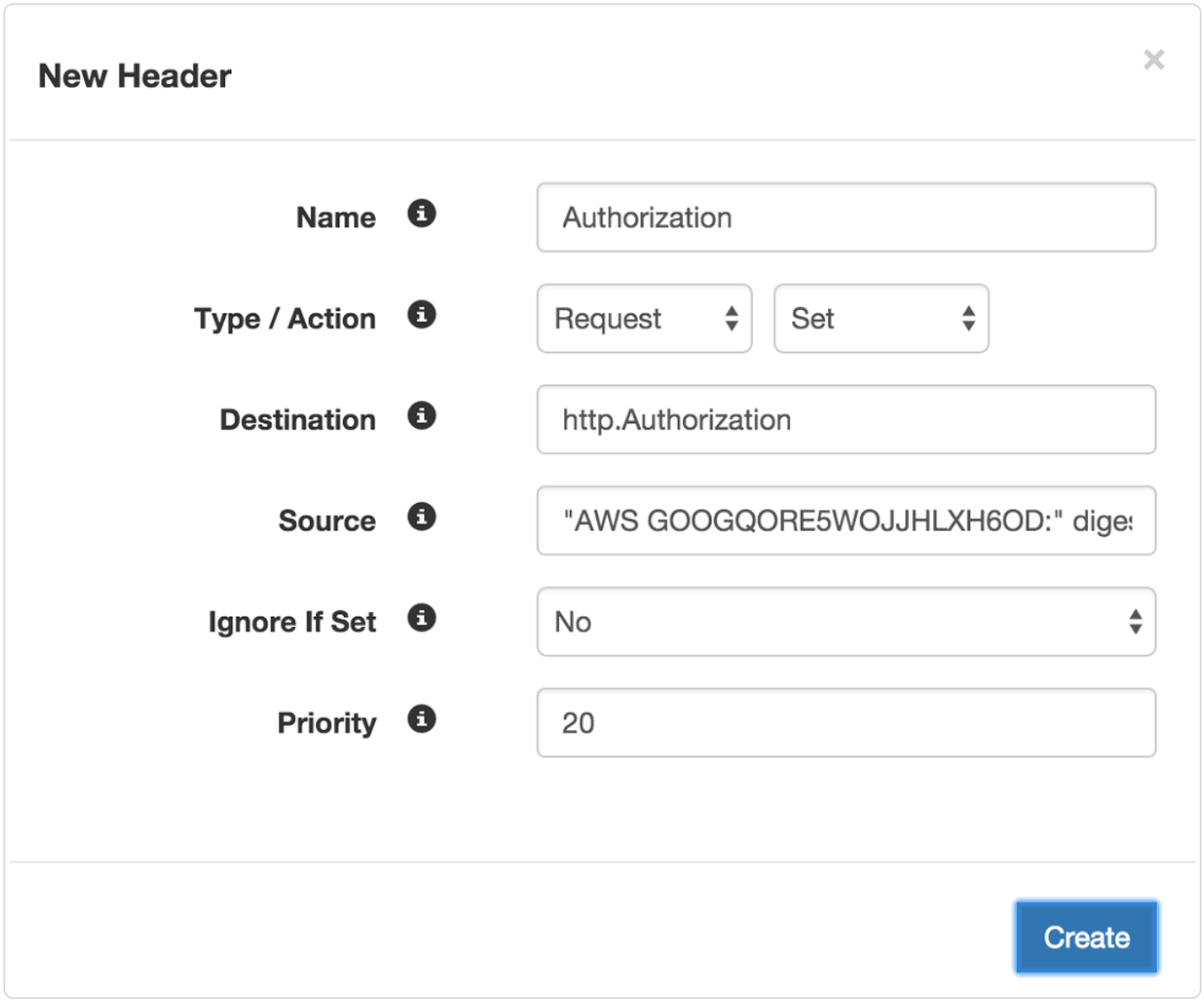
3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click **Content** from the section list on the left.
6. In the **Headers** area, click the **New** button to create a new header. The New Header window appears.



