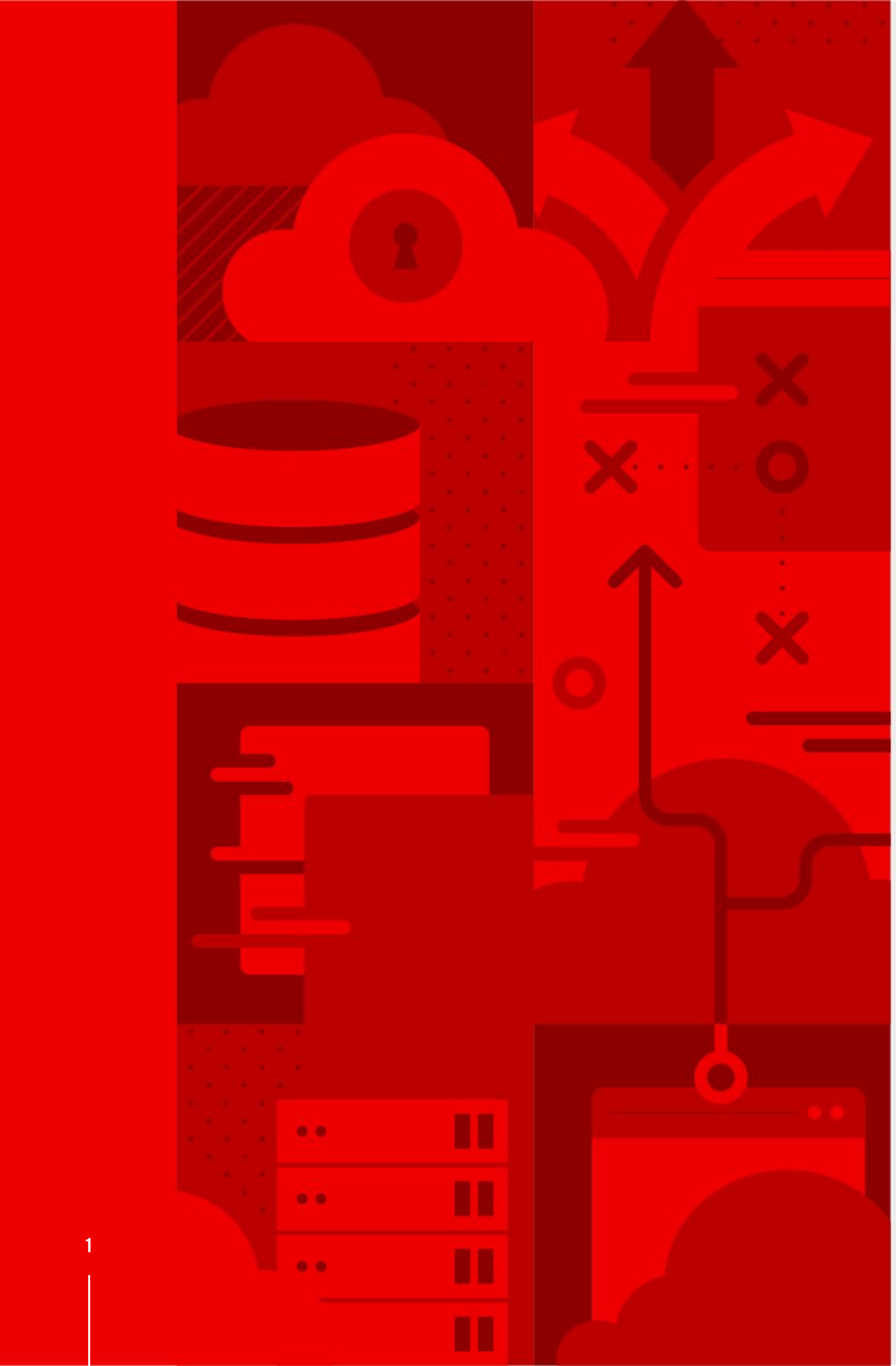


Red Hat 3scale Policy System

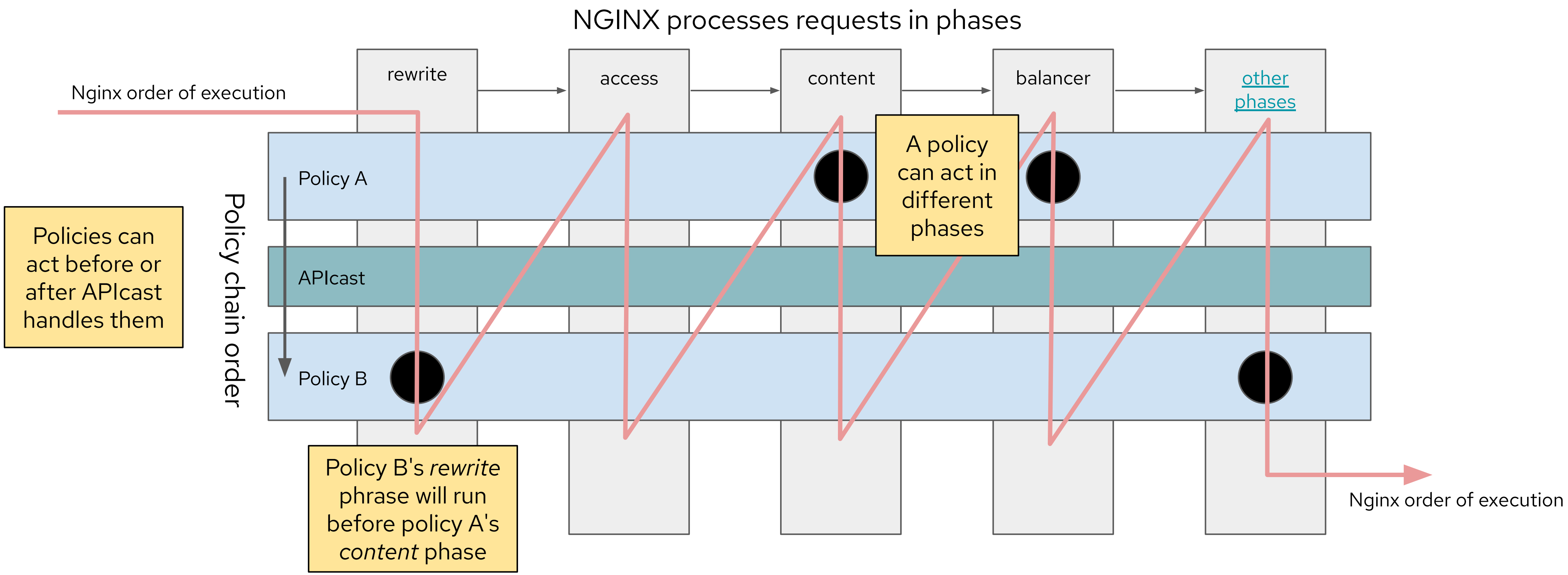
One at a time

Phillip Hagerman
Senior Technical Account Manager

April 5, 2022

APICast policy order and Nginx phases

Policies are processed per-defined order for each phase



Policies can act before or after APIcast handles them

A policy can act in different phases

Policy B's *rewrite* phrase will run before policy A's *content* phase

NGINX Phases

- rewrite
- access
- content
- balancer
- header_Filter
- body_filter
- post_action
- log

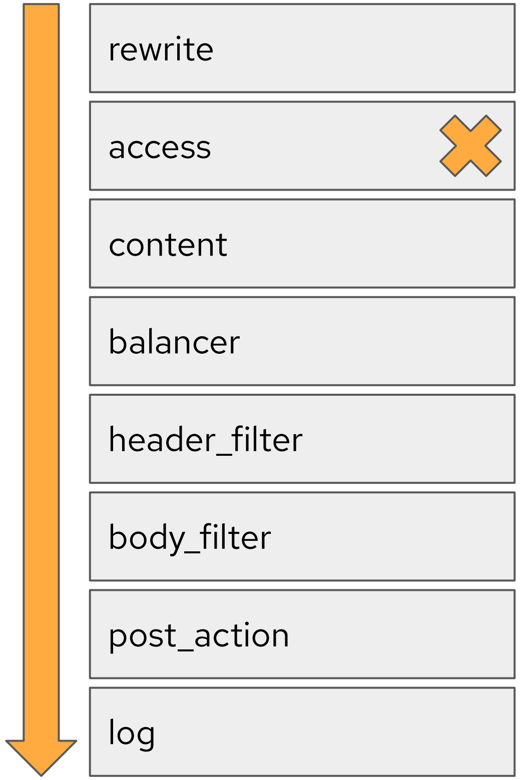
3scale Auth Caching

Controls how cache authorizations from the backend are stored

Possible use case: The authorization backend has frequent downtimes, or you need to require an authorization on every call.

Has multiple modes of operation:

- Strict - only caches successful calls, the cache is invalidated on denied/failed calls
- Resilient - calls are authorized according the state of the last request made, failed calls do not invalidate the cache
- None - caching is disabled
- Allow - caches authorized and denied calls. This means that any call authorized prior to the backed going down will continue to be authorized. Understand these implications, they only make sense in very specific situations



rewrite

access ✗

content

balancer

header_filter

body_filter

post_action

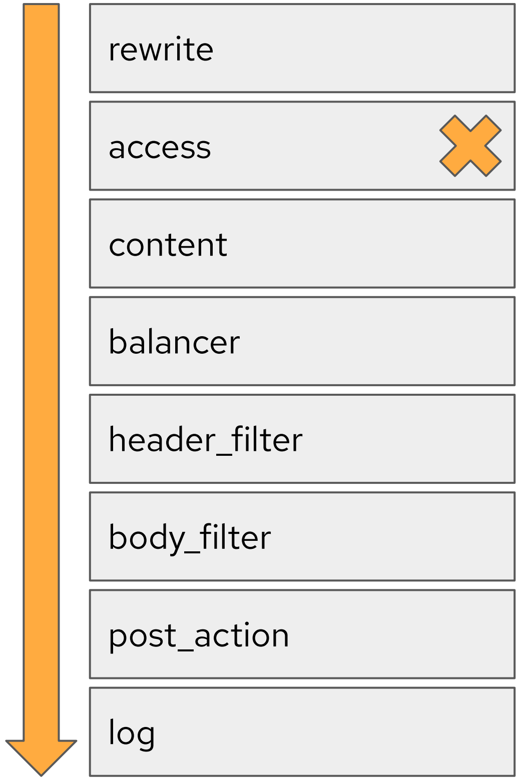
log

3scale Batcher

Caches authorizations from 3scale backend and batches reports

Possible use case: a high-load API needs better response times than it is currently getting, but there is less of a need for accurate rate-limiting.

