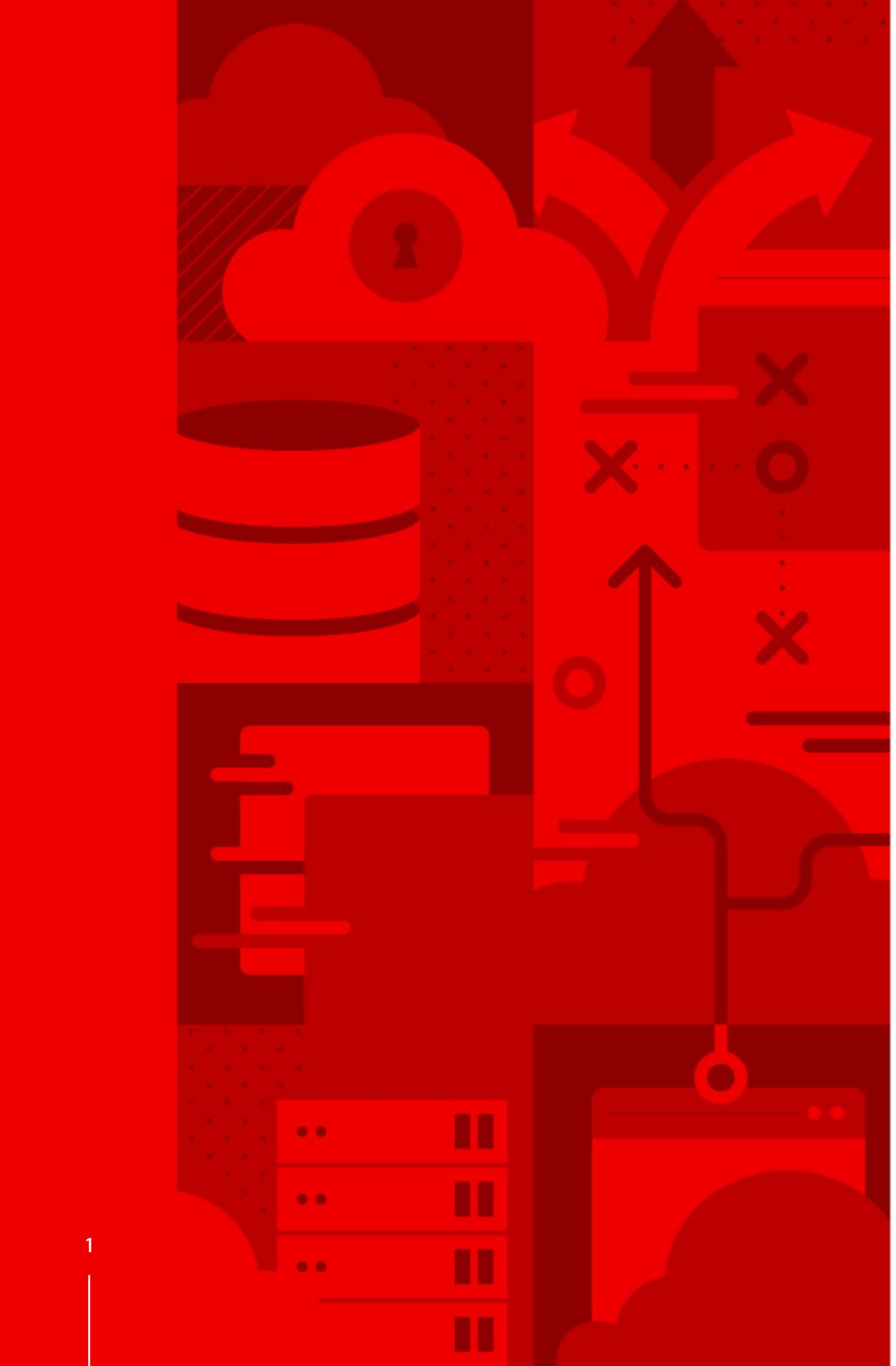


Red Hat 3scale Policies

Implementing custom policies

Phillip Hagerman
Technical Account Manager
June 17, 2021

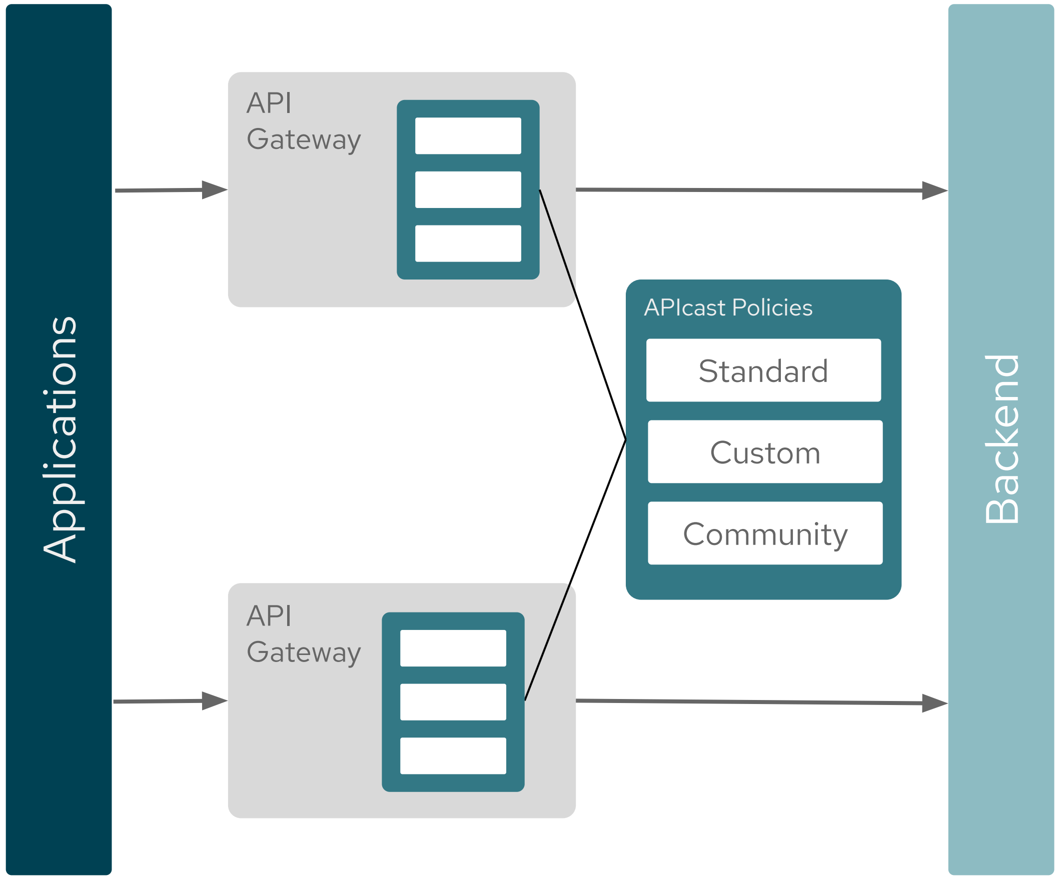
3scale Policies

Flexible modular control

3scale API Gateway policies

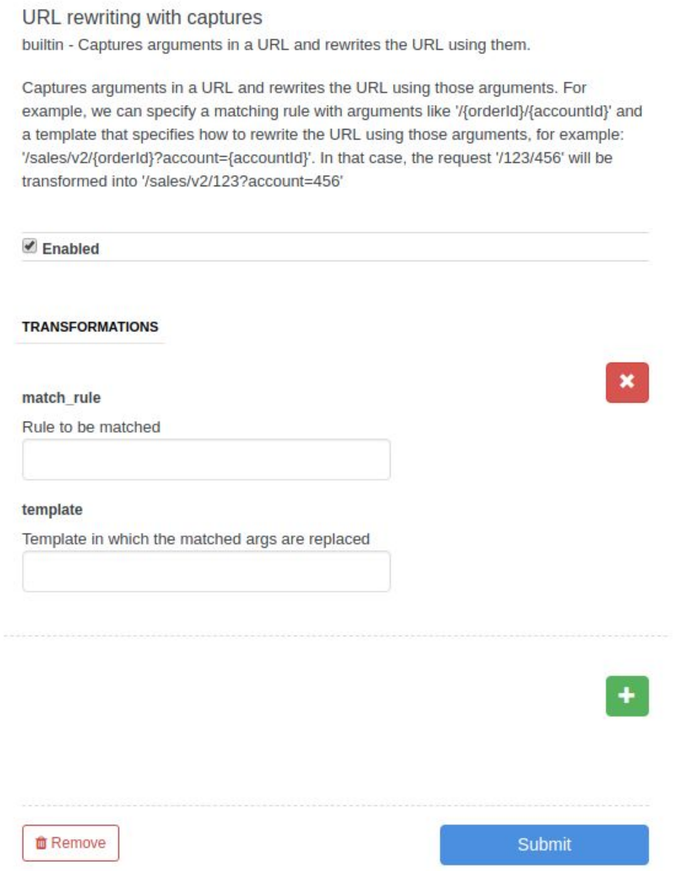
Modular Policy Architecture Benefits

- Descriptive configuration, not code
- Add gateway logic with new policies for any phase of the request cycle
- Better extensibility
- Improved maintainability
- Leverage community contributions
- OOTB policies configurable from UI



URL Rewriting with captures

Captures arguments in a URL and rewrites the URL using them



URL rewriting with captures

builtin - Captures arguments in a URL and rewrites the URL using them.