7. Fill out the **New Header** window as follows:
   - In the **Name** field, type `Date`.
   - From the **Type/Action** menus, select **Request** and **Set**.
   - In the **Destination** field, type `http.Date`.
   - In the **Source** field type `now`.
   - From the **Ignore if Set** menu, select **No**.
   - In the **Priority** field, type `10`.
8. Click the **Create** button. A new Date header appears in the Headers area of the Content section. You will use this later within the Signature of the Authorization header.

Create an Authorization header

1. Click the **New** button again to create another new header.



2. Fill out the **New Header** window as follows:
   - In the **Name** field, type `Authorization`.
   - From the **Type/Action** menus, select **Request** and **Set**.
   - In the **Destination** field, type `http.Authorization`.
   - From the **Ignore if Set** menu, select **No**.
   - In the **Priority** field, type `20`.
3. In the **Source** field, type the header authorization information using the following format:

```
  "AWS <access key>:" digest.hmac_sha1_base64("<GCS secret>", req.request LF LF LF req.http.Date LF "/<GCS bucket name>" req.u
rl.path)
```

   replacing `<access key>`, `<GCS secret>`, and `<GCS bucket name>` with the information you gathered before you began. For example:

```
  "AWS GOOGQORE5WOJJHLXH6OD:" digest.hmac_sha1_base64("oQb0hdmaxFOc5UmC6F833Cde0+ghRSgsr7CCnX62", req.request LF LF LF req.htt
p.Date LF "/test123" req.url.path)
```

4. Click the **Create** button. A new Authorization header appears in the Headers area of the Content section.

A detailed look at the Source field

So what's going on in the Source field of the Authorization header? Here's the basic format:

`AWS<access key><signature function><key><message>`

It tells us the following:

Element	Description

`AWS`	A constant placed before the access key. It's always AWS.
`access key`	The access key ID from your GCS developer's account. We used `GOOGQORE5WOJJHLXH6OD` in this example.
`signature function`	The algorithm used to validate the key and message of the signature. We used `digest.hmac_sha1_base64(<key>, <message>)` in this example.
`key`	The secret key ID from your GCS developer's account. We used `oQb0hdmaxFOc5UmC6F833Cde0+ghRSgsr7CCnX62` in this example.
`message`	The UTF-8 encoding of the StringToSign. See the table below for a break down of each portion of the message.

The message that's part of the Source field in the Authorization header takes on this basic format:

`<HTTP-verb><\n><Content-MD5>\n<Content-Type><\n><Date><\n><CanonicalExtensionHeaders><\n><CanonicalizedResource>`

It tells us the following:

Element	Description
`HTTP-verb`	The REST verb. We use `req.request` in this example.
`\n`	A newline indicator constant. It's always \n.
`Content-MD5`	The content-md5 header value, used as a message integrity check. It's often left blank. We use `LF` (line feed) in this example.
`Content-Type`	The content-type header value, used to specify the MIME-type. It's often left blank. We use `LF` in this example.
`Date`	The date and time stamp. We use `req.http.Date` (which we created first as a separate header in the steps above).
`CanonicalExtensionHeaders`	The x-amz- or x-goog- headers, which customize your GCS implementation. It's often left blank. We use `LF` in this example.
`CanonicalizedResource`	Your GCS resource path name. We're concatenating GCS bucket name `"/test123"` with object path `req.url.path` in this example.

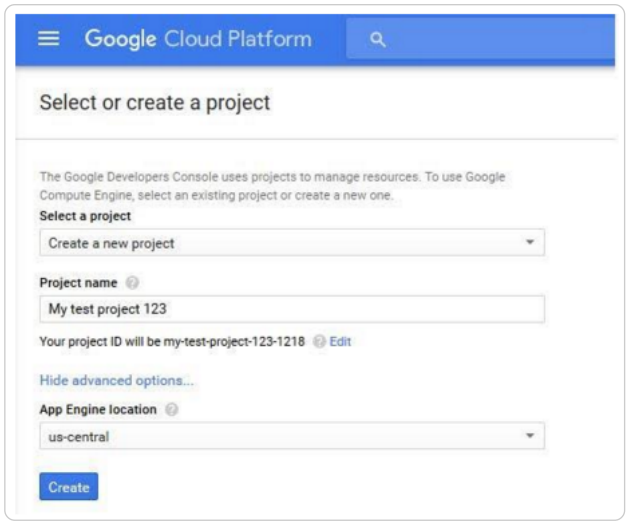
§ Google Compute Engine (/guides/integrations/google-compute-engine)