Uses cached credentials if they are present and then holds the counter of calls before writing it back to analytics in batches rather than individually.



rewrite

access ✗

content

balancer

header_filter

body_filter

post_action

log

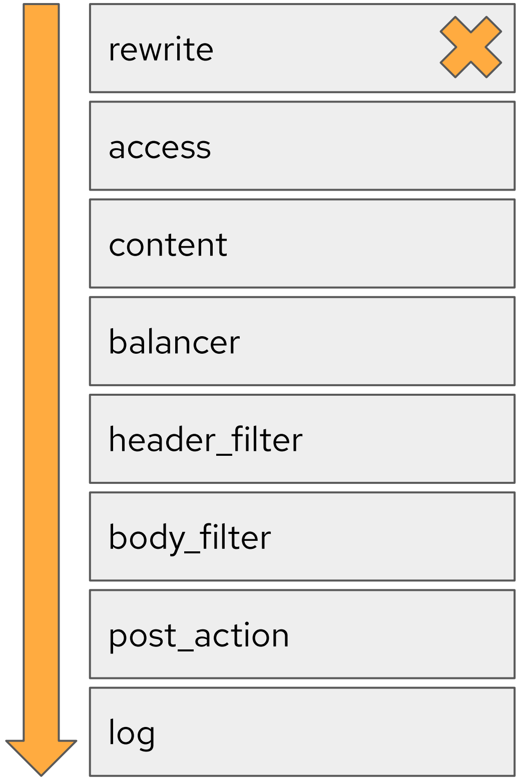
3scale Referrer

Caches authorizations from 3scale and batches reports

Possible use case: Empower developer to restrict access to their applications using this API product.

3scale supports the referrer filter which can be used to whitelist IP addresses or domain names from where an application can access the API

Must be enabled in the API applications Usage Rules Settings



Anonymous Access

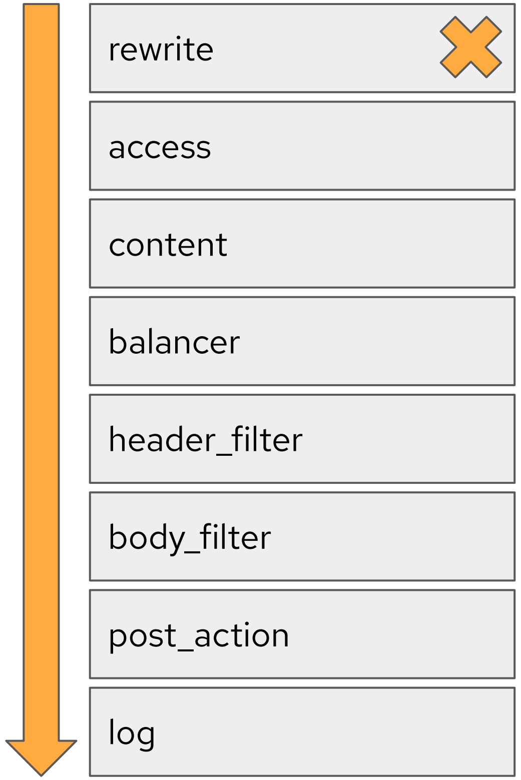
Exposes a service without authentication

Possible use case: Legacy applications that cannot be adapted to send the authentication parameters.

When credentials are not provided on the request the values in the policy are used with the request.

Only supports API Key or App Id/App Key options

Note: this policy must be placed above the APIcast Policy



rewrite ✗

access

content

balancer

header_filter

body_filter

post_action

log

Camel Service

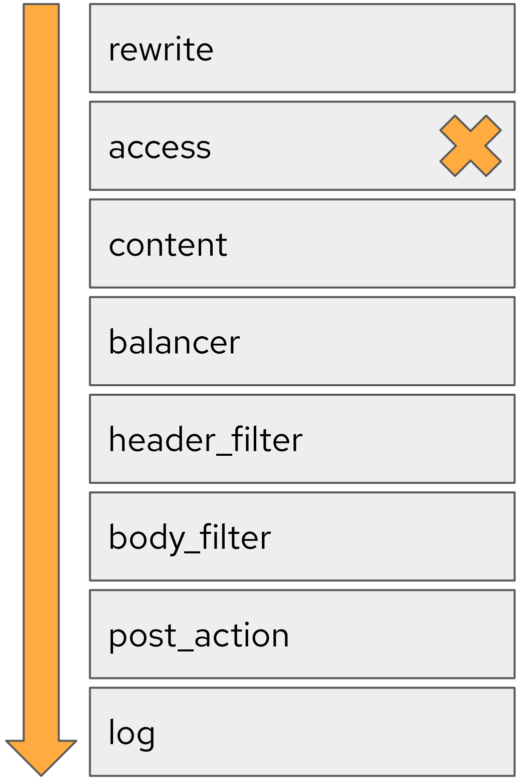
Define an *HTTP proxy* where traffic is sent over and Apache Camel proxy

Possible use case: There is a need for balancing a central request across multiple backends or do other transformations on the content provided.

Came works as a reverse proxy in this case.

1. APIcast sends the traffic to Camel
2. *Camel does its own actions on the request*
3. Camel sends the traffic to the backend
4. Camel receives the response from the backend
5. *Camel does its own actions on the response*
6. Camel sends the response back to APIcast

Note: this policy does not support authentication, you must use the Header Modification policy.



Conditional Policy

Still in *Technology Preview*

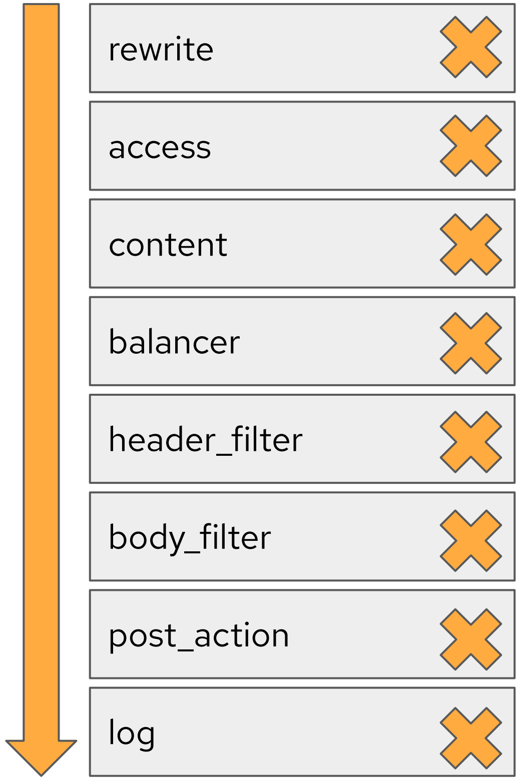
Possible use case: Provide different policy processing actions for a request that is a `POST` method, versus actions that are not `POST` actions.

Different than other APIcast policies as it you can define additional policy chains for the same APIcast product.

This policy is evaluated for every *nginx* phase. When the condition is true. Then the defined policies for that phase are run.

See ngx_variables.lua for available variables to use, or use the Liquid Context Debug Policy.

Note: The gui is incomplete at this time and policies must be updated via the "Proxy Policies Chain Update" 3scale Admin API.



Content Caching

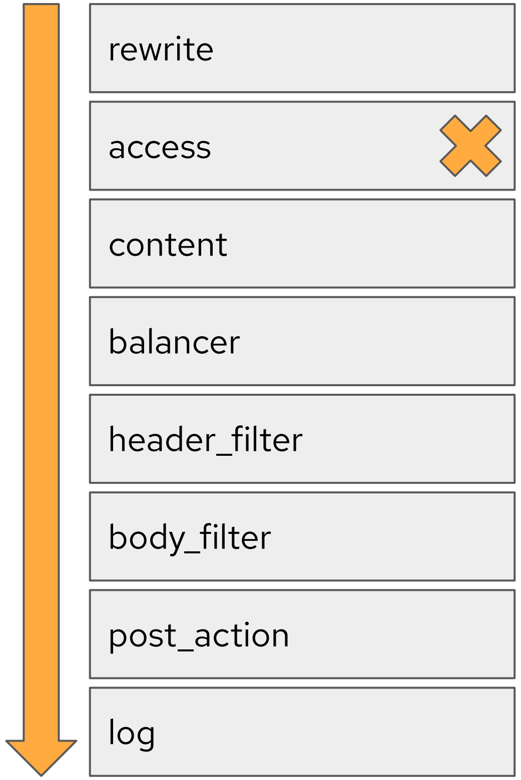
Option to enable content caching on responses

Allows for conditional caching of client requests.

If a rule matches, the cache will be enabled and the execution of the chain will stop, so sorting is very important on this policy.