Captures arguments in a URL and rewrites the URL using those arguments. For example, we can specify a matching rule with arguments like '/{orderId}/{accountId}' and a template that specifies how to rewrite the URL using those arguments, for example: '/sales/v2/{orderId}?account={accountId}'. In that case, the request '/123/456' will be transformed into '/sales/v2/123?account=456'

☑ Enabled

TRANSFORMATIONS

match_rule

Rule to be matched

template

Template in which the matched args are replaced

If we define:

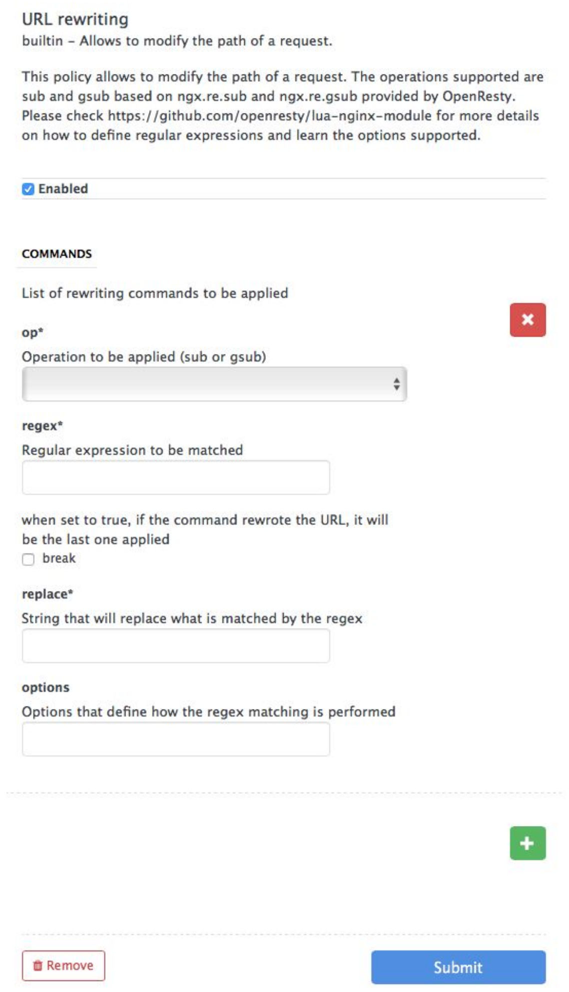
- Matching rule: "/{orderId}/{accountId}"
- Template: "/sales/v2/{orderId}?account={accountId}"

The request "/123/456"

Will become "/sales/v2/123?account=456"

URL rewriting

Allows modification of a path request



URL rewriting
builtin – Allows to modify the path of a request.

This policy allows to modify the path of a request. The operations supported are sub and gsub based on ngx.re.sub and ngx.re.gsub provided by OpenResty. Please check https://github.com/openresty/lua-nginx-module for more details on how to define regular expressions and learn the options supported.

☑ Enabled

COMMANDS

List of rewriting commands to be applied

op*
Operation to be applied (sub or gsub)

regex*
Regular expression to be matched

when set to true, if the command rewrote the URL, it will be the last one applied
☐ break