Google Compute Engine (GCE) lets you create and run a virtual machine (VM) on the Google infrastructure. The VM can be used as an origin server with your Fastly service once you set up and configure your VM instance and link your instance to a Fastly service.

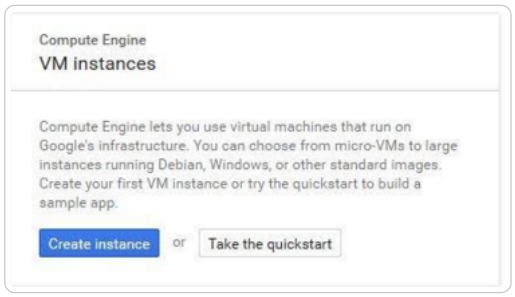
★ **TIP:** Fastly offers a Cloud Accelerator integration discount to any customer who uses GCE. Take advantage of this discount by emailing salesgcp@fastly.com (mailto:salesgcp@fastly.com) once you've completed the steps below. If you already have GCE configured as your origin and would like to take advantage of this discount, contact support (mailto:support@fastly.com).

Set up and create your GCE instance

1. Sign up for Google Compute Engine (https://cloud.google.com/compute/docs/) and start the basic set up. If you are already signed up and at your dashboard, click the **Get started** link in the Try Compute Engine area.
2. Create or select a project to hold your origin's data.

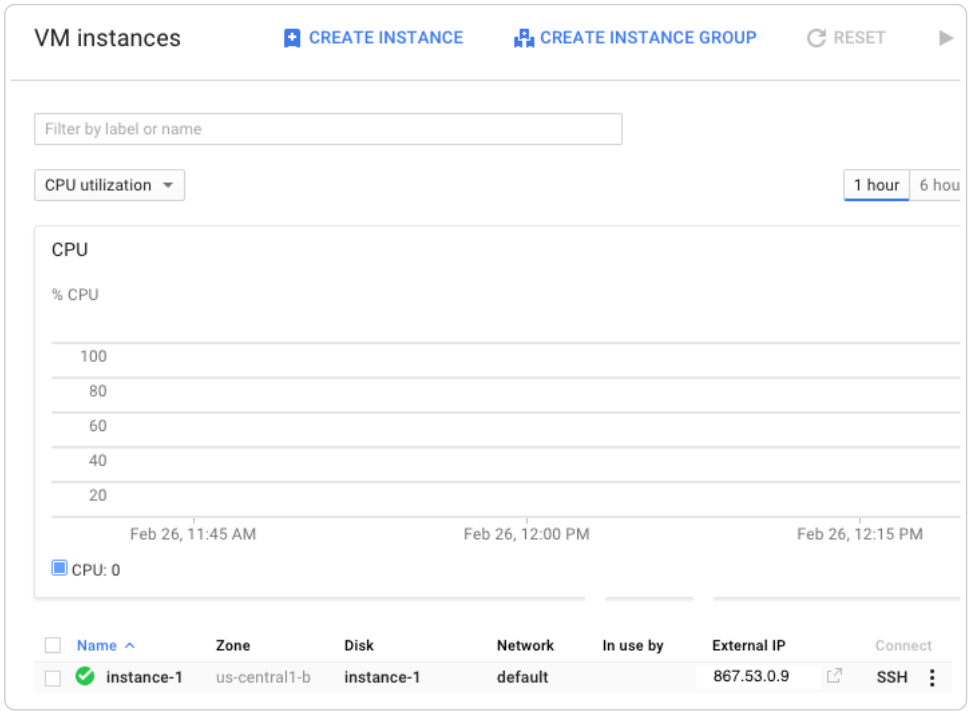


3. Click **Create instance** to set up your virtual machine. You can set up your instance using either Windows or Linux.



4. Fill in the necessary fields and click **Create**.

   When making your firewall selection, select either **Allow HTTPS traffic** (port 443) or **Allow HTTP traffic** (port 80); you will use one of those ports when you create your new origin in your Fastly service. If you select HTTPS traffic, you need to configure the VM to respond on port 443 with a valid TLS certificate.