If the upstream requires a different behavior regarding timeouts, use the Cache-Control header

Note: This can only be used for the client request where upstream responses cannot be used in the policy.



rewrite

access

content

balancer

header_filter

body_filter

post_action

log

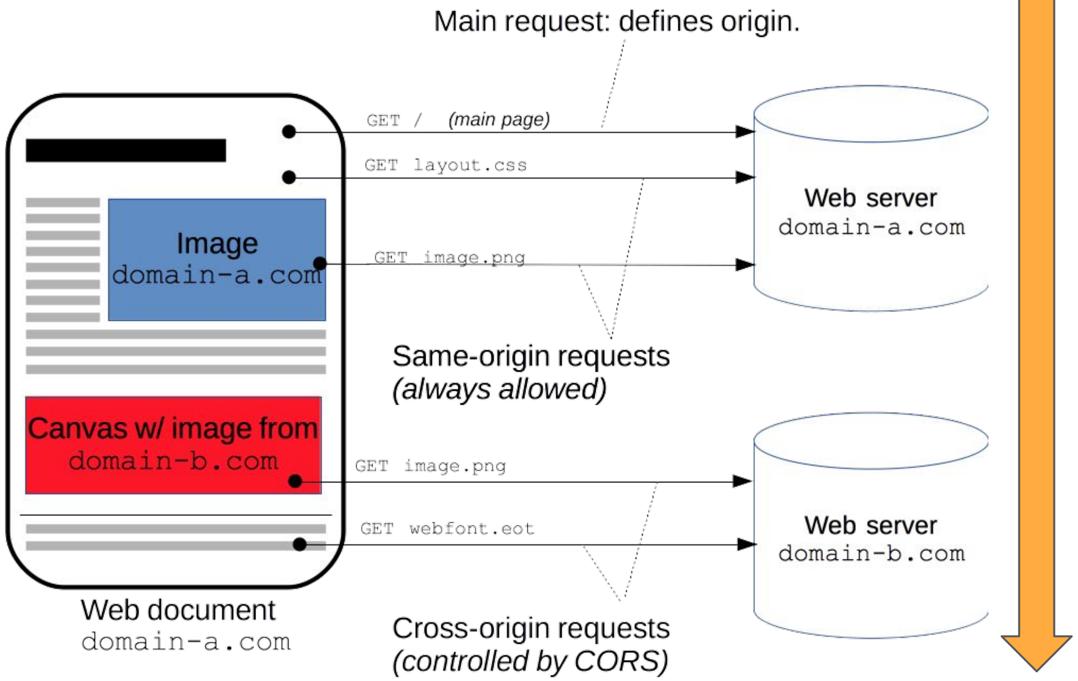
CORS Request Handling

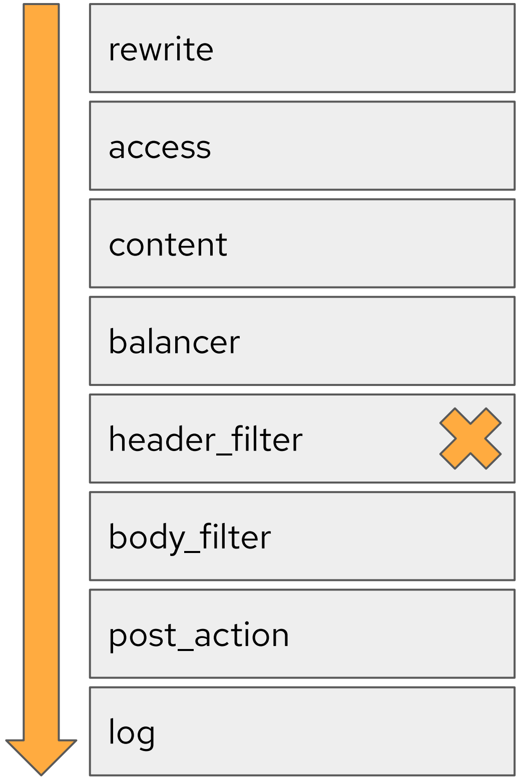
Enables CORS (Cross Origin Resource Sharing) request handling

Cross Origin Resource Sharing (CORS) controls the following items for what additional host information will be allowed:

- Allowed headers
- Allowed methods
- Allowed origin headers
- Allowed credentials
- Max age

Note: this policy must be placed above the APIcast Policy when using it.





Custom Metrics

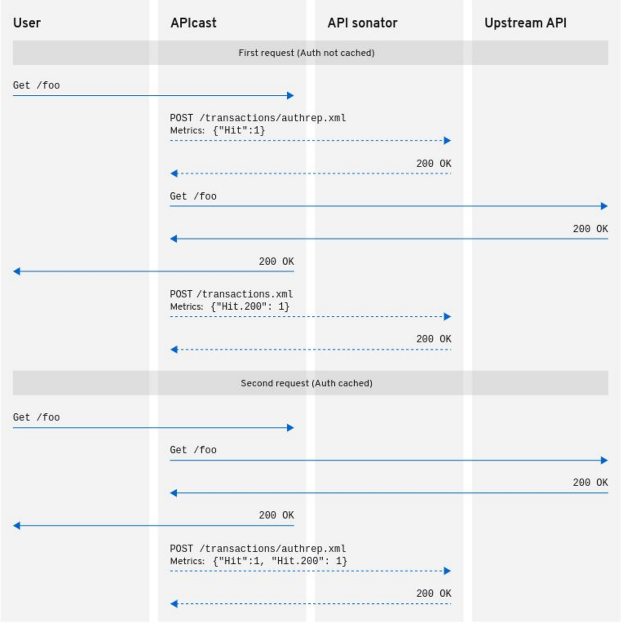
Custom metrics on Nginx post actions

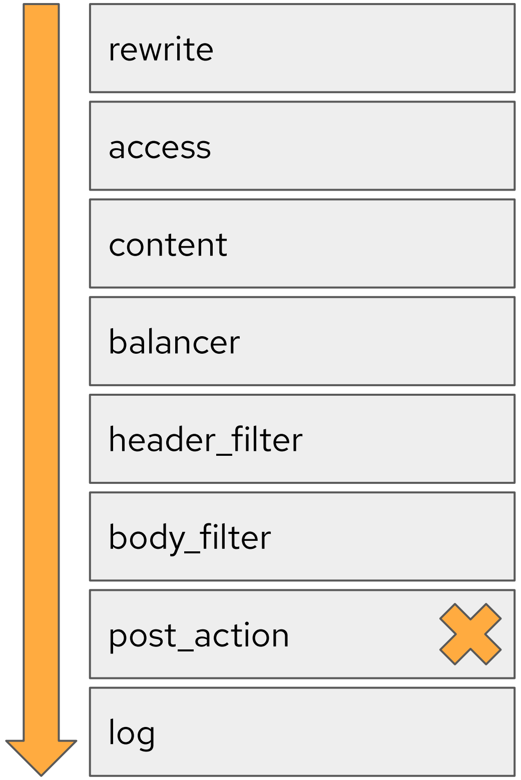
Possible use case: You have a need to log metrics for various HTTP status codes in addition to the standard 'hits' metric.

This policy can increment metrics after the response sent by the upstream based on available liquid variables.

If the authentication happens before the request, then a second call will be made for the custom metric

Note: this policy cannot be used with the batching policy, and the metrics must be already created before they will be incremented by this policy





rewrite

access

content

balancer

header_filter

body_filter

post_action

log

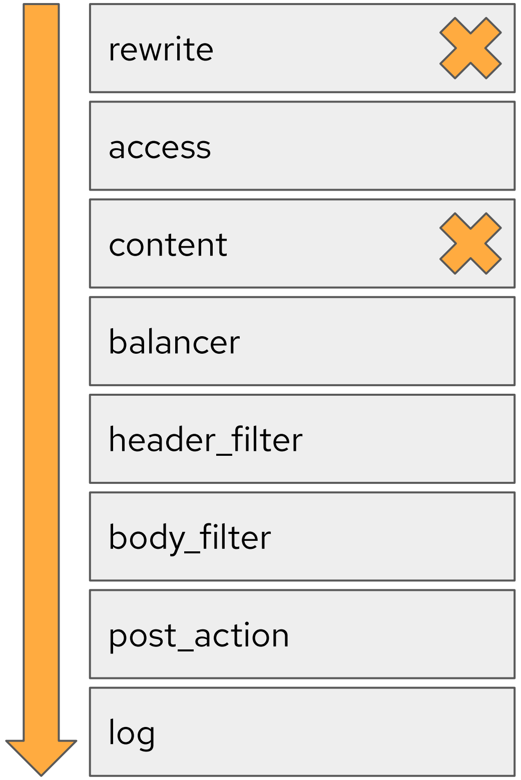
Echo

Prints the request back to the client and optionally sets a status code

Possible use case: This is useful for designing/prototyping.

This is a very simple policy and only takes 2 parameters, the HTTP code to return and the exit mode.

The exit mode determines if the request stops, or if only the rewrite phase is skipped.



rewrite ✗

access

content ✗

balancer

header_filter

body_filter

post_action

log

Edge Limiting

Adds rate-limiting

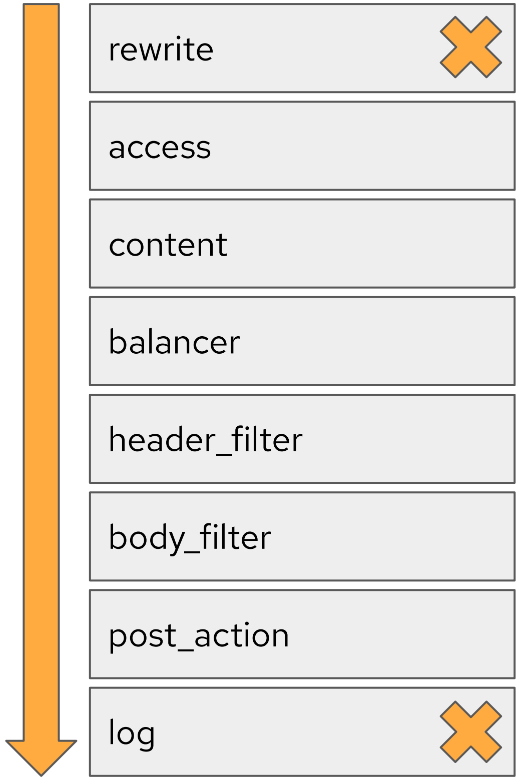
Possible use case: You need maintain firmer control over a specific request for APIs that are sensitive to high-volume traffic outside of application plan limits.