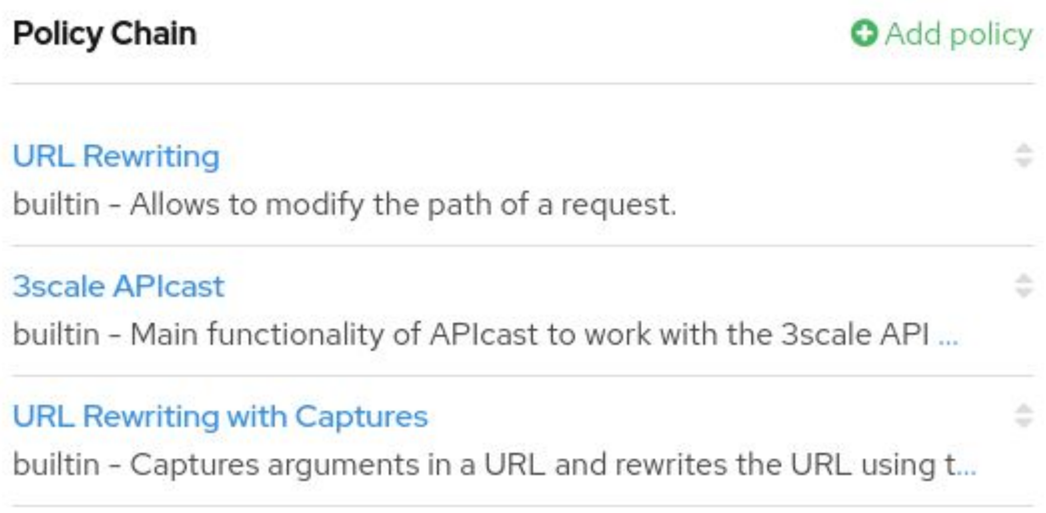
replace*
String that will replace what is matched by the regex

options
Options that define how the regex matching is performed

- Modify the path of a request.
- Can use full PCRE to do sub (single substitution) and gsub (global substitution) operations.

- When used before APIcast Policy then both Mapping Rules and upstream API will see modified URLs.
- When used after APIcast Policy then only the upstream API will see changes.

Policy Chain Order



Policy Chain

URL Rewriting
builtin - Allows to modify the path of a request.

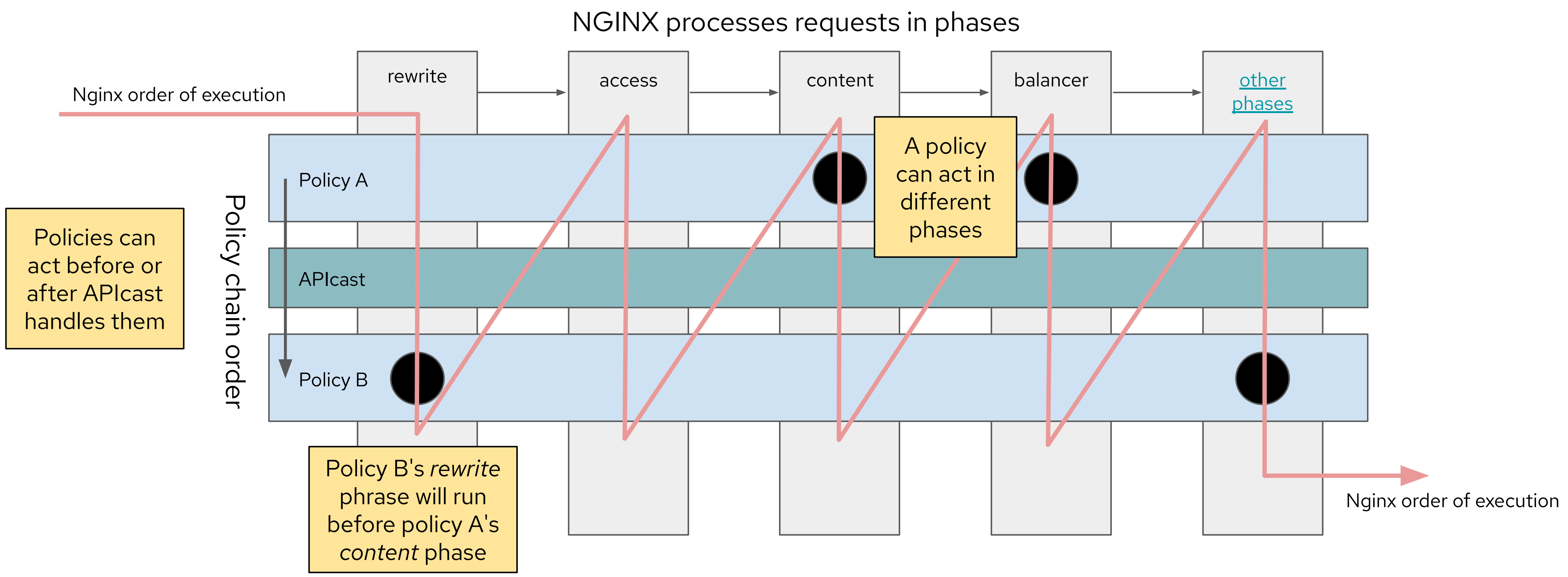
3scale APIcast
builtin - Main functionality of APIcast to work with the 3scale API ...