5. Make note of the following informatin for when you create your new origin in your Fastly service:

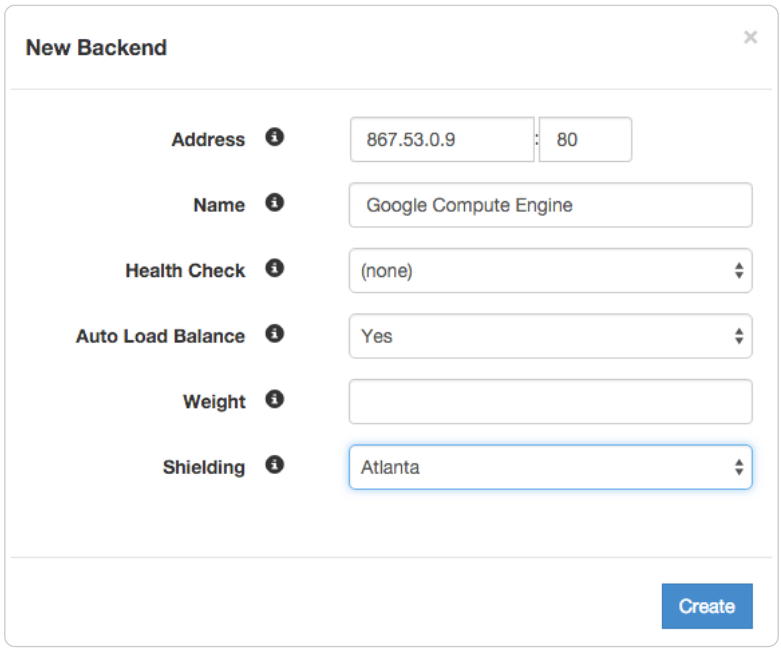
   - the instance's IP address (located in the External IP column at the bottom of the page). You'll use this in the **Address** field when you create your new origin.
   - the zone you are using (located in the Zone column at the bottom of the page). You'll use this to guide your selection of an appropriate

shielding location for your origin.

Create a new origin in your Fastly service for your GCE account

Link your GCE account to a Fastly service following the steps below.

1. Log in to the Fastly application.
2. Create a new service (/guides/basic-setup/working-with-services) if you don't already have one set up.
3. Select the appropriate service from the **Service** menu.
4. Click the blue **Configure** button to the right of the service name.
5. Click the **Hosts** pane from the list on the left.
6. In the **Backends** area click the **New** button to create a new backend. The New Backend window appears.



7. Fill out the **New Backend** window as follows:
   - In the **Address** field, type the IP address of your server.
   - In the **Port** field, type `80` or `443`. This should match the port that you selected in the GCE interface.
   - In the **Name** field, type the name of your server (for example, `Google Compute Engine`).
   - Leave the **Health Check**, **Auto Load Balance**, and **Weight** controls set to their default values.
   - From the **Shielding** menu, select an available interconnect location from the list of shielding locations. See our information on interconnect locations below for more details.
8. Click the **Create** button. The server appears in the Backends area.

In this example no cache control headers are set so default TTL will be applied.

Interconnect locations

The following interconnects allow you to establish direct links with Google's edge network when you choose your shielding location. By selecting one of the locations listed below, you will be eligible to receive discounted pricing (https://cloud.google.com/interconnect/cdn-interconnect#pricing) from Google CDN Interconnect for traffic traveling from Google Cloud Platform to Fastly's network. Most customers select interconnects closest to their origin.