Supports the following types of limits:

- leaky_bucket_limiters
- fixed_window_limiters
- conection_limiters

They can be scoped by service or globally. Each limit has its own parameters.

You can specify the HTTP code and either exit the request or log the overage.



Header Modification

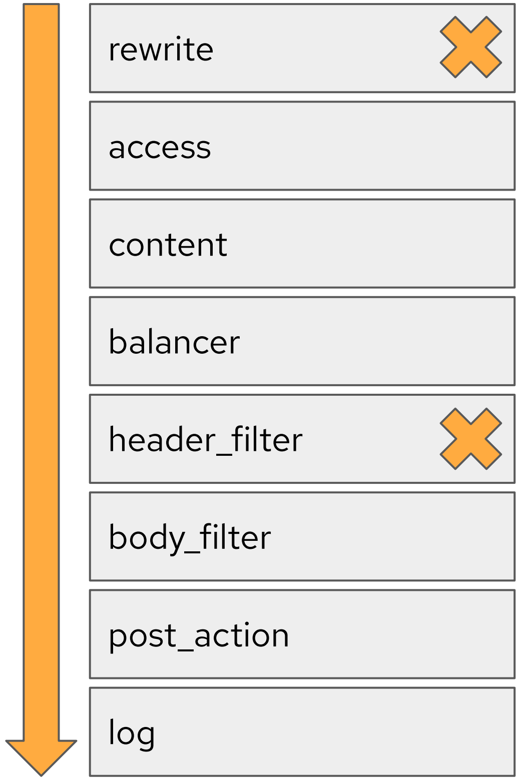
Allows including custom headers

Possible use case: You are using a Camel service to balance across multiple backends, and still need to provide authentication. The Header modification policy can inject the authentication headers.

You can inject custom headers using this policy.

You can also modify or delete existing headers.

It can modify headers in both the request and response.



- rewrite ✗
- access
- content
- balancer
- header_filter ✗
- body_filter
- post_action
- log

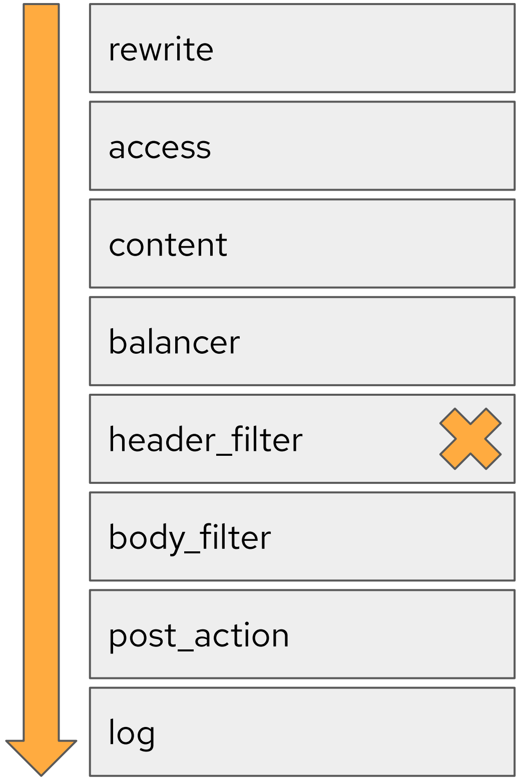
HTTP Status Code Overwrite

Modify the HTTP status code returned by the upstream

Possible use case: the backend returns a variety of HTTP codes that you want to obfuscate or simplify from your consumers

Allows you to remap the return codes of the backend API in a 1:1 relationship.

Can remap multiple codes in a single policy.



IP Check

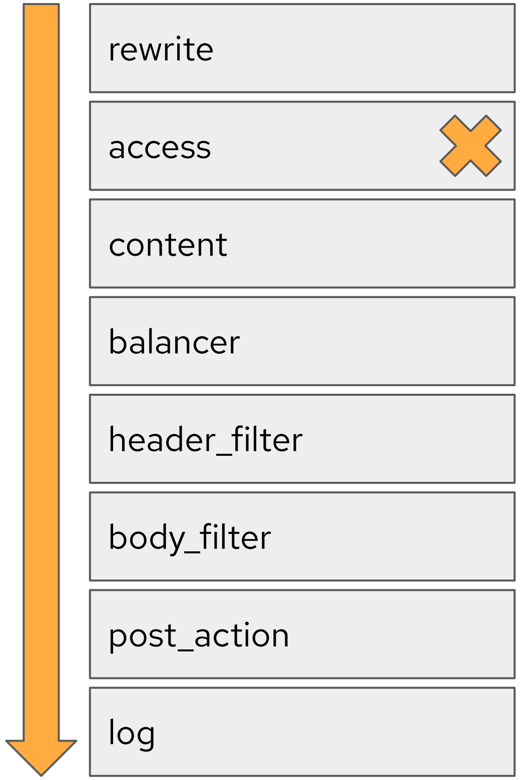
Accepts or denies a request based on the IP

Possible use case: You have an API that you need to protect from use by other systems. Use the IP Check to create an allow-list

Can pull the IP from the requester from multiple locations

- Last caller IP
- X-Forwarded-For header
- X-Real-IP header
- proxy_protocol_addr variable

Can use the list of IPs as a block list or an only-allow list and provide an error message. Single IPs (like 172.18.0.1) and CIDR ranges (like 172.18.0.0/16) can be used.



rewrite

access

content

balancer

header_filter

body_filter

post_action

log

JWT Claim Check

Allow or deny traffic based on a JWT claim

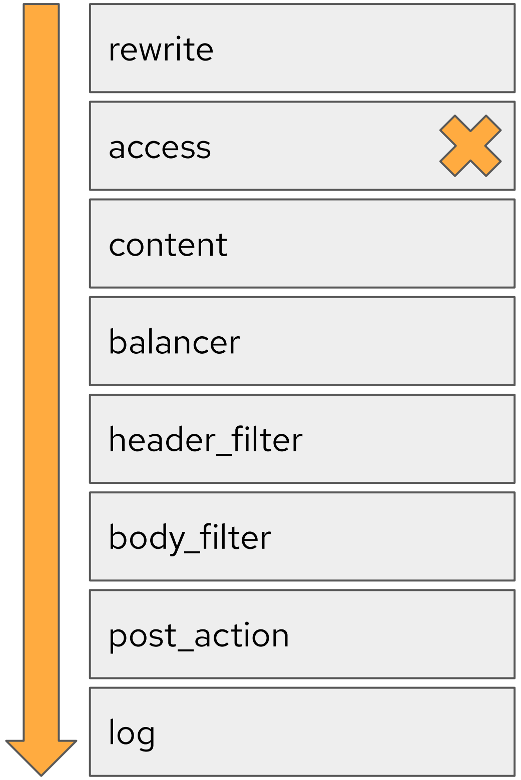
Possible use case: You need to restrict POST actions on an API to users with an JWT claim of 'admin'.

Restricts access based on the claims of the JWT for specific resources. Functions as an allow-only policy, items not matching the rules defined will be blocked with a customizable message for the specified HTTP method.

Other HTTP methods without rules defined will be allowed without checks against JWT claims.

Can define multiple JWT claim rules in the same policy. Can also define multiple Methods to protect in the same policy.

For each resource (i.e. /path) you need to allow a separate claim check is needed.



Liquid Context Debug

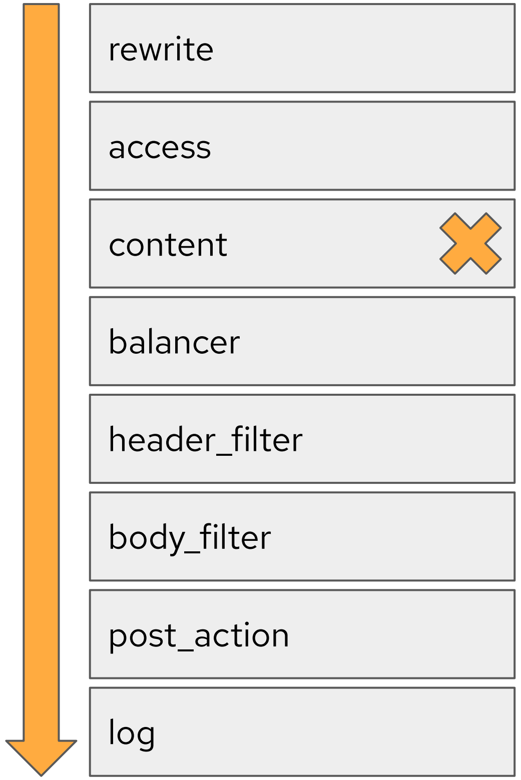
Inspects the available liquid context

Possible use case: A developer needs to better understand what variables are available to them for to use in other policies.

Meant for debugging purposes only. Can be left in the policy chain and disabled when no longer needed, or removed from the chain completely.

Responds back to the API request with a JSON object containing the liquid objects and values that are available when evaluating liquid templates in other policies.