URL Rewriting with Captures
builtin - Captures arguments in a URL and rewrites the URL using t...

- Policies can act on API calls before the Gateway handles them.
- This can be used to affect the way they are handled.

APICast policy order and Nginx phases

Policies are processed per-defined order for each phase


NGINX processes requests in phases
rewrite
access
content
balancer
other phases
Nginx order of execution
Policy A
APIcast
Policy B
Policy chain order
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 order of execution

API Gateway

Gateway Layer Policy Enforcer

- **Auth Caching:** Control 3scale authorization cache
- **Batcher**: Caches auth from backend and reports
- **Anonymous Access:** Provides default credentials for unauthenticated requests
- **CORS**: Enable Cross-Origin Resource Sharing
- **Echo**: Prints the request back to the client (status code optional)
- **Edge Limiting**: Algorithm-based rate limiting, allows global and per service caching
- **Header Modification**: Allows control of HTTP request and response headers
- **IP Check:** Accepts or denies a request based on the IP
- **JWT Claim Check**: Define rules based on JSON Web Token (JWT) claim, resource target, and the method that you are interested in blocking
- **Liquid Context Debugging:** Expose request context values, useful for debugging
- **Logging**: Enables / disables access logs per service
- **OAuth Token Introspection**: Executes OAuth 2.0 token introspection for every API call
- **Prometheus Metrics**: Enable backend metrics
- **Referrer**: Sends the contents of the Referer HTTP header to backend so it can be validated
- **Retry**: Sets the number of retry requests to the upstream API
- **RH-SSO/Keycloak Role Check**: Adds role check when used with the OpenID Connect authentication option
- **Routing**: Route requests to different target endpoints
- **SOAP**: Adds support for small subset of SOAP
- **TLS Client Certificate Validation**: implements a TLS handshake and validates the client certificate against a whitelist
- **Upstream**: Modify upstream URL of the request based on its path
- **Upstream Connection:** change the default values of proxy connect, send, read timeout
- **Url Rewriting**: Allows modification of a path request & query string
- **Url Rewriting with Captures**: Retrieves arguments in the URL and uses their values in the rewritten URL

Custom Policies

How to make them.

How to use them.

Anatomy of a policy

Example structure of a demo repository

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

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

Openshift definitions

ImageStream

```
- apiVersion: v1
  kind: ImageStream
  metadata:
    annotations:
    labels:
      app: apicast
    name: apicast-new-policies
```

BuildConfig (for policy)

```
- apiVersion: v1
  kind: BuildConfig
  metadata:
    annotations:
    labels:
      app: apicast
    name: apicast-new-policies
  spec:
    output:
      to:
        kind: ImageStreamTag
        name:
'apicast-new-policies:${NEW_POLICY_RELEASE}'
    source:
      git:
        uri: ${GIT_REPO}
        ref: 'master'
      type: Git
    strategy:
      sourceStrategy:
        from:
          kind: ImageStreamTag
          name:
'amp-apicast-custom:${CUSTOM_IS_TAG}'
          namespace: ${APICAST_CUSTOM_NAMESPACE}
```

BuildConfig (for implementation)

```
- apiVersion: v1
  kind: BuildConfig
  metadata:
    annotations:
    labels:
      app: apicast
    name: apicast-custom
  spec:
    nodeSelector: null
    output:
      to:
        kind: ImageStreamTag
        name: 'amp-apicast:${AMP_RELEASE}'
    postCommit:
      args:
        - '--test'
        - '--lazy'
      command:
        - bin/apicast
    resources: {}
    runPolicy: Serial
    source:
      images:
        - from:
            kind: ImageStreamTag
            name:
'apicast-new-policies:${NEW_POLICY_RELEASE}'
```

cont.

```
          paths:
            - destinationDir: policies
              sourcePath:
/opt/app-root/policies/ngx-example
      type: Dockerfile
      dockerfile: |
        FROM scratch
        COPY . src
        USER root

    strategy:
      dockerStrategy:
        from:
          kind: ImageStreamTag
          name:
'amp-apicast-custom:${CUSTOM_IS_TAG}'
          namespace: ${APICAST_CUSTOM_NAMESPACE}
      type: Docker
```

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 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 gateway

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

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

Demo

Additional Resources

Policy Development: Recommended readings

APICast Policies

Example Policy Repository

Phase Logger Policy

Built-in Policies Source Code

Demo Repository

CodeReady Containers

Thank you

Red Hat