Interconnects exist in the following locations within North America:

- Atlanta
- Chicago
- Dallas-Forth Worth
- Los Angeles
- New York City
- San Jose

- Washington DC

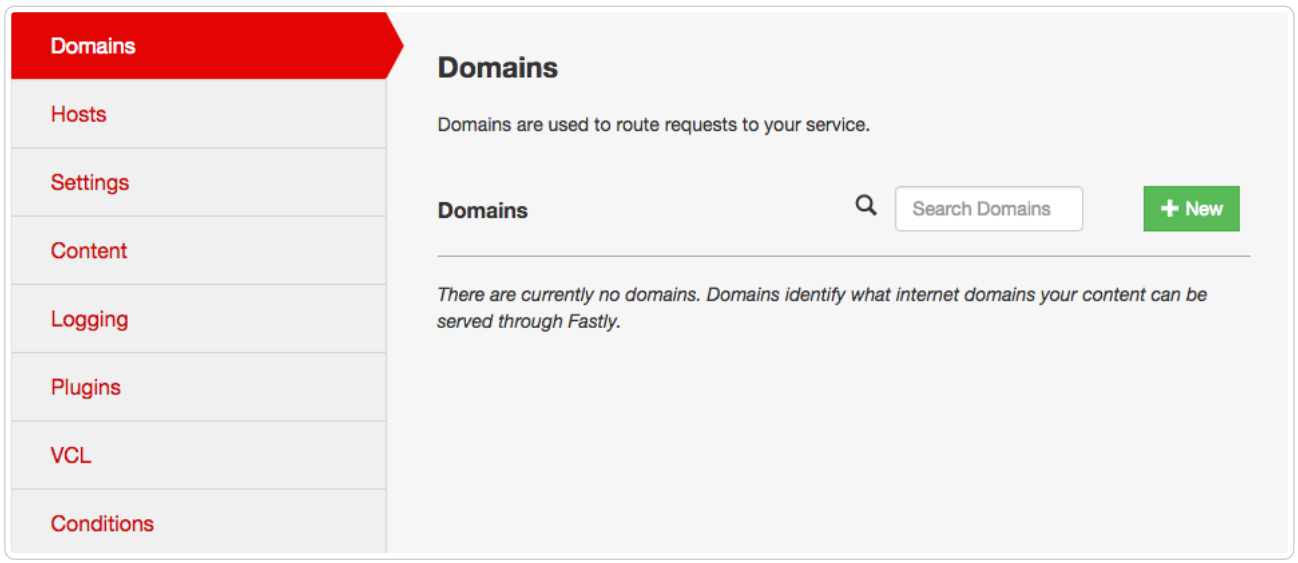
Interconnects outside of North America exist in:

- Amsterdam
- Frankfurt
- Hong Kong
- London
- Singapore
- Tokyo

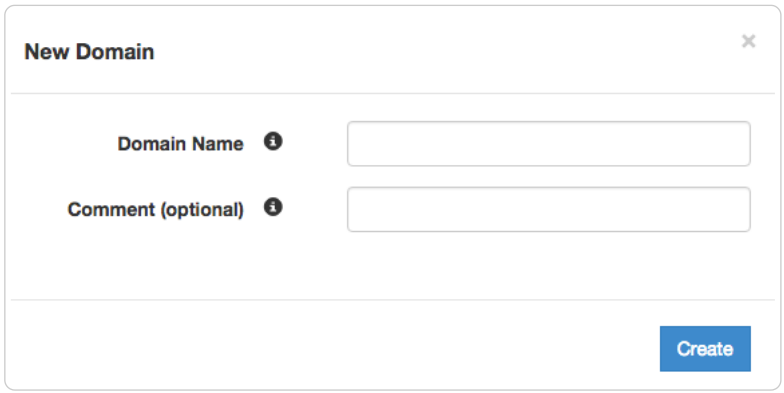
Review our caveats of shielding (/guides/performance-tuning/shielding#caveats-of-shielding) and select an interconnect accordingly.

Create new domains for GCE to respond to

1. Click **Domains** from the list on the left. The Domains controls appear.



2. Click the **New** button in the **Domains** area. The New Domain window appears.



3. In the **Domain Name** field, type the name that users will type in their browsers to access your site.
4. Click the **Create** button to create the new domain.
5. Deploy the new origins by activating (/guides/basic-setup/working-with-services#deactivating-and-reactivating-services) the new version of your service.

Test your results and create a CNAME record

You can now test your results (/guides/basic-configuration/testing-setup-before-changing-domains). In the example above, it would be `www.example.com.global-nossl.fastly.net`. After you test and you're satisfied with the results, create a CNAME record (/guides/basic-setup/adding-cname-records) for your domain (e.g., `www.example.com`) pointing to `global-nossl.fastly.net`.

§ Zencoder (/guides/integrations/zencoder)

Zencoder provides HLS output that Fastly can cache and deliver. To start, you'll need to create a Zencoder job that outputs live HLS to S3. Zencoder provides documentation on this setup process (https://app.zencoder.com/docs/guides/getting-started/live-s3-quickstart).

Next, configure a new Fastly service using the S3 bucket that you are using with Zencoder as an origin. You can refer to our documentation on setting up S3 as an origin (/guides/integrations/amazon-s3).

Once this is complete, you can serve your HLS streams through Fastly as you would any other content.

• Guides (/guides/) > Online video streaming (/guides/online-video) > Live streaming (/guides/live-streaming/)

§ Configuration guidelines for live streaming (/guides/live-streaming/configuration-guidelines-for-live-streaming)

The Fastly network can deliver live streams for any HTTP streaming technology (/guides/about-fastly-services/about-fastlys-video-caching-service), archived or recorded, on any public or private cloud storage service. When configuring VCL (/guides/vcl/uploading-custom-vcl) to deliver live streams, we recommend following these guidelines:

- **Configure shielding.** Configure shielding (/guides/performance-tuning/shielding) by designating a specific shield POP for your origin to ensure live streams remain highly available within the Fastly network. If your setup includes primary and alternate origins (e.g., for high profile live streams), be sure to select a shield POP, close to each origin, one for each origin you define.
- **Set different caching TTLs.** Set manifest file TTLs to less than half of the video segment duration, typically 1-2 seconds for 5-second video segments. For long DVRs and live-to-VOD transitions, set segment TTLs longer on shields and shorter on edge POPs such that they are served from memory (that is, less than 3600s).
- **Configure Streaming Miss.** Configure Streaming Miss (/guides/performance-tuning/improving-caching-performance-with-large-files#streaming-miss) to reduce the time clients (players) must wait to begin downloading streams when Fastly's edge servers must fetch content from your origin.
- **Enable TCP optimizations.** Enable TCP tuning optimizations (e.g., CWND size) between cache servers and clients. For example, consider setting the CWND value higher, between 20-25. Enabling TCP optimizations allows clients to better estimate network bandwidth and thereby select the ideal rendition and bitrate (/guides/on-the-fly-packaging/adaptive-bitrate-playback-url-guidelines) for the best playback quality.
- **Take advantage of surrogate key purging.** All video segments and the manifest for a live stream can be purged using a single API call by using Fastly’s surrogate key feature (/guides/purging/getting-started-with-surrogate-keys).
- **Consider setting up failover (fallback) origins.** Consider configuring your VCL to allow your origins to failover (/guides/performance-tuning/failover-configuration) from high-profile primary streams to alternate streams in case of encoder failures or other issues (e.g., high resource utilization).
- **Manage live-to-VOD smoothly.** Most encoders generate a separate video manifest when making the same live stream available for VOD. If your VOD manifest has the same URL as the live one, purge the live stream video manifest or wait for the caches to invalidate (as they will be set with low TTLs). If your setup archives the live stream as progressive mp4s, consider delivering them using Fastly’s OTFP service (/guides/about-fastly-services/about-fastlys-onthefly-packaging-service).

Questions about live streaming setup and configuration? Contact our Customer Support (mailto:support@fastly.com) team any time.

NOTE: Wowza integrations. When configuring your Wowza origin server, be sure to select the Live HTTP Origin (https://www.wowza.com/forums/content.php?227-How-to-configure-a-live-stream-repeater) application type. If you select Live Edge, Wowza will always return a unique URL for manifest requests, resulting in extremely low cache hit.

• Guides (/guides/) > Online video streaming (/guides/online-video) > On-the-fly packaging (/guides/on-the-fly-packaging/)

§ Adaptive bitrate playback URL guidelines (/guides/on-the-fly-packaging/adaptive-bitrate-playback-url-guidelines)

Fastly's On-the-Fly Packaging (OTFP) service (/guides/about-fastly-services/about-fastlys-onthefly-packaging-service) supports any directory structure you might use to store different quality levels of a video. To construct adaptive bitrate (ABR) playback URLs for a video, make directory paths to that video unique. Ensure all the files associated with a particular video (e.g., quality levels, subtitles) exist under a single directory.

For example, say you had a video called Example Video. Assuming you had multiple quality levels and associated files for Example Video, the following directory structure would provide the best start to constructing ABR playback URLs:

Directory path example	Description
`/foo/bar/example-video/`	Base folder unique to this video
`/foo/bar/example-video/480p_30fps.mp4`	Quality level 480p with 30 frames per sec with audio
`/foo/bar/example-video/720p_30fps.mp4`	Quality level 720p with audio with 30 frames per sec with audio
`/foo/bar/example-video/720p_60fps.mp4`	Quality level 720p with audio with 60 frames per sec with audio
`/foo/bar/example-video/1080p_30fps.mp4`	Quality level 1080p with audio with 30 frames per sec with audio
`/foo/bar/example-video/1080p_60fps.mp4`	Quality level 1080p with audio with 60 frames per sec with audio
`/foo/bar/example-video/4k_30fps.mp4`	Quality level 4k with audio with 30 frames per sec with audio

With this directory structure, the ABR playback URL for all videos in the base directory would follow this template:

```
http://example.com/path/to/dir/<video_id>/<quality_file1_name_wo_ext>,<quality_file2_name_wo_ext>,...,<quality_fileN_name_wo_ext>/master.<f4m|m3u8|mpd>
```

For example, the ABR playback URLs for Example Video in every format would be:

Format	Example URL
HDS	`http://example.com/foo/bar/example-video/480p_30fps,720p_30fps,720p_60fps,1080p_30fps,1080p_60fps,4k_30fps/master.f4m`
HLS	`http://example.com/foo/bar/example-video/480p_30fps,720p_30fps,720p_60fps,1080p_30fps,1080p_60fps,4k_30fps/master.m3u8`
MPEG-DASH	`http://example.com/foo/bar/example-video/480p_30fps,720p_30fps,720p_60fps,1080p_30fps,1080p_60fps,4k_30fps/master.mpd`

You can reduce the duplication in ABR playback URLs separating out the repeated prefix and suffix info as follows:

`<filename_prefix><filename_variable><filename_suffix_wo_ext>.mp4`

and the template would change to one of the following:

```
http://example.com/path/to/dir/<video_id>/<filename_prefix><quality_file1_variable_name_wo_ext>,<quality_file2_variable_name_wo_ext>,...,<quality_fileN_variable_name_wo_ext>,<filename_suffix_wo_ext>/master.<f4m|m3u8|mpd>
```

```
http://example.com/path/to/dir/<video_id>/<filename_prefix><quality_file1_variable_name_wo_ext>,<quality_file2_variable_name_wo_ext>,...,<quality_fileN_variable_name_wo_ext>/master.<f4m|m3u8|mpd>
```

```
http://example.com/path/to/dir/<video_id>/<quality_file1_variable_name>,<quality_file2_variable_name>,...,<quality_fileN_variable_name>,<filename_suffix_wo_ext>/master.<f4m|m3u8|mpd>
```

IMPORTANT: To use token authentication (/guides/tutorials/enabling-token-authentication) with ABR manifest URLs, special modifications must be made using custom VCL (/guides/vcl/uploading-custom-vcl). Contact support@fastly.com (mailto:support@fastly.com) for assistance.

§ Collecting OTFP metrics (/guides/on-the-fly-packaging/collecting-oftp-metrics)

Fastly allows you to collect and process On-the-Fly Packaging (OTFP) service (/guides/about-fastly-services/about-fastlys-onthefly-packaging-service) metrics for analysis using a combination of custom VCL updates and specific log streaming settings. Once you've set up OTFP metrics collection through remote log streaming you can use any of a number of third-party and open source software options to aggregate your logging data for visualization and further analysis.

Upload custom VCL

1. If you have not already done so, request the ability to upload custom VCL (/guides/vcl/uploading-custom-vcl) code be enabled for your account.
2. Review the caveats of mixing and matching Fastly VCL with custom VCL (/guides/vcl/mixing-and-matching-fastly-vcl-with-custom-vcl).

3. Add the following custom VCL to your Fastly VCL:

```
sub vcl_deliver {
  # Identify Request type
  if (req.url.ext ~ "m3u8|ts|aac|webvtt") {
    set resp.http.Otfp-Format = "HLS";
  } else if (req.url.ext ~ "mpd|m4s") {
    set resp.http.Otfp-Format = "DASH";
  } else {
    set resp.http.Otfp-Format = "OTHER";
  }

  # Extract name-value pairs Otfp Info herder
  if (resp.http.X-Fastly-Otfp-Info) {
    set resp.http.Otfp-SS = regsub(resp.http.X-Fastly-Otfp-Info, ".*ss=(\S+).*", "\1");
    set resp.http.Otfp-SL = regsub(resp.http.X-Fastly-Otfp-Info, ".*sl=(\S+).*", "\1");
    set resp.http.Otfp-VL = regsub(resp.http.X-Fastly-Otfp-Info, ".*vl=(\S+).*", "\1");

    # Resolution (rs name-value) not available for audio-only segments
    if (resp.http.X-Fastly-Otfp-Info ~ ".*rs=(\S+).*") {
      set resp.http.Otfp-RS = re.group.1;
    } else {
      set resp.http.Otfp-RS = "-";
    }
  }
#FASTLY deliver

  return(deliver);
}
```

Create a logging endpoint

Follow the instructions to set up remote log streaming (/guides/streaming-logs/setting-up-remote-log-streaming) for your account and when creating your specific logging endpoint, set the **Format String** field to the following:

```
%h now.sec %r %>s %b resp.http.Otfp-Format resp.http.Otfp-SS resp.http.Otfp-SL resp.http.Otfp-VL resp.http.Otfp-RS
```

Control log file timing with a logging endpoint condition

To avoid excess log files, consider attaching a condition to the logging endpoint so logs are only sent when video segments are requested so that logging specifically exclude those files sent from Fastly's Origin Shield.

1. Click the gear icon next to the logging service you enabled when setting up remote log streaming (/guides/streaming-logs/setting-up-remote-log-streaming) and select **Conditions**. The New Condition window appears.
2. Fill out the **New Condition** window as follows:
   - In the **Name** field, type a human-readable name for the condition.
   - In the **Apply If** field, type `resp.http.X-Fastly-Otfp-Info && !req.http.Fastly-FF`.
   - Leave the default value set in the **Priority** field.
3. Click **Create**.

Analyze logging data

In addition to any Varnish variable (https://www.varnish-cache.org/docs/2.1/reference/vcl.html#variables), and a variety of Fastly's extensions to VCL (/guides/vcl/guide-to-vcl#fastly%27s-vcl-extensions), log files include the following video-specific fields:

- `ss` - video segment start presentation time in seconds
- `sl` - video segment duration in seconds
- `vl` - video duration in seconds
- `rs` - video track display resolution in pixels

You can use these fields to run queries for analysis and use what you discover to refine your video delivery settings.