Note: This policy must be placed ahead of the the APIcast policy and any Upstream policies to function correctly.



rewrite

access

content ✗

balancer

header_filter

body_filter

post_action

log

Logging

Controls logging

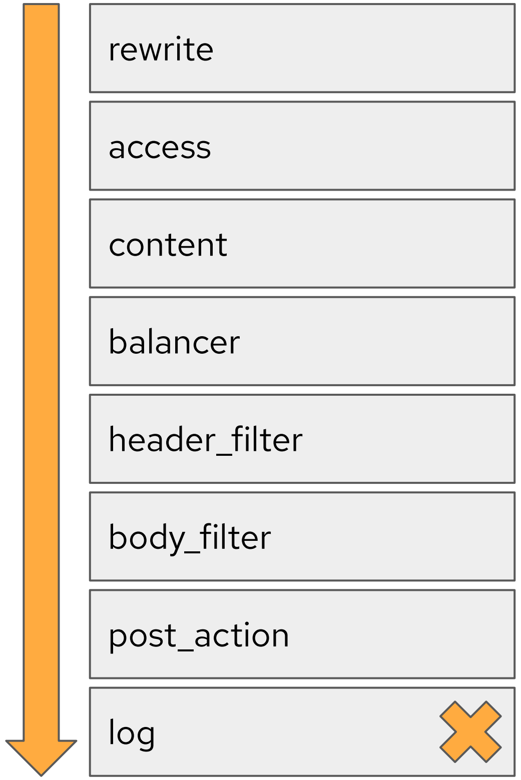
Possible use case: an API developer wants to log only failures by HTTP code and know which service id is failing for which requests.

Allows you to enable or disable access logs per-service. Allows a custom access log format per-service as well. Combines well with the global `APICAST_ACCESS_LOG_FILE` environment variable.

Note: By default the `enable_json_logs` and `enable_access_logs` variables are not enabled.

Utilizing json logs, the custom format will be disabled.

For the custom format you can use nginx default variables, request and response headers, service information, and all properties provided by the `THREESCALE_CONFIG_FILE` and `THREESCALE_PORTAL_ENDPOINT` parameters.

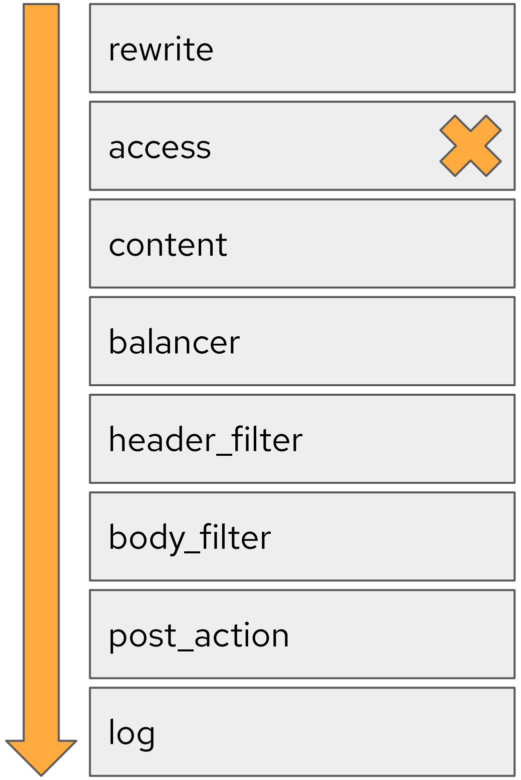


Maintenance Mode

Rejects incoming requests

Possible use case: Maintenance periods for specific resources.

Uses a conditional check (with liquid variables available) to define when to reject a request. The response message and status code are configurable.



NGINX Filter

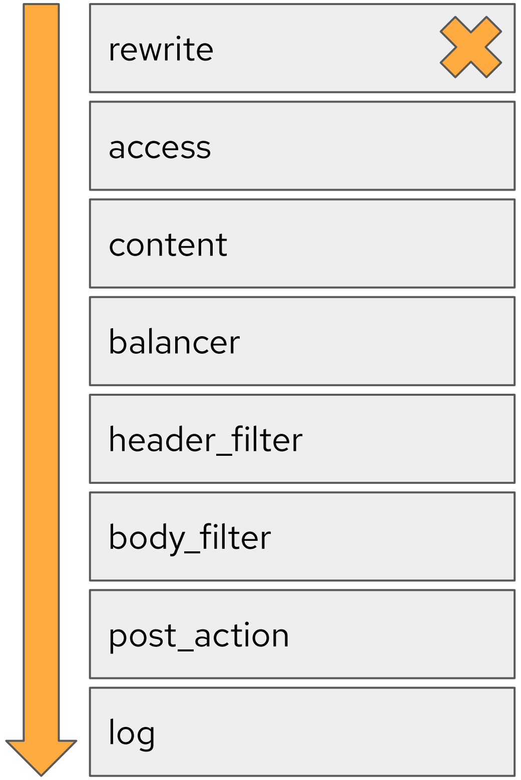
Skip NGINX filters on certain headers

Possible use case: An API developer knows that his backend will pass headers that cannot be validated by NGINX on valid responses.

NGINX automatically checks some headers and rejects requests when it cannot validate them. If you want NGINX to skip validation of particular headers, add the NGINX Filter policy.

Simple policy that accepts a list of headers and optionally passes them upstream. You can specify multiple headers in the same policy.

Note: Using this policy and a header modification policy can create conflicts, exercise caution.



rewrite ✗

access

content

balancer

header_filter

body_filter

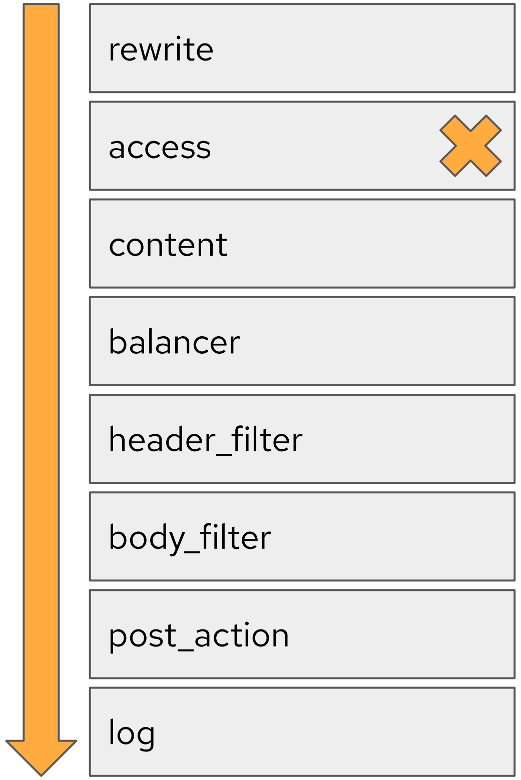
post_action

log

OAuth 2.0 Mutual TLS Client Authentication

Configures OAuth 2.0 Mutual TLS Client Authentication

This policy contains no parameters.



OAuth 2.0 Token Introspection

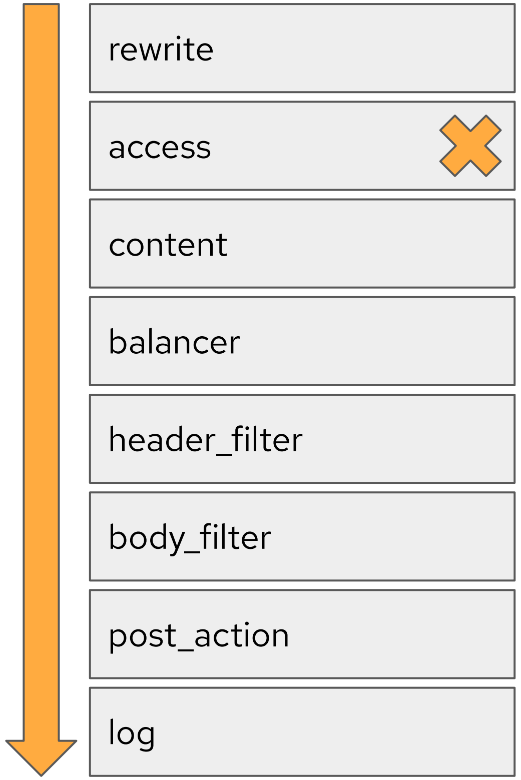
Configures OAuth 2.0 Token Introspection

Allows validating the JWT used for services with the OIDC authentication option using the Token Introspection Endpoint of the issuer (for example Red Hat Single Sign-On).

Authenticates against the Introspection Endpoint using either the client_id and client_secret or the token and issuer endpoint provided provided by OIDC.

A number of tokens can be cached up to 10000, or token caching can be disabled by setting `max_cached_tokens` to 0. Optionally, you can also specify a TTL from 1 to 3600 in seconds.

Note: Because authorization is required for the Introspection call to defeat token scanning (RFC7662), regardless of the setting in the `auth_type` field APIcast uses Basic Authentication to authorize the call to the Introspection Endpoint in the form of a Bas64-encoded *<client_id>:<client_secret>*. The uses of other authentication mechanisms is not currently supported.



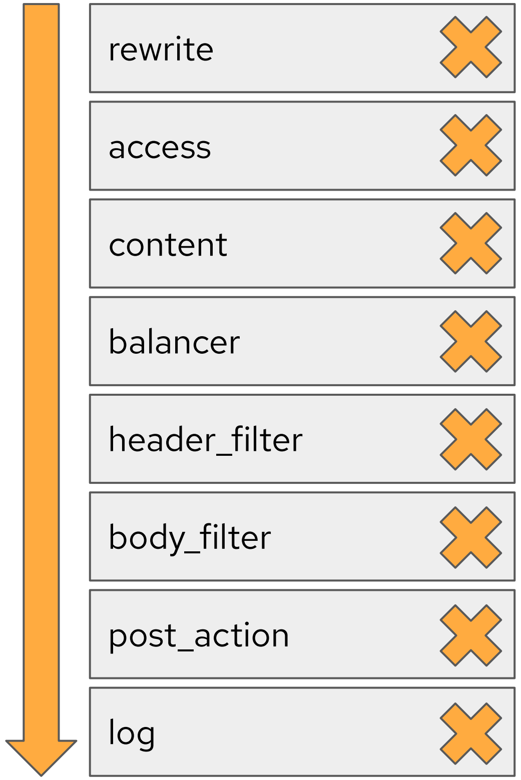
On Fail

Blocks request if any policy fails

Possible use case: You have published a specific HTTP code in your developer portal for situational failures or bad requests.

Without the On Fail policy, any policy in the chain that has an incorrect configuration is skipped by APIcast.

Note: this policy can be anywhere in the chain.



Proxy Service

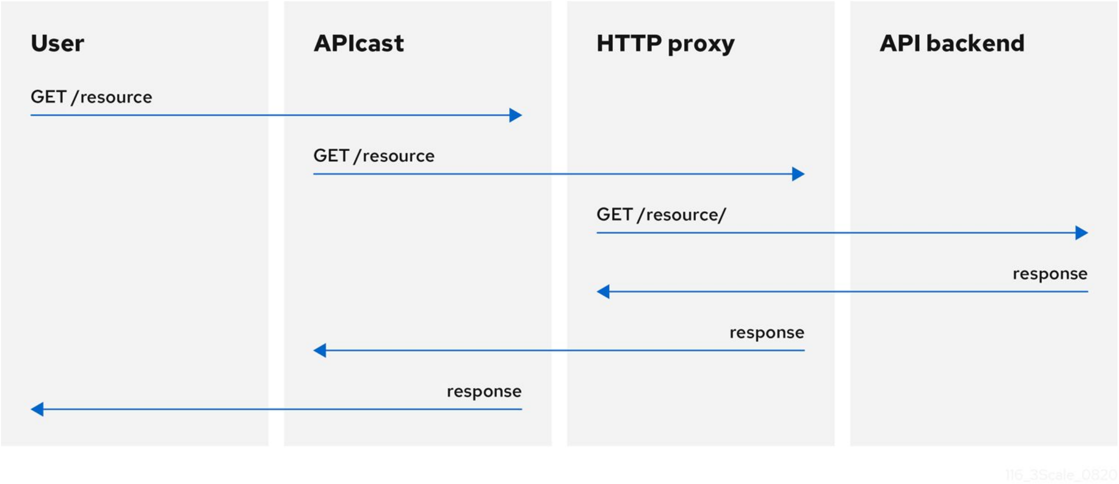
Adds an HTTP proxy to the service.

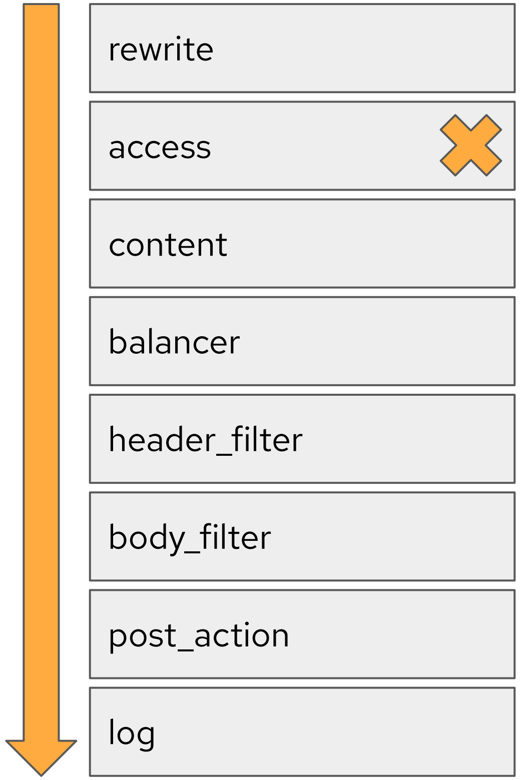
This policy was designed to be able to send requests to a *generic HTTP proxy*. This will apply only to the specific API. If you want to send all 3scale traffic through a proxy, you must use the `HTTP_PROXY` , `HTTPS_PROXY`, or `ALL_PROXY` environment variables.

Can define the proxy for:

- all requests
- http requests
- https requests

`all_proxy` is used when `http_proxy` or `https_proxy` is not defined.





rewrite

access ✗

content

balancer

header_filter

body_filter

post_action

log

Note: this policy will overwrite the proxy environment variables if they are set.

Rate Limit Headers

Set rate limit headers on response

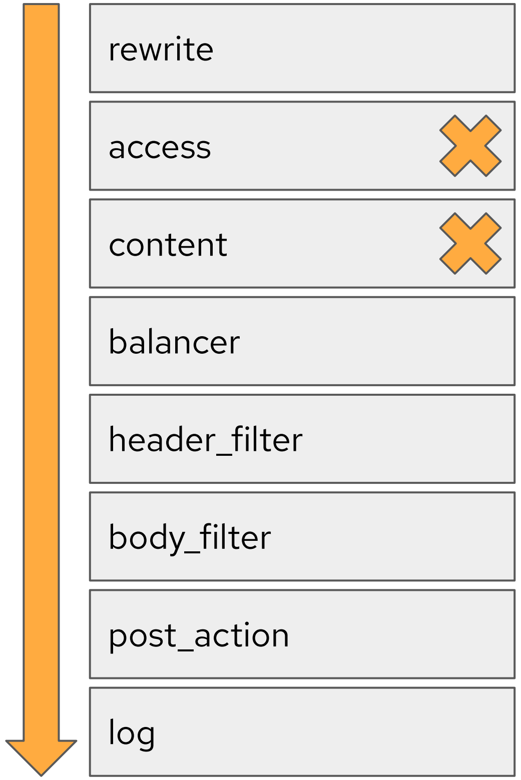
Possible use case: An API developer expects their consumers applications to throttle requests when the RateLimits are close to exhausted.

Different than the Edge Limit policy, this is an informational policy. It contains no parameters, but when enabled will set headers on the response for each message describing where the request falls within the quotas of an application plan with rate limits. It adds the following headers:

- RateLimit-Limit: the total requests for the time window
- RateLimit-Remaining: the remaining requests in the quota
- RateLimit-Reset: the number of seconds remaining in the current time window

No headers are added if the application does not have ratelimits.

Note: this policy must be placed before the 3scale APIcast policy, or it will not work.



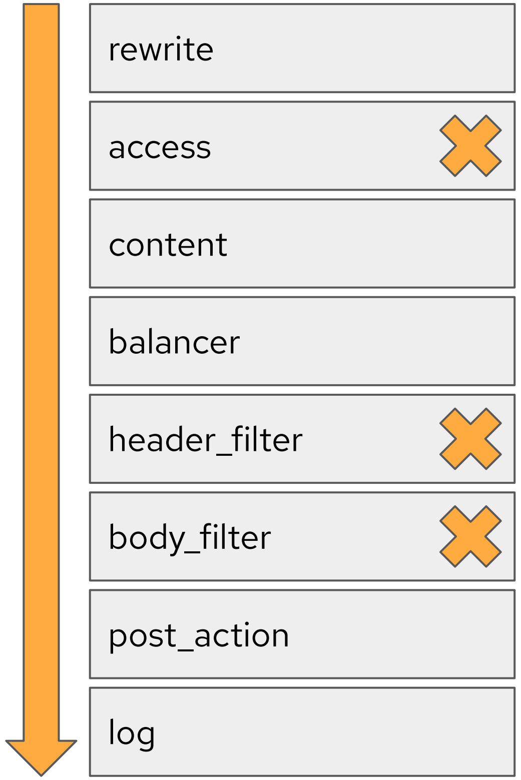
Request/Response Content Limits

Limit request or response base on the size of the content

Possible use case: An API developer wants to protect the backend application from large batches of elements in POST requests, and publishes a maximum size for the payload of a POST request.

By default there are no size limits to the request nor response of an API product. This policy will define limits.

The request and response limits are defined separately. If set to 0 the size is unlimited.



- rewrite
- access ✗
- content
- balancer
- header_filter ✗
- body_filter ✗
- post_action
- log

Retry

Allows to retry requests to the upstream

The configuration of this policy is simple, the parameters are the maximum number of tries.

Note: Currently is is not possible to configure in which cases to retry from the policy. That is controlled with the environment variable: `APICAST_UPSTREAM_RETRY_CASES`

RH-SSO/Keycloak Role Check

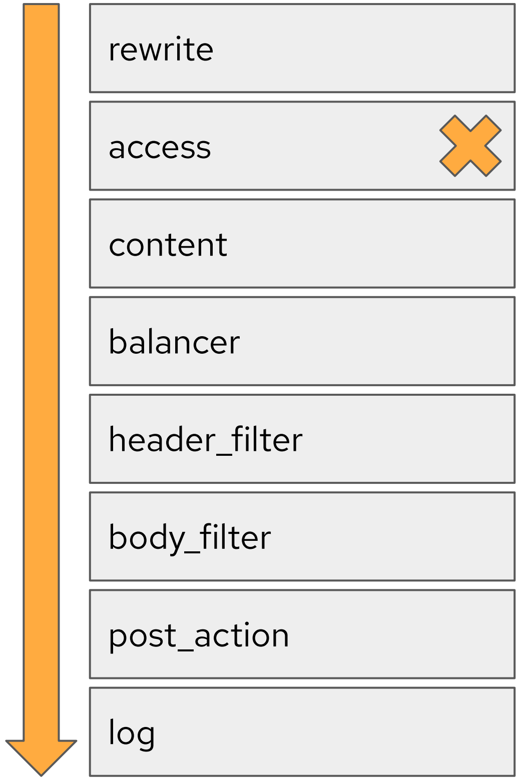
Adds role check with Keycloak

Possible use case: An API developer wants to restrict access to a Product to only consumers from a specific department defined by a role in the JWT token.

For a given resource you can explicitly restrict or allow only configured IDP roles. This policy requires inspects the JWT for the specified scopes. The scope objects define the resource controlled by the rule, which can be plain text or liquid.

For Realms, only the name is needed. For client roles, a type parameter is needed for the value match to take place. Multiples of each role check can be defined.

Note: It is not possible to configure a blacklist and a whitelist in the same policy, however more than one instance of the policy can be added to the chain.



rewrite

access ✗

content

balancer

header_filter

body_filter

post_action

log

Routing

Modify the upstream URL of a request

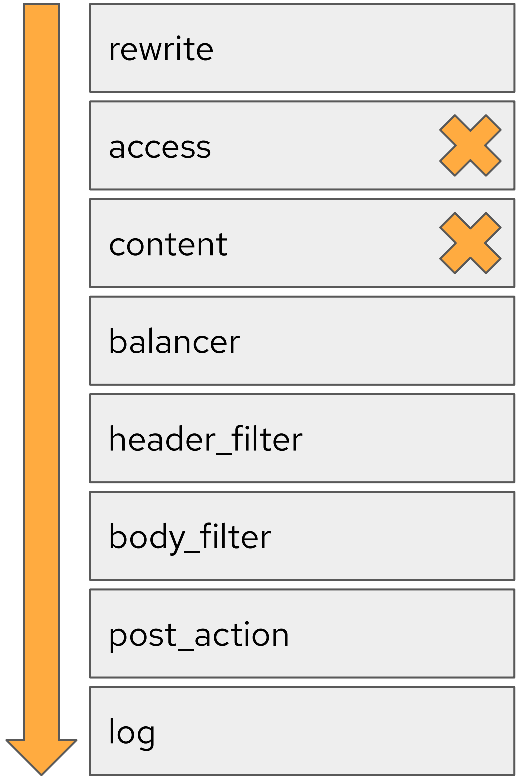
Possible use case: An API developer has a synchronized and a batching backend for the same actions. They want to route batched operations to the second server based on a header value and path.

Rules for routing can be defined for:

- JWT claims (requires the JWT be validated by a JWT Claim Check policy)
- Header and query values
- Paths
- Liquid variables

Multiple conditions can be combined together in a single routing operation. Multiple routing operations can be defined in a single policy.

Note: When you add the Routing policy to a policy chain, the Routing policy must always be immediately before the standard 3scale APIcast policy



rewrite

access ✗

content ✗

balancer

header_filter

body_filter

post_action

log

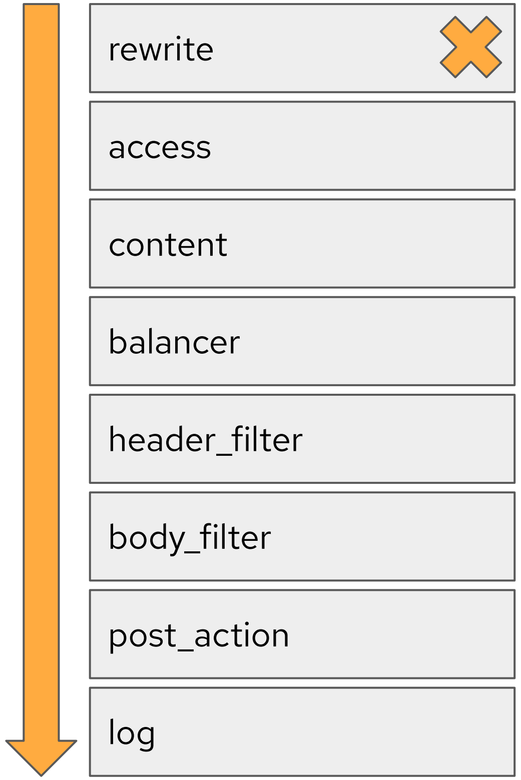
SOAP

Support for a small subset of SOAP

The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. This policy allows for incrementing a system metric when a matching seek value is found either in the SOAPAction header field. The delta of the metric must also be supplied.

More than one mapping rule can be matched in the same policy.

Note: the metric named in the policy must be a existing valid name, or no actions will take place.



rewrite ✗

access

content

balancer

header_filter

body_filter

post_action

log

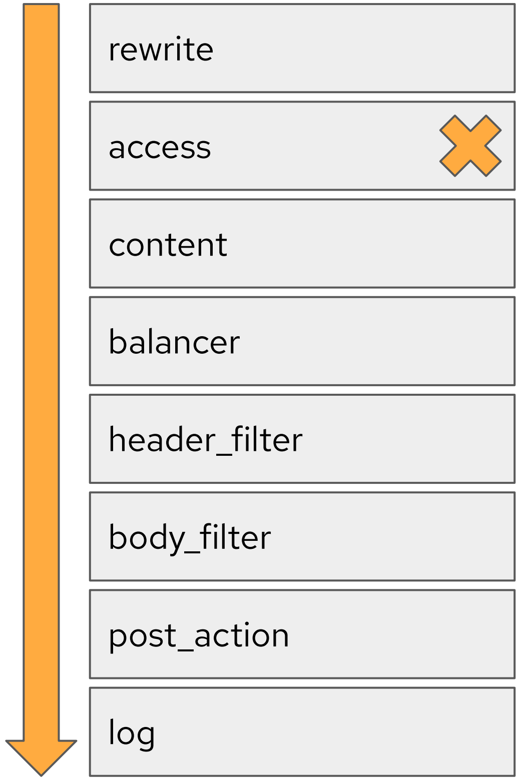
TLS Client Certificate Validation

Validate certificates provided by the client

Possible use case: an API needs to be strictly controlled by SSL certificates to only allow internal traffic with certificates issued by an internal authority.

APIcast will verify the authenticity of the certificate provided by the client against a set of individual certificates or CAs configured in the policy.

Note: In case of an expired or invalid certificate, the request is rejected and no other policies will be processed.



rewrite

access

content

balancer

header_filter

body_filter

post_action

log

TLS Termination

Configure TLS termination certificates

APIcast pulls these configuration settings before establishing a connection to the client, this allows APIcast to finish TLS requests for each API without using a single cert for all APIs.

You can also add multiple certificates and organize them as well.

The certificates can be uploaded directly into the policy config, or they can be specified from a mounted filesystem location in the DeploymentConfig for APIcast.

This policy needs:

- Certificate issued by the user
- A PEM-formatted server certificate
- A PEM-formatted certificate private key

Upstream Connection

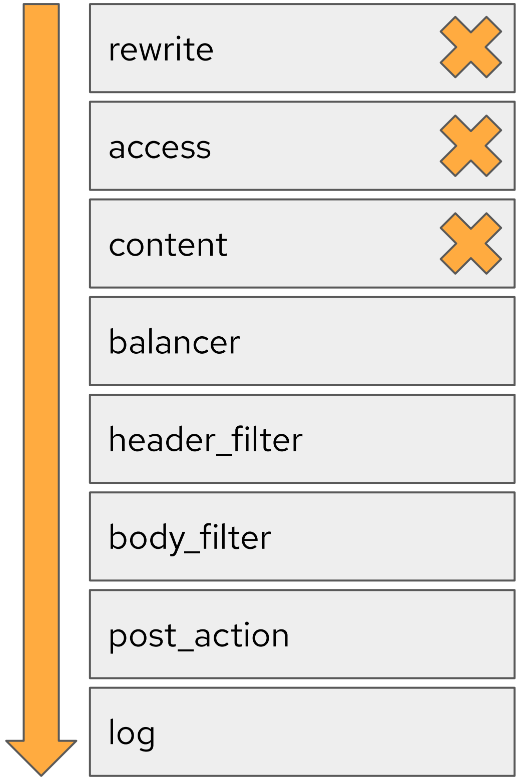
Modify the upstream URL of the request based on its path

Possible use case: The back end servers for a growing API have different hostnames for each version of the service and a developer wants to unify these using the same Public Base URL and a version value in the header, such as directing:

- /public/v1/api ~> api-v1.example.com
- /public/v2/api ~> api-v2.example.com

Parse the Host request header using a regex and replace the upstream URL defined in the Private Base URL with a different URL. Can define multiple rules.

Note: This policy should be placed before the APIcast policy.



rewrite ✗

access ✗

content ✗

balancer

header_filter

body_filter

post_action

log

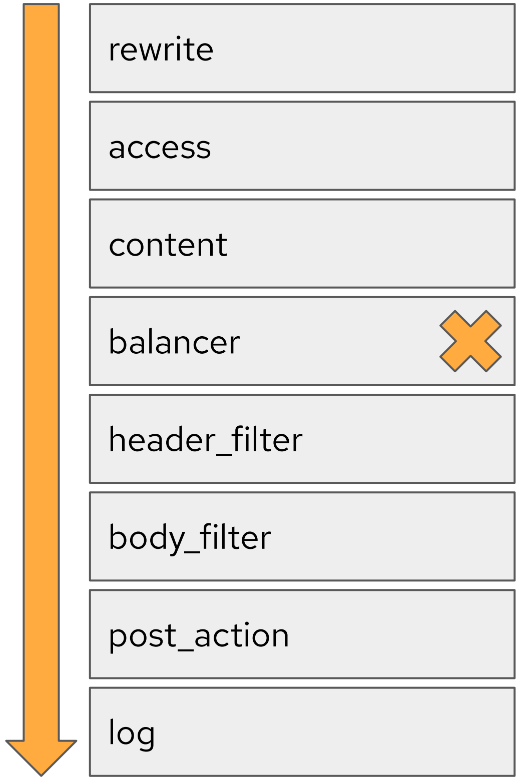
Upstream Mutual TLS

Certificates to be used with the upstream API

Possible use case: A specific API needs to perform a secure handshake with its backend using a key that is unique to the API.

This policy defines what certificate to use when connecting to the upstream API server. The certificate can be stored as a secret, or embedded directly into the policy configuration.

Note: The Upstream mutual TLS policy will overwrite `APICAST_PROXY_HTTPS_CERTIFICATE_KEY` and `APICAST_PROXY_HTTPS_CERTIFICATE` environment variable values. It uses the certificates set by the policy, so those environment variables will have no effect.



URL Rewriting

Allows to modify the path of a request

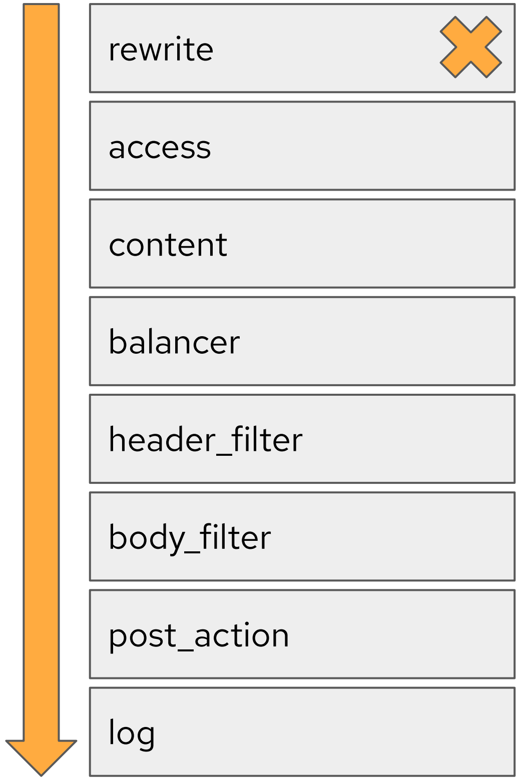
Possible use case: A backend server will fail if there are additional, unrecognized query arguments.

This policy allows you to modify the path and arguments of a query. It can also be limited to specific HTTP verbs as well. Many combinations of both URL substitution and argument manipulation can be used together in a single policy.

For the query string you can `add`, `set`, `push`, and `delete` a specified argument with a supplied value. For the path you can locate a string and replace either the first instance or all instances. For path string replacements, you can also specify a `break` to stop processing further rules when that one is matched.

You can use multiples of each and combinations of both as well as having more than one copy of the policy in the chain.

Note: When this policy is placed above the APIcast policy the mapping rules and metrics will apply to the modified path.



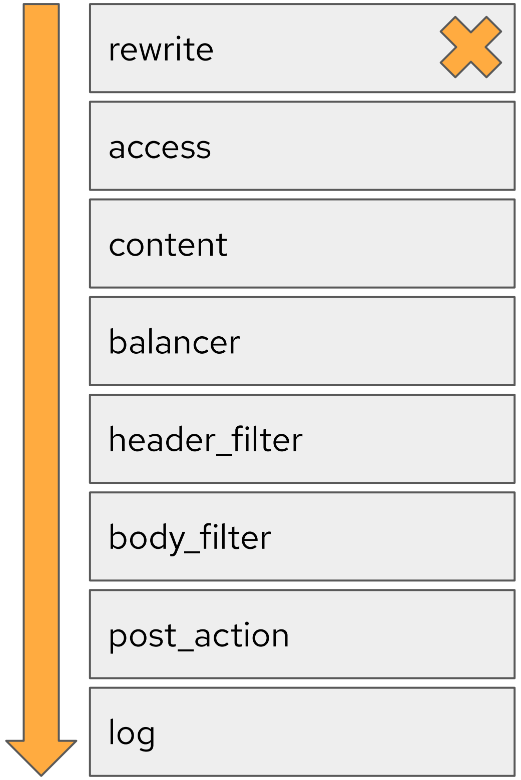
URL Rewriting with Captures

Captures arguments in a URL and rewrites the URL using them

Possible use case: An API developer needs to simplify a complicated URL structure on their backend server with a simpler arrangement to publish to their consumer community.

Similar to URL rewriting, these transformations are pattern based. A rule is created with named placeholders for values that can be reused in the new URL.

Note: As with other actions that run in the rewrite phase if they are placed before the APIcast policy the mapping rules, methods, and metrics will apply to the rewritten URL. Conversely if placed after they will apply on the original request URL.



rewrite

access

content

balancer

header_filter

body_filter

post_action

log

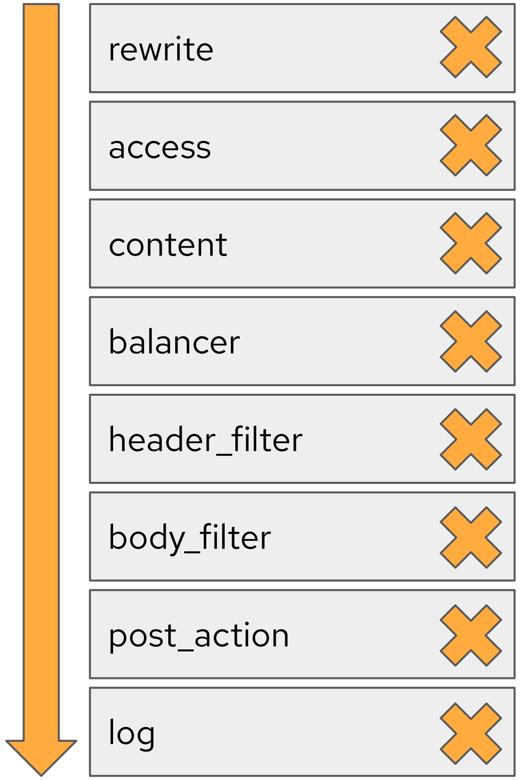
Custom Policies

Custom Policies

A whole new world

If the existing built-in policies are insufficient for your needs. Custom policies can be written to fit your specific needs. Additional reading links are provided to assist with developing custom policies.

They can be created to perform virtually any actions or manipulation of the requests and responses.



Anatomy of a Policy

Example structure of a repository

```
.
├── openshift.yml
└── policies
    └── ngx_example
        └── 1.0.0
            ├── apicast-policy.json
            ├── init.lua
            └── ngx_example.lua
```

The two required files for a policy are:

1. ./policies/${name}/${version}/apicast-policy.json

   Which defines the presentation of the policy in the gui

2. ./policies/${name}/${version}/init.lua

   Which loads the policy code

Note: for human-readability the best practice is to place your custom policy code into a named file and load that file inside init.lua

Policy Build Process

First import a custom ImageStream to use for reference

```
$ oc -n {namespace} import-image amp-apicast-custom:3scale2.10.0 \
--from=registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.10
```

Install the configs to OpenShift

```
$ oc -n {namespace} new-app -f
openshift.yml -o yaml | oc apply -f -
```

Creates one new ImageStream and two new BuildConfigs

Build the custom policy

```
$ oc -n {namespace} start-build
apicast-new-policies --wait --follow
```

Builds a new ImageStream based on the reference with the additional policies inside

Build into the gateway

```
$ oc -n {namespace} start-build
apicast-custom --wait --follow
```

Rebuilds the staging and production pods with the new ImageStream containing the new policies

Custom Policy Resources

Policy Development: Recommended readings

APICast Policies

Example Policy Repository

Phase Logger Policy

Built-in Policies Source Code

Demo Repository

CodeReady Containers

Policy Chains

How to Publish a Policy Chain

- Navigate to the Product > Integration > Policies
- Add the policy(ies) that performs the required actions to accomplish your goal
- Edit their parameters
- Order them as needed
- Save the Policy Chain
- Promote the configuration into the testing sandbox
- Verify functionality
- Publish from testing sandbox to production

Posit: We have a working API that uses weather.gov as its backend and have an appropriate application plan subscription with key.

- Goal: redirect weather API to echo backend for a specific request
- Create "/echo" mapping rule
- Add Upstream Policy (above APIcast)
- Match "echo" to https://echo-api.3scale.net:443
- Save and Publish
- /echo yields https://echo-api.3scale.net/echo results.
- /alerts/active yields weather alerts

Q&A

Homework

Apply what you learned

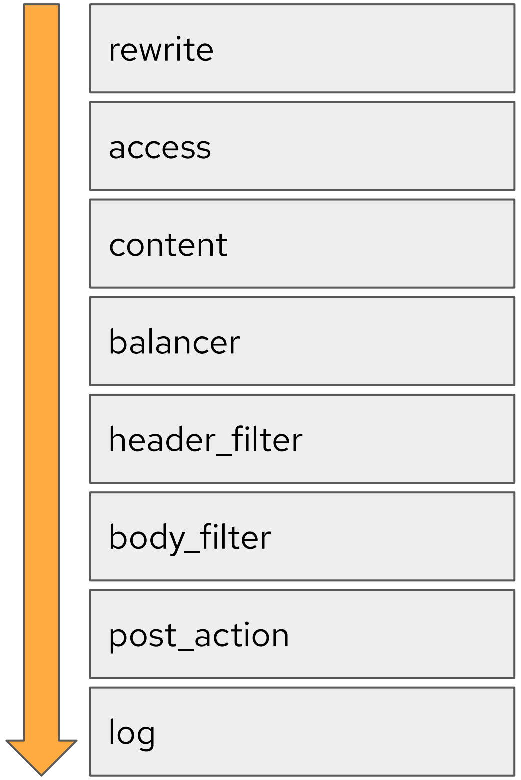
Red Hat

3scale Hackathon

API developer homework.

Problem: You have an API that is functional but you want to create a short-code synonym for an existing call without creating a new mapping rule.

How do you solve this problem?



Additional Resources

3scale Documentation Portal

Administering the API Gateway

Policy documentation

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat