Red Hat 3scale API Management 2.14

管理 API 网关

中等到高级目标，以管理您的安装。

Last Updated: 2024-06-29

中等到高级目标，以管理您的安装。

法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

本指南提供有关配置任务的信息，可在基本安装之后执行。

目录

对红帽文档提供反馈 5
部分 I. API 网关 6
第 1 章 3SCALE API 管理 APICAST API 网关的高级操作简介 7
1.1. 用于调用 3SCALE API 管理 API 的公共基本 URL 7
1.2. APICAST 如何应用映射规则来捕获 3SCALE API 管理 API 的使用 7
1.3. APICAST 如何处理具有自定义要求的 API 10
第 2 章 操作 DOCKER 容器化环境 12
2.1. 对 DOCKER 容器化环境的 APICAST 进行故障排除 12
第 3 章 高级 APICAST 配置 13
3.1. 定义 SECRET 令牌 13
3.2. 凭证 13
3.3. 配置错误消息 14
3.4. 配置历史记录 15
3.5. 调试 15
3.6. 路径路由 16
第 4 章 APICAST 策略 17
4.1. 用于更改默认 3SCALE API 管理 APICAST 行为的标准策略 17
4.2. 3SCALE API 管理标准策略中的策略链 72
4.3. 使用 API 修改代理策略链 81
4.4. 自定义 3SCALE API 管理 APICAST 策略 83
第 5 章 将策略链与 APICAST 原生部署集成 90
5.1. 在策略中使用变量和过滤器 90
第 6 章 使用 FUSE 中的策略扩展转换 3SCALE API 管理消息内容 94
6.1. 在 FUSE 中集成 APICAST 与 APACHE CAMEL 转换 94
6.2. 配置使用 OPENSHIFT 上 FUSE 中的 APACHE CAMEL 创建的 APICAST 策略扩展 97
第 7 章 APICAST 环境变量 100
ALL_PROXY 103
APICAST_ACCESS_LOG_BUFFER 104
APICAST_ACCESS_LOG_FILE 104
APICAST_BACKEND_CACHE_HANDLER 104
APICAST_CACHE_MAX_TIME 105
APICAST_CACHE_STATUS_CODES 105
APICAST_CONFIGURATION_CACHE 105
APICAST_CONFIGURATION_LOADER 106
APICAST_CUSTOM_CONFIG 106
APICAST_ENVIRONMENT 106
APICAST_EXTENDED_METRICS 106
APICAST_HTTPS_CERTIFICATE 107
APICAST_HTTPS_CERTIFICATE_KEY 107
APICAST_HTTPS_PORT 107
APICAST_HTTPS_PROXY_PROTOCOL 107
APICAST_HTTPS_VERIFY_DEPTH 107
APICAST_HTTP_PROXY_PROTOCOL 108
APICAST_LARGE_CLIENT_HEADER_BUFFERS 108
APICAST_LOAD_SERVICES_WHEN_NEEDED 108

APICAST_LOG_FILE 109
APICAST_LOG_LEVEL 109
APICAST_MANAGEMENT_API 110
APICAST_MODULE 110
APICAST_OIDC_LOG_LEVEL 110
APICAST_PATH_ROUTING 110
APICAST_PATH_ROUTING_ONLY 111
APICAST_POLICY_LOAD_PATH 111
APICAST_PROXY_HTTPS_CERTIFICATE 111
APICAST_PROXY_HTTPS_CERTIFICATE_KEY 112
APICAST_PROXY_HTTPS_PASSWORD_FILE 112
APICAST_PROXY_HTTPS_SESSION_REUSE 112
APICAST_REPORTING_THREADS 113
APICAST_RESPONSE_CODES 113
APICAST_SERVICE_CACHE_SIZE 113
APICAST_SERVICE_${ID}_CONFIGURATION_VERSION 114
APICAST_SERVICES_LIST 114
APICAST_SERVICES_FILTER_BY_URL 114
APICAST_UPSTREAM_RETRY_CASES 115
APICAST_WORKERS 115
BACKEND_ENDPOINT_OVERRIDE 115
HTTP_KEEPALIVE_TIMEOUT 115
HTTP_PROXY,HTTP_PROXY 116
HTTPS_PROXY,HTTPS_PROXY 116
NO_PROXY,NO_PROXY 117
OPENSSL_VERIFY 117
OPENTRACING_CONFIG 117
OPENTRACING_HEADER_FORWARD 118
OPENTRACING_TRACER 118
RESOLVER 118
THREESCALE_CONFIG_FILE 118
THREESCALE_DEPLOYMENT_ENV 119
THREESCALE_PORTAL_ENDPOINT 119

第 8 章 配置 APICAST 以获得更好的性能 122
8.1. 常规指南 122
8.2. 默认缓存 123
8.3. 异步报告线程 125
8.4. 3SCALE API 管理 BATCHER 策略 126

第 9 章 向 PROMETHEUS 公开 3SCALE API 管理 APICAST METRICS 128
9.1. 关于 PROMETHEUS 128
9.2. APICAST 与 PROMETHEUS 集成 129
9.3. 3SCALE API 管理 APICAST 的 OPENSHIFT 环境变量 130
9.4. 3SCALE API 管理 APICAST 指标公开给 PROMETHEUS 131

第 10 章 将 OPENTELEMETRY SDK 与 APICAST 集成 134
10.1. 用于部署 OTP 和 GRPC TRACE 的 JAEGER 服务示例 134
10.2. 配置 APICAST 进行追踪 136
10.3. 其他资源 137

部分 II. API 版本控制 138

第 11 章 API 版本 139

11.1. 目标 139
11.2. 先决条件 139
11.3. URL 版本 139
11.4. 端点版本控制 143
11.5. 自定义标头版本 143

部分 III. API 身份验证 145

第 12 章 身份验证模式 146
12.1. 支持的验证模式 146
12.2. 设置身份验证模式 146
12.3. 标准验证模式 148
12.4. 参考器过滤 149

第 13 章 将 3SCALE API 管理与 OPENID CONNECT 身份提供程序集成 154
13.1. 集成 3SCALE API 管理和 OPENID CONNECT 身份提供程序概述 155
13.2. APICAST 如何处理 JSON WEB 令牌 157
13.3. 3SCALE API 管理 ZYNC 如何将应用程序详情与 OPENID CONNECT 身份提供程序同步 159
13.4. 将 3SCALE API 管理与红帽单点登录集成为 OPENID CONNECT 身份提供程序 160
13.5. 将 3SCALE API 管理与第三方 OPENID CONNECT 身份提供程序集成 167
13.6. 测试 3SCALE API 管理与 OPENID CONNECT 身份提供程序的集成 170
13.7. 3SCALE API 管理与 OPENID CONNECT 身份提供程序集成的示例 172

对红帽文档提供反馈

我们感谢您对我们文档的反馈。

要改进，创建一个 JIRA 问题并描述您推荐的更改。提供尽可能多的详细信息，以便我们快速解决您的请求。

前提条件

- 您有红帽客户门户网站帐户。此帐户可让您登录到 Red Hat Jira Software 实例。如果您没有帐户，系统会提示您创建一个帐户。

流程

1. 单击以下链接：创建问题。
2. 在 **Summary** 文本框中输入问题的简短描述。
3. 在 **Description** 文本框中提供以下信息：
   - 找到此问题的页面的 URL。
   - 有关此问题的详细描述。
     您可以将信息保留在任何其他字段中的默认值。
4. 点 **Create** 将 JIRA 问题提交到文档团队。

感谢您花时间来提供反馈。

管理 API 网关可帮助您将高级配置功能应用到 3scale 安装。有关安装的基本详情，请参阅安装 3scale。

部分 I. API 网关

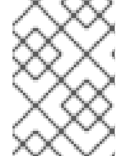
第 1 章 3SCALE API 管理 APICAST API 网关的高级操作简介

3scale APIcast 的高级操作将有助于您调整对应用程序编程接口(API)的访问配置。

1.1. 用于调用 3SCALE API 管理 API 的公共基本 URL

公共基本 URL 是您的 API 客户用来向 API 产品发出请求的 URL，该产品可通过 3scale 公开。这将是您的 APIcast 实例的 URL。

如果您使用自助管理部署选项之一，您可以在您管理的域名上为提供的每个环境（登台和生产）选择自己的公共基本 URL。这个 URL 应该与您 API 后端的 URL 不同，可能类似 **https://api.yourdomain.com:443**，其中 **yourdomain.com** 是属于您的域。设置公共基础 URL 后，请确保保存更改，如有必要，将暂存中的更改提升到生产。



注意

您指定的公共基本 URL 必须使用 OpenShift 集群中可用的端口。默认情况下，OpenShift 路由器仅侦听标准 HTTP 和 HTTPS 端口（80 和 443）上的连接。如果您希望用户通过某些其他端口连接到您的 API，请与您的 OpenShift 管理员合作以启用该端口。

APIcast 仅接受对公共基础 URL 中指定的主机名的调用。例如，如果您将 **https://echo-api.3scale.net:443** 指定为公共基本 URL，则正确的调用将是：

```
curl "https://echo-api.3scale.net:443/hello?user_key=you_user_key"
```

如果您的 API 没有公共域，您可以在请求中使用 APIcast IP 地址，但您仍需要在*公共基础 URL* 字段中指定一个值，即使该域不是真实的。在这种情况下，请确保在 Host 标头中提供主机。例如：

```
curl "http://192.0.2.12:80/hello?user_key=your_user_key" -H "Host: echo-api.3scale.net"
```

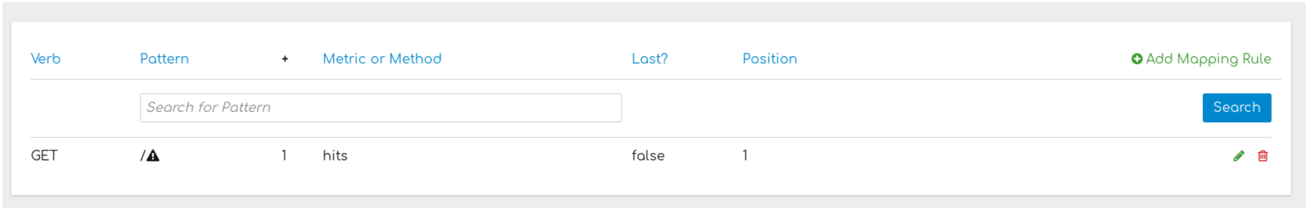
如果要在本地机器上部署，您可以将"localhost"指定为域，因此公共基本 URL 类似 **http://localhost:80**，然后您可以发出类似如下的请求：

```
curl "http://localhost:80/hello?user_key=your_user_key"
```

如果您有多个 API 产品，请为每个产品设置适当的公共基础 URL。APIcast 根据主机名路由请求。

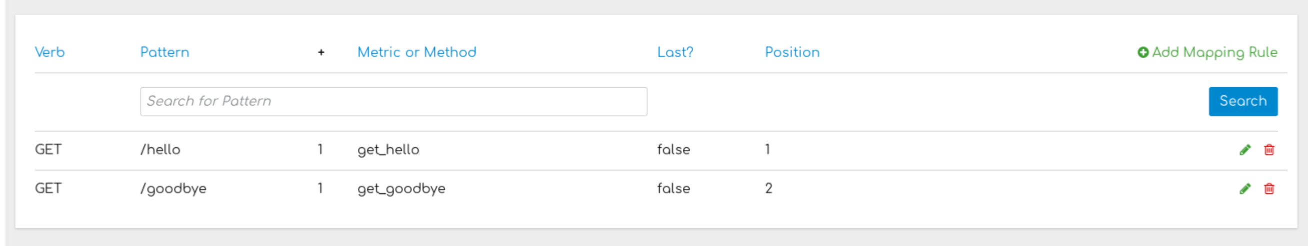
1.2. APICAST 如何应用映射规则来捕获 3SCALE API 管理 API 的使用

根据对 API 的请求，映射规则定义指标或指定您要捕获 API 使用情况的方法。以下是映射规则的示例：



此规则意味着，任何以 / 开头的 **GET** 请求都以 1 为指标 **hits** 的增量。此规则匹配到您的 API 的任何请求。虽然这是有效的映射规则，但它太通用，如果您添加了更具体的映射规则，则通常会导致被双倍计数。

Echo API 的以下映射规则显示更为具体的示例：



映射规则在 API 产品和 API 后端级别上工作。

- 在产品级别上映射规则。
  - 映射规则具有优先权。这意味着产品映射规则是第一个要评估的规则。
  - 映射规则始终被评估，并且独立于这些后端接收重定向的流量。
- 在后端级别映射规则。
  - 当您向后端添加映射规则时，这些规则将添加到所有产品中，并绑定所述后端。
  - 映射规则在产品级别上定义的映射规则后评估。
  - 只有在流量重定向到映射规则所属的同一后端时才评估映射规则。
  - 产品后端的路径会自动放在捆绑到上述产品的后端的每个映射规则的前面。

使用产品和后端映射规则示例

以下示例显示了一个后端产品的映射规则。

- Echo API 后端：
  - 具有专用端点：**https://echo-api.3scale.net**
  - 包含使用以下模式的 2 个映射规则：

    ```
    /hello
    /bye
    ```

- Cool API 产品：
  - 具有这个公共端点：**https://cool.api**
  - 通过以下路由路径使用 Echo API 后端： **/echo**。
- 使用以下模式映射规则会自动成为 Cool API 产品的一部分：

  ```
  /echo/hello
  /echo/bye
  ```

  - 这意味着发送到公共 URL **https://cool.api/echo/hello** 的请求被重新定向到 **https://echo-api.3scale.net/hello**。

  - 同样，发送到 **https://cool.api/echo/bye** 的请求重定向至 **https://echo-api.3scale.net/bye**。

现在，假设有一个名为 Tools For Devs 的其他产品，它使用相同的 Echo API 后端。

- Tools for Devs 产品：
  - 具有这个公共端点：**https://dev-tools.api**
  - 通过以下路由路径使用 Echo API 后端：**/tellmeback**。
- 使用以下模式映射规则是 Tools For Devs 产品的一部分：

  > /tellmeback/hello
  > /tellmeback/bye

  - 因此，发送到公共 URL **https://dev-tools.api/tellmeback/hello** 的请求会被重定向到 **https://echo-api.3scale.net/hello**。
  - 同样，发送到 **https://dev-tools.api/tellmeback/bye** 的请求重定向至 **https://echo-api.3scale.net/bye**。
- 如果您使用 **/ping** 模式的映射规则添加到 Echo API 后端，则产品 - Cool API 和 Tools For Devs 都会受到影响：
  - cool API 具有一个使用此模式的映射规则：**/echo/ping**。
  - Tools For Devs 有具有此模式的映射规则：**/tellmeback/ping**。

映射规则的匹配

3scale 根据前缀应用映射规则。表示法遵循 OpenAPI 和 ActiveDocs 规格：

- 映射规则必须以正斜杠(/)开头。
- 在路径上对文字字符串执行匹配，即 URL，例如 **/hello**。
  - 映射规则保存后，将导致请求发送到您已设置的 URL 字符串，并调用您围绕每个映射规则定义的指标或方法。
- 映射规则可以在查询字符串或正文中包含参数，例如 /**{word}?value={value}**。
- APIcast 获取参数的方式如下：
  - **GET** 方法：来自查询字符串。
  - **POST**、**DELETE** 或 **PUT** 方法：来自正文。
- 映射规则可以包含命名通配符，例如 /**{word}**。此规则与占位符 **{word}** 中的任何内容匹配，它会使 /**morning** 等请求与映射规则匹配。通配符可以在斜杠之间或斜杠和点之间出现。参数也可以包含通配符。
- 默认情况下，所有映射规则都会根据您指定的排序顺序从第一到最后一个评估。如果您添加规则 **/v1**，它将匹配路径以 **/v1** 开头的请求，例如：**/v1/word** 或 **/v1/sentence**。
- 您可以在模式的末尾添加一个美元符号($)来指定完全匹配项。例如，**/v1/word** 仅匹配 **/v1/word** 请求，不匹配 **/v1/word/hello** 请求。要完全匹配，还必须确保禁用了与所有(/)匹配的默认映射规则。

- 多个映射规则可以匹配请求路径，但如果都不匹配，则会使用 HTTP 404 状态代码丢弃该请求。

映射规则工作流

映射规则有以下工作流：

- 您可以随时定义一个新的映射规则。请参阅 定义映射规则。
- 下一次重新加载时将灰显映射规则，以防止意外修改。
- 要编辑现有的映射规则，您必须首先通过单击右侧的铅笔图标启用它。
- 若要删除规则，可单击回收站图标。
- 当您提升 **Integration** > **Configuration** 中的更改时，所有修改和删除都会保存。

停止其他映射规则

要停止处理进一步映射规则，请在创建新映射规则时选择 **Last?** 选项，特别是在处理一个或多个映射规则后。例如，如果您定义了与 **API 集成设置**中不同指标关联的多个映射规则，例如：

```
(get) /path/to/example/search
(get) /path/to/example/{id}
```

规则 **/path/to/example/search** 可以标记为 **Last?**，然后在调用 **(get)/path/to/example/search** 后，APIcast 停止处理，且不会在剩余的规则中搜索匹配项，规则 **(get)/path/to/example/{id}** 的指标将不会被递增。

1.3. APICAST 如何处理具有自定义要求的 API

有些特殊情形需要自定义 APIcast 配置，以便 API 用户能够成功调用 API。

主机标头

除非 **Host** 标头与预期匹配，否则仅将这些 API 产品需要此选项。在这些情况下，在 API 产品前面有一个网关会导致问题，因为 **Host** 是网关之一，例如：**xxxx-yyy.staging.apicast.io**。

要避免这个问题，您可以在 *Authentication Settings* **的 Host Header 字段中定义 API 产品所需的主机：[Your_product_name] > Integration > Settings**。

其结果是托管 APIcast 实例会在请求调用中重写主机规格。

▾ AUTHENTICATION SETTINGS

Host Header

Lets you define a custom `Host` request header. This is needed if your API backend only accepts traffic from a specific host.

保护 API 后端

在 APIcast 在生产环境中工作后，您可能希望将直接访问 API 产品限制为仅那些指定您指定的 secret 令牌的调用。为此，请设置 APIcast **Secret Token**。如需有关如何设置它的信息，请参阅 高级 APIcast 配置。

将 APIcast 与私有 API 搭配使用

借助 APIcast，可以保护互联网上不能公开访问的 API。必须满足的要求有：

- 自我管理的 APIcast 必须用作部署选项。
- APIcast 需要能够从公共互联网访问，并且能够向 3scale 服务管理 API 发出出站调用。
- API 产品应该可由 APIcast 访问。
  在这种情况下，您可以在 *Private Base URL* 字段中设置您的内部域名或 API 的 IP 地址，并照常执行其余步骤。但是，执行此操作意味着您无法利用暂存环境。测试调用将无法成功，因为暂存 APIcast 实例由 3scale 托管，无法访问您的私有 API 后端。在生产环境中部署 APIcast 后，如果配置正确，APIcast 可以按预期工作。

第 2 章 操作 DOCKER 容器化环境

2.1. 对 DOCKER 容器化环境的 APICAST 进行故障排除

本节介绍了在 Docker 容器化环境中使用 APIcast 时最常发现的问题。

2.1.1. *无法连接到 Docker 守护进程*错误

docker: 无法连接到 Docker 守护进程。Is the docker daemon running on this host?错误消息可能是 Docker 服务尚未启动。您可以通过运行 **sudo systemctl status docker.service** 命令来检查 Docker 守护进程的状态。

确保您以 **root** 用户身份运行此命令，因为 Docker 容器化环境默认需要在 RHEL 中具有 root 权限。有关详细信息，请参阅 此处。

2.1.2. 基本 Docker 命令行界面命令

如果您以分离模式启动容器（**-d** 选项），并希望检查正在运行的 APIcast 实例的日志，您可以使用 **log** 命令：**sudo docker logs <container>**。其中 **<**;container> 是容器名称（上例中的"*apicast* "）或容器 ID。您可以使用 **sudo docker ps** 命令获取正在运行的容器及其 ID 和名称的列表。

要停止容器，请运行 **sudo docker stop <container>** 命令。您还可以运行 **sudo docker rm <container>** 命令删除容器。

有关可用命令的更多信息，请参阅 Docker 命令参考。

第 3 章 高级 APICAST 配置

本节介绍暂存环境中 3scale API 网关的高级设置选项。

3.1. 定义 SECRET 令牌

出于安全考虑，从 3scale 网关到 API 后端的任何请求都包含名为 **X-3scale-proxy-secret-token** 的标头。您可以在 Integration 页面的 **Authentication Settings** 中设置此标头的值。

AUTHENTICATION SETTINGS

Host Header

Lets you define a custom `Host` request header. This is needed if your API backend only accepts traffic from a specific host.

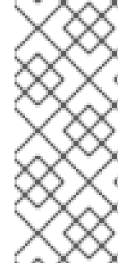
Secret Token: Shared_secret_sent_from_proxy_to_API_backend

Enables you to block any direct developer requests to your API backend; each 3scale API gateway call to your API backend contains a request header called `X-3scale-proxy-secret-token`. The value of this header can be set by you here. It's up to you ensure your backend only allows calls with this secret header.

将 secret 令牌设置为代理和 API 之间的共享机密，以便您可以阻断所有来自网关的 API 请求（如果不希望它们）。当您使用沙盒网关设置流量管理策略时，这会添加额外的安全层来保护您的公共端点。

您的 API 后端必须具有公共可解析域才能使网关正常工作，因此知道您的 API 后端的任何人都可以绕过凭证检查。这不应该成为问题，因为暂存环境中的 API 网关并不适用于生产环境，但始终最好有一个可用的隔离。

3.2. 凭证



注意

user_key、**app_key** 和 **client_secret** 字段有最大长度和保留字符的限制：

只能使用字母数字字符 [0-9, a-z, A-Z], 分号-减号 (-), 5 到 256 个字符。不允许使用空格。

此外，**app_id** 和 **client_id** 的最大长度为 140 个字符。

3scale 中的 API 凭证是 **user_key** 或 **app_id/app_key**，具体取决于您使用的身份验证模式。OpenID Connect 对暂存环境中的 API 网关有效，但它无法在 Integration 页面中进行测试。

但是，您可能想要在 API 中使用不同的凭证名称。在这种情况下，如果使用 API 密钥模式，您需要为 **user_key** 设置自定义名称：

Auth user key: user_key

另外，对于 **app_id** 和 **app_key** ：

App ID parameter: app_id

Name of the parameter that acts of behalf of app id

App Key parameter: app_key

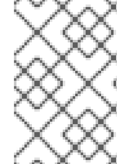
Name of the parameter that acts of behalf of app key

例如，如果这更适合您的 API，您可以将 **app_id** 重命名为 **key**。网关将使用名称 **key**，并将其转换为 **app_id**，然后向 3scale 后端执行授权调用。请注意，新凭证名称必须是字母数字。

您可以决定您的 API 是否在查询字符串（或者正文（如果不是 GET）或标头中传递凭证。

CREDENTIALS LOCATION*

- As HTTP Headers
- As query parameters (GET) or body parameters (POST/PUT/DELETE)



注意

APIcast 在提取凭据时规范化标头名称。这意味着它们不区分大小写，下划线和连字符可以平等对待。例如，如果您将 App Key 参数设置为 **App_Key**，其他值（如 **app-key**） 也被接受，作为有效的 app 键标头。

3.3. 配置错误消息

本节论述了如何配置 APIcast 错误消息。

作为代理，3scale APIcast 使用以下方法管理请求：

- 如果没有错误，APIcast 会将请求从客户端传递到 API 后端服务器，并将 API 响应返回给客户端，无需修改。如果要修改响应，您可以使用 Header Modification 策略。

- 如果 API 响应错误消息，如 **404 Not Found** 或 **400 Bad Request**，APIcast 将返回消息到客户端。但是，如果 APIcast 检测到其他错误，如 **身份验证缺失**，APIcast 会发送错误消息并终止请求。

因此，您可以将这些错误消息配置为由 APIcast 返回：

- **身份验证失败**
  此错误意味着 API 请求不包含有效的凭证，无论是因为伪装凭证，还是因为应用被暂时暂停。另外，这个错误会在指标被禁用时生成，表示其值为 **0**。

- **缺少身份验证**
  当 API 请求不包含任何凭证时，会生成此错误。当用户没有将其凭证添加到 API 请求时会出现这种情况。

- **无匹配项**
  此错误意味着请求与任何映射规则不匹配，因此没有更新指标。这不一定是错误，但意味着用户正在尝试随机路径，或者您的映射规则无法涵盖合法情况。

- **超过用量限制**

此错误意味着客户端达到所请求端点的速率限值。如果请求与多个映射规则匹配，客户端可能会达到多个速率限制。

要配置错误，请按照以下步骤执行：

1. **[Your_product_name] > Integration > Settings**。
2. 在 **Gateway response** 下，选择您要配置的错误类型。
3. 为这些字段指定值：
   - Response Code: HTTP 响应代码。
   - Content-type ： **Content-Type** 标头的值。
   - Response Body：响应消息正文的值。
4. 要保存您的更改，请点击 **Update Product**。

3.4. 配置历史记录

每次单击 **Promote v.[n] to Staging APIcast**时，其中 **[n]** 代表版本号，当前配置保存在 JSON 文件中。暂存网关将利用每个新请求拉取最新的配置。对于每个环境、暂存或生产环境，您可以看到所有之前配置文件的历史记录：

1. 从 **[Your_product_name] > Integration > Configuration**。
2. 点您感兴趣的环境旁的 **Configuration history** 链接： *Staging APIcast* 或 *Production APIcast*。

请注意，无法自动回滚到以前的版本。相反，您可以访问所有配置版本的历史记录及其关联的 JSON 文件。使用这些文件来检查您在任何时间点上部署的配置。如果需要，可以手动重新创建任何部署。

3.5. 调试

设置网关配置很容易，但您可能仍会遇到错误。在这种情况下，网关可以返回有用的调试信息来跟踪错误。

要从 APIcast 获取调试信息，您必须在 API 请求中添加以下标头： **X-3scale-debug: {SERVICE_TOKEN}**，以及与您要到达的 API 服务的服务帐户令牌。

当找到标头且服务令牌有效时，网关会将以下信息添加到响应标头中：

```
X-3scale-matched-rules: /v1/word/{word}.json, /v1
X-3scale-credentials: app_key=APP_KEY&app_id=APP_ID
X-3scale-usage: usage%5Bversion_1%5D=1&usage%5Bword%5D=1
```

X-3scale-matched-rules 表示在用逗号分开的列表中为请求匹配了哪些映射规则。

标头 **X-3scale-credentials** 返回传递给 3scale 后端的凭证。

x-3scale-usage 表示报告到 3scale 后端的使用情况。**usage%5Bversion_1%5D=1&usage%5Bword%5D=1** 是一个 URL 编码的 **usage[version_1]=1&usage[word]=1**，显示 API 请求会对每个方法 (metrics) **version_1** 和 **word** 增加 1 个 hit。

3.6. 路径路由

如果配置了 **APICAST_SERVICES_LIST** 环境变量，APIcast 处理 3scale 帐户上配置的所有 API 服务（或服务子集）。通常，APIcast 会根据请求的主机名将 API 请求路由到适当的 API 服务，方法是将它与 *公共基础 URL* 匹配。找到匹配项的第一个服务用于授权。

路径路由功能允许在多个服务上使用相同的 *公共基础 URL*，并使用请求的路径路由请求。要启用该功能，请将 **APICAST_PATH_ROUTING** 环境变量设置为 **true** 或 **1**。启用后，APIcast 将根据主机名和路径将传入的请求映射到服务。

如果要使用同一 *公共基本 URL* 通过不同的网关公开托管的多个后端服务，可使用此功能。要达到此目的，您可以为每个 API 后端配置多个 API 服务，如 *私有基本 URL* 并启用路径路由功能。

例如，您使用以下方式配置了 3 个服务：

- Service A Public Base URL: **api.example.com** 映射规则： **/a**
- Service B Public Base URL: **api.example.com** 映射规则： **/b**
- Service C Public Base URL: **api.example.com** 映射规则： **/c**

如果路径路由为 disabled (**APICAST_PATH_ROUTING=false**)，对 **api.example.com** 的所有调用都将尝试匹配 Service A。因此，调用 **api.example.com/c** 和 **api.example.com/b** 将失败，并显示 *"No Mapping Rule match"* 错误。

如果路径路由为 enabled (**APICAST_PATH_ROUTING=true**)，则调用将由主机和路径匹配。因此：

- **api.example.com/a** 将路由到 Service A
- **api.example.com/c** 将路由到 Service C
- **api.example2.com/b** 将失败，显示"No Mapping Rule match"错误，即它不匹配 Service B，因为 *公共基础 URL* 不匹配。

如果使用路径路由，您必须确保使用相同 *公共基本 URL* 的不同服务中的映射规则之间没有冲突，例如，仅在一个服务中使用方法 + 路径模式的组合。

第 4 章 APICAST 策略

APIcast 策略是修改 APIcast 操作方式的功能单元。可以启用、禁用和配置策略，以控制它们如何修改 APIcast。使用策略添加默认 APIcast 部署中不可用的功能。您可以创建自己的策略，或使用 Red Hat 3scale 提供的标准策略。

以下主题提供有关标准 APIcast 策略、创建策略链和创建自定义 APIcast 策略的信息。

4.1. 用于更改默认 3SCALE API 管理 APICAST 行为的标准策略

3scale 提供内置的标准策略，它们是修改 APIcast 处理请求和响应的方式的功能单元。您可以启用、禁用或配置策略来控制它们如何修改 APIcast。

详情请参阅 在 3scale 管理门户中启用策略。3scale 提供以下标准策略：

- 3scale API 管理身份验证缓存
- 3scale API 管理 Batcher
- 3scale API 管理推荐器
- Anonymous Access（匿名访问）
- Camel Service
- 条件策略
- 内容缓存
- CORS 请求处理
- 自定义指标
- Echo
- 边缘限制
- 标头修改
- HTTP 状态代码覆盖
- HTTP2 端点
- IP 检查
- JWT 申索检查
- Liquid Context Debug
- 日志记录
- 维护模式
- NGINX Filter
- OAuth 2.0 通用 TLS 客户端身份验证

- OAuth 2.0 令牌内省
- On Fail
- 代理服务
- 速率限制标头
- 响应请求内容限制
- Retry
- RH-SSO/Keycloak 角色检查
- 路由
- SOAP
- TLS 客户端证书验证
- TLS 终止
- Upstream
- 上游连接
- Upstream Mutual TLS
- URL Rewriting
- 使用捕获的 URL 重写
- Websocket

4.1.1. 在 3scale API 管理门户中启用策略

在管理门户中，您可以为每个 3scale API 产品启用一个或多个策略。

先决条件

- 3scale API 产品。

流程

1. 登录 3scale。
2. 在管理门户控制面板中，选择要为其启用策略的 API 产品。
3. 从 **[your_product_name]**，进入到 **Integration > Policies**。
4. 在 POLIC **IES** 部分下，单击 **Add policy**。
5. 选择您要添加的策略，并在任何必填字段中输入值。
6. 单击 **Update Policy Chain** 以保存策略链。

4.1.2. 3scale API 管理身份验证缓存

3scale 身份验证缓存策略会缓存对 APIcast 发出的身份验证调用。您可以选择操作模式来配置缓存操作。

3scale Auth 缓存在以下模式中可用：

1.strict - Cache only authorized calls.

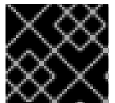
"strict"模式仅缓存授权的调用。如果策略在"strict"模式下运行，并且调用失败或被拒绝，策略会使缓存条目失效。如果后端无法访问，则所有缓存的调用都会被拒绝，无论它们缓存的状态如何。

2.Resilient – Authorize according to last request when backend is down.

"弹性"模式会缓存授权和拒绝调用。如果策略在"弹性"模式下运行，则失败的调用不会使现有的缓存条目无效。如果后端变得不可访问，则命中缓存的调用仍会根据缓存的状态继续获得授权或拒绝。

3.Allow - 当后端停机时，允许所有内容，除非在前和拒绝了。

"Allow"模式会缓存授权和被拒绝的调用。如果策略在"allow"模式下运行，则缓存的调用根据缓存的状态继续被拒绝或允许。但是，任何新调用都将缓存为授权。



重要

在"允许"模式下操作存在安全隐患。在使用"allow"模式时，请考虑以上问题并进行练习。

4.none - 禁用缓存。

"None"模式禁用缓存。如果您希望策略保持活动状态，但不想使用缓存，则此模式非常有用。

配置属性

属性	描述	值	必需？
caching_type	**caching_type** 属性允许您定义缓存将在其中操作的模式。	数据类型：枚举的字符串 [resilient, strict, allow, none]	是

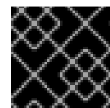
策略对象示例

```
{
  "name": "caching",
  "version": "builtin",
  "configuration": {
    "caching_type": "allow"
  }
}
```

有关如何配置策略的详情，请参考文档中的创建策略链 部分。

4.1.3. 3scale API 管理 Batcher

3scale Batcher 策略提供了标准的 APIcast 授权机制的替代方案，其中为 APIcast 接收的每个 API 请求调用 3scale 后端(Service Management API)。



重要

要使用此策略，您必须在策略链中的 **3scale Batcher** 前放置 **3scale** Batcher。

3scale Batcher 策略会缓存授权状态和批处理使用报告，从而显著减少请求数到 3scale 后端。借助 3scale 批处理器策略，您可以通过降低延迟并提高吞吐量来提高 APIcast 性能。

当 3scale Batcher 策略被启用时，APIcast 会使用以下授权流：

1. 在每个请求中，策略检查是否缓存凭证：
   - 如果缓存了凭据，策略将使用缓存的授权状态，而不是调用 3scale 后端。
   - 如果没有缓存凭据，策略会调用后端，并使用可配置的生存时间(TTL)来缓存授权状态。
2. 策略不立即向 3scale 后端报告与请求对应的使用量，而是累计使用计数器将它们报告到批处理理中的后端。单独的线程在单个调用中报告 3scale 后端的总用量计数器，具有可配置的频率。

3scale Batcher 策略提高了吞吐量，但准确性较低。使用限制和当前利用率存储在 3scale 中，APIcast 只能在向 3scale 后端发出调用时获取正确的授权状态。启用 3scale Batcher 策略时，在一个时间段内 APIcast 不会发送调用到 3scale。在这个时间窗内，发出调用的应用可能会超过定义的限值。

如果吞吐量比速率限制的准确性更重要，则将此策略用于高负载 API。当报告频率和授权 TTL 低于速率限制周期时，3scale Batcher 策略可以带来更好的准确性。例如，如果限制是每天的，并且报告频率和授权 TTL 被配置为几分钟。

3scale Batcher 策略支持以下配置设置：

- **auths_ttl**: 当授权缓存过期时，以秒为单位设置 TTL。
  - 当缓存当前调用的授权时，APIcast 将使用缓存的值。在 **auths_ttl** 参数中设置的时间后，APIcast 会移除缓存并调用 3scale 后端来检索授权状态。
  - 将 **auths_ttl** 参数设置为 **0** 以外的值。将 **auths_ttl** 设置为 **0** 的值可在第一次缓存请求时更新授权计数器，从而导致速率限制无效。
- **batch_report_seconds**: 设置批处理报告 APIcast 发送到 3scale 后端的频率。默认值为 **10** 秒。

4.1.4. 3scale API 管理推荐器

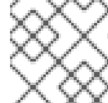
3scale 推荐器策略启用了 *推荐过滤器功能*。当服务策略链中启用策略时，APIcast 将 3scale Referencerer 策略的值发送到 *Service Management API*，作为向上方 *AuthRep* 调用。3scale Referencerer 策略的值在调用中的 **referrer** 参数中发送。

有关 *Referrer Filtering* 如何工作的更多信息，请参阅**身份验证模式**下的 Referrer Filtering 部分。

4.1.5. Anonymous Access（匿名访问）

匿名访问策略会公开一项服务，无需身份验证。例如，这对无法适应发送身份验证参数的传统应用程序很有用。匿名访问策略只支持使用 API Key 和 App Id / App Key 身份验证选项的服务。当为未提供任何凭证的 API 请求启用策略时，APIcast 将授权调用使用策略中配置的默认凭据。若要授权 API 调用，配置有凭据的应用必须存在并且处于活动状态。

使用 Application Plans，您可以配置用于默认凭证的应用程序的速率限值。



注意

在策略链中同时使用这些策略时，您需要将匿名访问策略放在 APIcast 策略的前面。

以下是策略所需的配置属性：

- **auth_type**
  从以下其中一个备选中选择一个值，并确保属性与为 API 配置的身份验证选项对应：
  - **app_id_and_app_key**
    应用程序 ID/应用程序密钥身份验证选项：
  - **user_key**
    用于 API 密钥身份验证选项。
- **app_id** （仅适用于 **app_id_and_app_key** 身份验证类型）
  API 调用中不提供任何凭据时将用于授权的应用程序的 App ID。
- **app_key** （仅适用于 **app_id_and_app_key** 身份验证类型）
  API 调用中不提供任何凭据时将用于授权的应用的 App Key。
- **user_key** （仅适用于 **user_key** auth_type）
  API 调用中不提供任何凭据时将用于授权的应用的 API 密钥。

图 4.1. 匿名访问策略

Anonymous access
builtin - Provides default credentials for unauthenticated requests

This policy allows to expose a service without authentication. It can be useful, for example, for legacy apps that cannot be adapted to send the auth params. When the credentials are not provided in the request, this policy provides the default ones configured. An app_id + app_key or a user_key should be configured.

☑ Enabled

auth_type*

app_id_and_app_key

app_key*

myappid

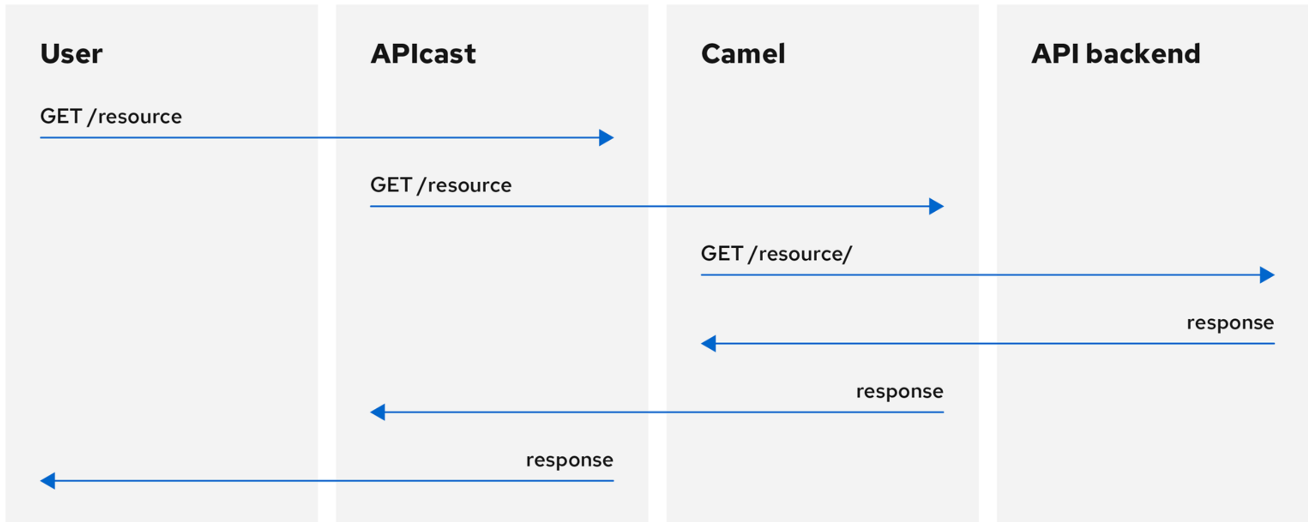
app_id*

secret-app-key-123

4.1.6. Camel Service

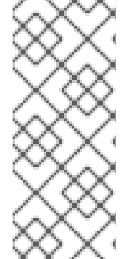
您可以使用 Camel 服务策略定义 *HTTP 代理*，其中 3scale 流量通过定义的 Apache Camel 代理发送。在这种情况下，Camel 充当反向 HTTP 代理，APIcast 将流量发送到 Camel，然后 Camel 会将流量发送到 API 后端。

以下示例显示了流量流：



发送到 3scale 后端的所有 APIcast 流量都不使用 Camel 代理。此策略仅适用于 Camel 代理以及 APIcast 和 API 后端之间的通信。

如果要通过代理发送所有流量，则必须使用 **HTTP_PROXY** 环境变量。



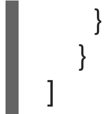
注意

- Camel 服务策略禁用所有负载平衡策略，流量发送到 Camel 代理。
- 如果定义了 **HTTP_PROXY**、**HTTPS_PROXY** 或 **ALL_PROXY** 参数，此策略将覆盖这些值。
- 代理连接不支持身份验证。您可以使用标头修改策略进行身份验证。

配置

以下示例显示了策略链配置：

```
"policy_chain": [
    {
      "name": "apicast.policy.apicast"
    },
    {
      "name": "apicast.policy.camel",
      "configuration": {
          "all_proxy": "http://192.168.15.103:8080/",
          "http_proxy": "http://192.168.15.103:8080/",
          "https_proxy": "http://192.168.15.103:8443/"
```



```
      }
    }
]
```

如果未定义 **http_proxy** 或 **https_proxy**，则使用 **all_proxy** 值。

使用案例示例

Camel 服务策略旨在使用 Apache Camel 在 3scale 中应用更为精细的策略和转换。此策略支持通过 HTTP 和 HTTPS 与 Apache Camel 集成。如需了解更多详细信息，请参阅 第 6 章 *使用 Fuse 中的策略扩展转换 3scale API 管理消息内容*。

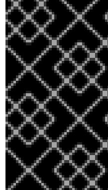
有关使用通用 HTTP 代理策略的详情，请参考 第 4.1.25 节 "代理服务"。

项目示例

请参阅 GitHub 上的 Camel 代理策略 中的 **camel-netty-proxy** 示例。此示例项目显示 HTTP 代理，它将响应正文从 API 后端转换为大写。

4.1.7. 条件策略

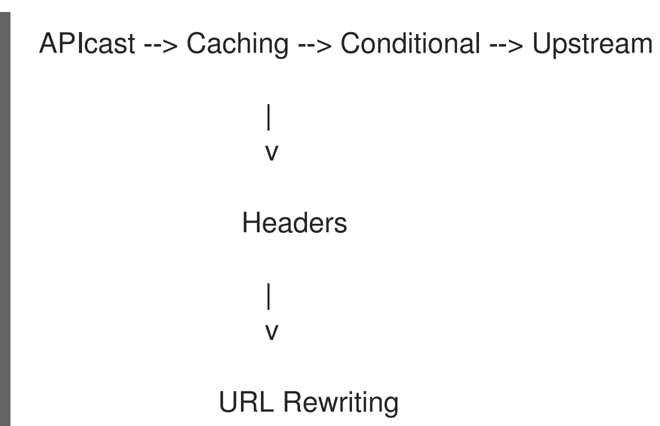
条件策略与其他 APIcast 策略不同，因为它包含一系列策略。它定义在每个 *nginx 阶段* 评估的条件，如 *访问*、*重写* 和 *日志* 等。当条件为 true 时，条件策略会为其链中包含的每个策略运行该阶段。



重要

APIcast Conditional Policy 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

以下示例假定 Conditional Policy 定义以下条件： **请求方法为 POST**。



```
APIcast --> Caching --> Conditional --> Upstream

                           |
                           v

                        Headers

                           |
                           v

                      URL Rewriting
```

在这种情况下，当请求是 **POST** 时，每个阶段的执行顺序如下：

1. APIcast
2. Caching
3. Headers
4. URL Rewriting
5. Upstream

如果没有 **POST** 请求，每个阶段的执行顺序如下：

1. APIcast
2. Caching
3. Upstream

Conditions

确定在 Conditional Policy 链中运行的策略是否可以使用 *JSON* 表达或使用 liquid 模板。

本例检查请求路径是否为 **/example_path** ：

```
{
  "left": "{{ uri }}",
  "left_type": "liquid",
  "op": "==",
  "right": "/example_path",
  "right_type": "plain"
}
```

左和右边的运算对象都可以评估为 liquid 或纯字符串。纯文本字符串是默认值。

您可以将操作与 **and** 或 **or** 组合使用。此配置检查与上例相同，外加 **Backend** 标头值：

```
{
  "operations": [
    {
      "left": "{{ uri }}",
      "left_type": "liquid",
      "op": "==",
      "right": "/example_path",
      "right_type": "plain"
    },
    {
      "left": "{{ headers['Backend'] }}",
      "left_type": "liquid",
      "op": "==",
      "right": "test_upstream",
      "right_type": "plain"
    }
  ],
  "combine_op": "and"
}
```

如需了解更多详细信息，请参阅 策略配置模式。

liquid 支持的变量

- uri
- 主机
- remote_addr

- headers['Some-Header']

更新的变量列表可在此处找到： ngx_variable.lua

这个示例在请求的 **Backend** 标头是 *staging* 时执行上游策略：

```
{
  "name":"conditional",
  "version":"builtin",
  "configuration":{
    "condition":{
      "operations":[
        {
          "left":"{{ headers['Backend'] }}",
          "left_type":"liquid",
          "op":"==",
          "right":"staging"
        }
      ]
    },
    "policy_chain":[
      {
        "name":"upstream",
        "version": "builtin",
        "configuration":{
          "rules":[
            {
              "regex":"/",
              "url":"http://my_staging_environment"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.8. 内容缓存

内容缓存策略允许您根据自定义条件启用和禁用缓存。这些条件只能应用于客户端请求，因为策略中无法使用上游响应。

当内容缓存策略位于策略链中时，APIcast 会在发送上游请求前将 **HEAD** 请求发送到 **GET** 请求。如果您不想进行此转换，请不要将内容缓存策略添加到策略链中。

如果发送了 cache-control 标头，它将优先于 APIcast 设定的超时。

如果 Method 是 GET，则以下示例配置将缓存响应。

配置示例

```
{
  "name": "apicast.policy.content_caching",
  "version": "builtin",
  "configuration": {
```

```
    "rules": [
     {
      "cache": true,
      "header": "X-Cache-Status-POLICY",
      "condition": {
       "combine_op": "and",
       "operations": [
        {
         "left": "{{method}}",
         "left_type": "liquid",
         "op": "==",
         "right": "GET"
        }
       ]
      }
     }
    ]
   }
  }
```

支持的配置

- 将以下任一方法的内容缓存策略设置为禁用：**POST**、**PUT** 或 **DELETE**。
- 如果一条规则匹配，并且它启用了缓存，则执行将停止并且不会禁用。这里按优先级排序非常重要。

上游响应标头

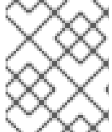
NGINX **proxy_cache_valid** 指令信息只能全局设置，使用 **APICAST_CACHE_STATUS_CODES** 和 **APICAST_CACHE_MAX_TIME**。如果您的上游需要与超时相关的不同行为，请使用 **Cache-Control** 标头。

4.1.9. CORS 请求处理

通过允许您指定以下指定来控制 CORS 行为，可通过交叉资源共享(CORS)请求处理策略来控制 CORS：

- 允许的标头
- 允许的方法
- 允许的源标头
- 允许的凭证
- 最长期限

CORS 请求处理策略将阻止所有未指定的 CORS 请求。



注意

当在策略链中使用这两个策略时，您需要将 CORS Request 处理策略放在 APIcast 策略的前面。

配置属性

属性	描述	值	必需？
allow_headers	**allow_headers** 属性是一个数组，您可以在其中指定允许哪些 CORS 标头。	data type：字符串数组，必须是 CORS 标头	否
allow_methods	**allow_methods** 属性是一个数组，您可以在其中指定 APIcast 允许的 CORS 方法。	data type：枚举字符串的数组 [GET, HEAD, POST, PUT, DELETE, PATCH, OPTIONS, TRACE, CONNECT]	否
allow_origin	**allow_origin** 属性允许您指定原始域 APIcast 允许	数据类型：字符串	否
allow_credentials	**allow_credentials** 属性允许您指定 APIcast 是否允许带有凭证的 CORS 请求	数据类型：布尔值	否
max_age	**max_age** 属性允许您设置 preflight 请求结果的缓存时长	数据类型：整数	否

策略对象示例

```
{
  "name": "cors",
 "version": "builtin",
 "configuration": {
   "allow_headers": [
     "App-Id", "App-Key",
     "Content-Type", "Accept"
   ],
   "allow_credentials": true,
   "allow_methods": [
     "GET", "POST"
   ],
   "allow_origin": "https://example.com",
   "max_age" : 200
  }
}
```

有关如何配置策略的详情，请参考 3scale API 管理门户中的修改策略链。

4.1.10. 自定义指标

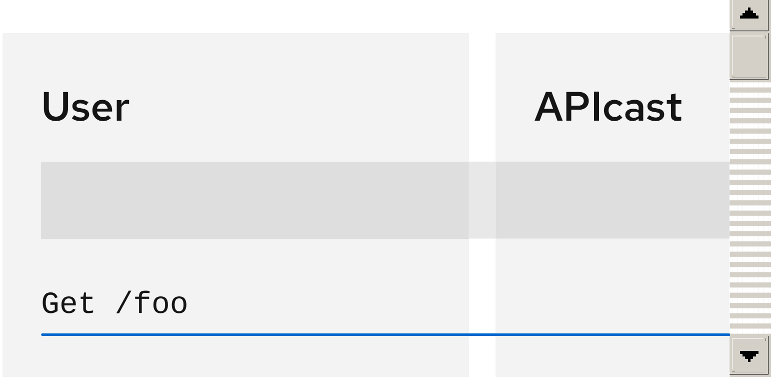
自定义 Metrics 策略会添加可用性，以在上游 API 发送的响应后添加指标。此策略的主要用例是根据响应代码状态、标头或不同的 NGINX 变量添加指标。

自定义指标的限制

- 当在将请求发送到上游 API 之前进行身份验证时，将对后端进行第二次调用，以向上游 API 报告新指标。
- 此策略不适用于批处理策略。
- 需要在管理门户中创建指标，然后策略才会推送指标值。

请求流示例

下图显示了未缓存身份验证的请求流示例，以及身份验证缓存时的流。



配置示例

如果上游 API 返回 400 状态，此策略会按标头递增来递增指标错误：

```
{
  "name": "apicast.policy.custom_metrics",
  "configuration": {
    "rules": [
      {
        "metric": "error",
        "increment": "{{ resp.headers['increment'] }}",
        "condition": {
          "operations": [
            {
              "right": "{{status}}",
              "right_type": "liquid",
              "left": "400",
              "op": "=="
            }
          ],
          "combine_op": "and"
        }
      }
    ]
  }
}
```

如果上游 API 返回 200 状态，则此策略使用 status_code 信息递增 hits 指标：

```
{
  "name": "apicast.policy.custom_metrics",
  "configuration": {
```

```
      "rules": [
        {
          "metric": "hits_{{status}}",
          "increment": "1",
          "condition": {
            "operations": [
              {
                "right": "{{status}}",
                "right_type": "liquid",
                "left": "200",
                "op": "=="
              }
            ],
            "combine_op": "and"
          }
        }
      ]
    }
  }
```

4.1.11. Echo

Echo 策略将传入请求打印回客户端，以及可选的 HTTP 状态代码。

配置属性

属性	描述	值	必需？
status	Echo 策略将返回到客户端的 HTTP 状态代码	数据类型：整数	否
exit	指定 Echo 策略将使用的退出模式。**request** 退出模式将停止处理传入请求。**set** exit 模式跳过重写阶段。	data type：枚举字符串 [request, set]	是

策略对象示例

```
{
  "name": "echo",
  "version": "builtin",
  "configuration": {
    "status": 404,
    "exit": "request"
  }
}
```

有关如何配置策略的详情，请参考 文档中的 3scale API 管理 部分创建策略链。

4.1.12. 边缘限制

Edge Limiting 策略旨在为发送到后端 API 的流量提供灵活的速率限制，并可与默认的 3scale 授权一起使用。策略支持的用例的一些示例包括：

- 最终用户速率限制： 对在请求的 Authorization 标头中的 JWT 令牌的 **sub** (subject)声明值的限制。这配置为 **{{ jwt.sub }}**。
- 请求每秒(RPS)速率限制.
- 每个服务的全局速率限值：每个服务应用限制，而不是每个应用程序。
- 并发连接限制 ：设置允许的并发连接数。

限制类型

该策略支持 lua-resty-limit-traffic 库提供的以下类型的限制：

- leaky_bucket_limiters
  基于泄漏 bucket 算法， 它基于平均请求数加上最大突发大小。
- fixed_window_limiters
  根据固定的时间窗口：最后 n 秒。
- connection_limiters
  基于连接的并发数量。

您可以通过服务或全局限制来限制任何限制。

限制定义

这些限制有一个键，对用于定义限制的实体进行编码，如 IP 地址、服务、端点、标识符、特定标头的值和实体。此密钥在限制器的 **key** 参数中指定。

key 是由以下属性定义的对象：

- name
  定义密钥的名称。它在范围内必须是唯一的。
- scope
  定义密钥的范围。支持的范围有：
  - 每个影响一个服务（**service**）的服务范围。
  - 影响所有服务（**global**）的全局范围。
- name_type _ 定义如何评估 **name** 值：
  - 纯文本（**plain**）
  - Liquid（**liquid**）

每个限制也具有一些因类型而异的参数：

- leaky_bucket_limiters
  速率，突发.
  - rate
    定义可以在不延迟的情况下每秒发出多少个请求。

  - burst
    定义每秒的请求量可能超过允许的速率。为超过由 **rate** 指定的允许费率的请求引入人为延迟。在超过 **burst** 中定义的每秒请求数后，请求将被拒绝。

- fixed_window_limiters
  计算，**窗口.count** 定义在 **window** 中定义的每秒数可以发出多少个请求。

- connection_limiters
  conn,**burst**,**delay**.

  - conn
    定义允许的最大并发连接数。它允许通过每秒 **burst** 连接超过该数量。

  - delay
    定义延迟超过限制的连接的秒数。

例子

- 每分钟允许 10 个请求到 service_A：

```
{
  "key": { "name": "service_A" },
  "count": 10,
  "window": 60
}
```

- 允许 100 个连接，突发 10，延迟 1 秒：

```
{
  "key": { "name": "service_A" },
  "conn": 100,
  "burst": 10,
  "delay": 1
}
```

您可以为每个服务定义多个限制。如果定义了多个限制，则在至少达到一个限值时请求可以被拒绝或延迟。

移动模板

Edge Limiting 策略允许通过在键中支持 Liquid 变量来指定动态密钥的限制。为此，密钥的 **name_type** 参数必须设置为 **random**，**name** 参数可以使用 Liquid 变量。例如，客户端 IP 地址 **{{ remote_addr }}** 或 JWT 令牌 **sub** 声明的 **{{ jwt.sub }}**。

示例

```
{
  "key": { "name": "{{ jwt.sub }}", "name_type": "liquid" },
  "count": 10,
  "window": 60
}
```

有关 Liquid 支持的详情请参考 第 5.1 节 “在策略中使用变量和过滤器”。

应用条件

每个限制器必须具有定义何时应用限制器的条件。条件在限制器的 **condition** 属性中指定。

condition 由以下属性定义：

- combine_op
  应用到操作列表的布尔值运算符。支持 **or** 和 **and**。
- **操作**
  需要评估的条件列表。每个操作都由一个具有以下属性的对象表示：
  - left
    操作的左侧部分.
  - left_type
    如何评估 **left** 属性（名文或 liquid）。
  - right
    操作的右侧部分.
  - right_type
    如何评估 **right** 的属性（名文或 liquid）。
  - op
    Operator 在左侧和右侧部分之间应用。支持以下两个值： **==** (等于)和 **!=** （不等于）。

示例

```
"condition": {
  "combine_op": "and",
  "operations": [
    {
      "op": "==",
      "right": "GET",
      "left_type": "liquid",
      "left": "{{ http_method }}",
      "right_type": "plain"
    }
  ]
}
```

配置速率限制计数器的存储

默认情况下，Edge Limiting 策略将 OpenResty 共享字典用于速率限制计数器。但是，您可以使用外部 Redis 服务器，而不是共享的字典。这在部署多个 APIcast 实例时非常有用。您可以使用 **redis_url** 参数配置 Redis 服务器。

错误处理

限制器支持以下参数来配置错误的处理方式：

- limits_exceeded_error
  指定超出配置的限制时将返回到客户端的错误状态代码和消息。应配置以下参数：
  - status_code
    超过限值时请求的状态代码。默认：**429**。

- error_handling
  使用以下选项指定如何处理错误：
  - exit
    停止处理请求并返回错误消息。
  - log
    完成处理请求并返回输出日志。

- configuration_error
  指定在配置不正确时将返回到客户端的错误状态代码和消息。应配置以下参数：
  - status_code
    存在配置问题时的状态代码。默认：**500**。
  - error_handling
    使用以下选项指定如何处理错误：
    - exit
      停止处理请求并返回错误消息。
    - log
      完成处理请求并返回输出日志。

4.1.13. 标头修改

标头修改策略允许您修改现有标头，或者定义额外的标头以添加到或从传入的请求或响应中删除。您可以修改响应和请求标头。

Header 修改策略支持以下配置参数：

- Request（请求）
  要应用到请求标头的操作列表
- 响应
  应用到响应标头的操作列表

每个操作都由以下参数组成：

- **op**：指定要应用的操作。**add** 操作会为现有标头添加一个值。这个 **set** 操作会创建一个标头和值，如果已存在该标头的值，则会覆盖现有的标头值。**push** 操作会创建一个标头和值，但如果已存在，则不会覆盖现有的标头值。相反，**push** 会将值添加到现有标头中。**delete** 操作会删除标头。
- **header**：指定要创建或修改的标头，可以是可用作标头名称的任何字符串，如 **Custom-Header**。
- **value_type**：定义如何评估标头值，可以是 **plain**（纯文本）或 **liquid**（作为 Liquid 模板评估）。更多信息请参阅 第 5.1 节 “在策略中使用变量和过滤器”。
- **value**: 指定用于标头的值。对于值类型"liquid"，值应当采用 **{{ variable_from_context }}** 格式。删除时不需要.

策略对象示例

```
{
```

```
  "name": "headers",
  "version": "builtin",
  "configuration": {
    "response": [
      {
        "op": "add",
        "header": "Custom-Header",
        "value_type": "plain",
        "value": "any-value"
      }
    ],
    "request": [
      {
        "op": "set",
        "header": "Authorization",
        "value_type": "plain",
        "value": "Basic dXNlcm5hbWU6cGFzc3dvcmQ="
      },
      {
        "op": "set",
        "header": "Service-ID",
        "value_type": "liquid",
        "value": "{{service.id}}"
      }
    ]
  }
}
```

有关如何配置策略的详情，请参考 文档中的 3scale API 管理 部分创建策略链。

4.1.14. HTTP 状态代码覆盖

作为 API 供应商，您可以将 HTTP Status Code Overwrite 策略添加到 API 产品中。此策略允许您将上游响应代码改为您指定的响应代码。3scale 将 HTTP Status Code Overwrite 策略应用到从上游服务发送的响应代码。换句话说，当 3scale 公开的 API 返回不适合您情况的代码时，您可以配置 HTTP Status Response Code Overwrite 策略，将代码更改为对应用程序有意义的响应代码。

在策略链中，任何生成要更改的响应代码的策略都必须在 HTTP 状态代码 Overwrite 策略之前。如果没有生成您要更改的 Status Codes 的策略，则 HTTP Status Code Overwrite 策略的策略链位置无关紧要。

在管理门户中，将 HTTP Status Code Overwrite 策略添加到产品的策略链中。在策略链中，单击策略以指定您要更改的上游响应代码以及您要返回的响应代码。单击您要覆盖的每个额外上游响应代码的加号。例如，您可以使用 HTTP 状态代码覆盖策略将上游 **201**、"Created"、响应代码改为 **200**，"OK"，响应代码。

在超过内容限制时，响应代码可能要更改的另一个示例就是响应。当响应代码为 **414** （请求 URI 过长）时，上游可能会返回 **413**(载荷过大)。

在管理门户中添加 HTTP Status Code Overwrite 策略的一种替代方案是将 3scale API 与策略链配置文件搭配使用。

示例

策略链配置文件中的以下 JSON 配置将覆盖两个上游响应代码：

```
{
```

```
    "name": "statuscode_overwrite",
    "version": "builtin",
    "configuration": {
      "http_statuses": [
        {
          "upstream": 201,
          "apicast": 200
        },
        {
          "upstream": 413,
          "apicast": 414
        }
      ]
    }
  }
```

4.1.15. HTTP2 端点

HTTP2 端点策略启用发送请求和 APIcast 的消费者应用之间的 HTTP/2 协议和远程过程调用(gRPC)连接。当 HTTP2 端点策略位于产品的策略链中时，整个通信流（从向上游服务发出请求的消费者应用程序到 APIcast）可以使用 HTTP/2 协议和 gRPC。

当 HTTP2 端点策略位于策略链中时：

- 请求身份验证必须是 JSON Web 令牌或 **App_ID** 和 **App_Key** 对的方法。不支持 API 密钥身份验证。
- gRPC 端点终止传输层安全(TLS)。
- HTTP2 端点策略必须在 3scale APIcast 策略之前。
- 上游服务的后端可以实施 HTTP/1.1 纯文本或传输层安全(TLS)。
- 策略链还必须包含 TLS 终止策略。

  APIcast 配置策略链示例：

  ```
  "policy_chain": [
    { "name": "apicast.policy.tls" },
    { "name": "apicast.policy.grpc" },
    { "name": "apicast.policy.apicast" }
  ]
  ```

4.1.16. IP 检查

IP 检查策略用于根据 IP 列表拒绝或允许请求。

配置属性

属性	描述	数据类型	必需？

属性	描述	数据类型	必需？
check_type	**check_type** 属性有两个可能的值，即 **whitelist** 或 **blacklist**。**blacklist** 将拒绝来自 IP 的所有请求。**whitelist** 将拒绝*不是*来自列表中的 IP 的所有请求。	字符串，必须是 **whitelist** 或 **blacklist**	是
ips	**ips** 属性允许您指定 IP 地址列表以列入白名单或黑名单。可以使用单一 IP 和 CIDR 范围。	字符串数组，必须是有效的 IP 地址	是
error_msg	**error_msg** 属性允许您配置在请求被拒绝时返回的错误消息。	字符串	否
client_ip_sources	**client_ip_sources** 属性允许您配置如何检索客户端 IP。默认情况下使用最后一个调用者 IP。其他选项有 **X-Forwarded-For** 和 **X-Real-IP**。	字符串数组，有效选项为 **X-Forwarded-For**, **X-Real-IP**, **last_caller**。	否

策略对象示例

```
{
  "name": "ip_check",
  "configuration": {
    "ips": [ "3.4.5.6", "1.2.3.0/4" ],
    "check_type": "blacklist",
    "client_ip_sources": ["X-Forwarded-For", "X-Real-IP", "last_caller"],
    "error_msg": "A custom error message"
  }
}
```

有关如何配置策略的详情，请参考 文档中的 3scale API 管理 部分创建策略链。

4.1.17. JWT 申索检查

JWT 声明基于 JSON Web Token(JWT)声明，您可以通过定义新规则来阻止资源目标和方法。

关于 JWT 申索检查策略

为了基于 JWT 声明的值进行路由，您需要链中的策略验证 JWT，并将声明存储在策略共享的上下文中。

如果 JWT Claim Check 策略正在阻止资源和方法，该策略也会验证 JWT 操作。另外，如果方法资源不匹配，则请求将继续至后端 API。

示例：如果是 GET 请求，JWT 需要将角色声明设为 admin（如果请求将被拒绝）。另一方面，任何非 GET 请求都不会验证 JWT 操作，因此允许 POST 资源而无需 JWT 约束。

```
{
  "name": "apicast.policy.jwt_claim_check",
  "configuration": {
      "error_message": "Invalid JWT check",
      "rules": [
          {
              "operations": [
                  {"op": "==", "jwt_claim": "role", "jwt_claim_type": "plain", "value": "admin"}
              ],
              "combine_op":"and",
              "methods": ["GET"],
              "resource": "/resource",
              "resource_type": "plain"
          }
      ]
  }
}
```

在策略链中配置 JWT 声明检查策略

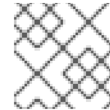
在策略链中配置 JWT 声明检查策略：

- 您需要有权访问 3scale 安装。
- 您需要等待所有部署完成。

配置策略

1. 要将 JWT 声明检查策略添加到您的 API 中，请按照 3scale API 管理管理门户中启用策略 中所述的步骤，然后选择 JWT 声明检查。
2. 单击 **JWT 声明检查** 链接。
3. 若要启用该策略，选中 **Enabled** 复选框。
4. 若要添加规则，可单击加号 **+** 图标。
5. 指定 **resource_type**。
6. 选择 operator。
7. 指明规则控制的**资源**。
8. 要添加允许的方法，请单击加号 **+** 图标。
9. 键入错误消息，以便在流量被阻止时向用户显示。
10. 使用 JWT Claim Check 设置完 API 后，单击 **Update Policy**。
    单击对应部分中的加号 **+** 图标，您可以添加更多资源类型和允许的方法。
11. 单击 **Update Policy Chain** 以保存您的更改。

4.1.18. Liquid Context Debug



注意

Liquid Context Debug 策略仅用于开发环境中而不是生产环境中的调试用途。

此策略使用 **JSON** 响应 API 请求，其中包含上下文中可用的对象和值，并可用于评估 Liquid 模板。与 3scale APIcast 或 上游策略结合使用时，必须将 Liquid Context Debug 放置在策略链中才能正常工作。为避免循环引用，该策略仅包含重复的对象，并将其替换为 stub 值。

启用策略时 APIcast 返回的值示例：

```
{
  "jwt": {
    "azp": "972f7b4f",
    "iat": 1537538097,
    ...
    "exp": 1537574096,
    "typ": "Bearer"
  },
  "credentials": {
    "app_id": "972f7b4f"
  },
  "usage": {
    "deltas": {
      "hits": 1
    },
    "metrics": [
      "hits"
    ]
  },
  "service": {
    "id": "2",
    ...
  }
  ...
}
```

4.1.19. 日志记录

Logging 策略有两个目的：

- 启用和禁用访问日志输出：
- 要为每个服务创建自定义访问日志格式，并能够设置编写自定义访问日志的条件：

您可以将 Logging 策略与访问日志位置的全局设置合并。设置 **APICAST_ACCESS_LOG_FILE** 环境变量，以配置 APIcast 访问日志的位置。默认情况下，此变量设置为 **/dev/stdout**，这是标准输出设备。有关全局 APIcast 参数的详情，请参阅 APIcast 环境变量。

另外，日志记录策略具有以下功能：

- 此策略仅支持 **enable_access_logs** 配置参数。
- 要启用访问日志，请选择 **enable_access_logs** 参数或禁用 Logging 策略。
- 为 API 禁用访问日志：

1. 启用策略。
2. 清除 **enable_access_logs** 参数。
3. 单击 **Submit** 按钮。

- 默认情况下，策略链中不启用此策略。

4.1.19.1. 为所有 API 配置日志策略

APICAST_ENVIRONMENT 可用于加载使策略全局应用到所有 API 产品的配置。以下是如何实现此目标的示例。APICAST_ENVIRONMENT 用于指向文件的路径，这取决于部署、模板或操作器的类型。

要在全局范围内配置日志记录策略，根据您的部署类型考虑以下内容：

- 对于基于模板的部署：需要通过 ConfigMap 和 VolumeMount 将文件挂载到容器中。
- 对于 3scale 基于 operator 的部署：
  - 在 3scale 2.11 及以前的版本中，需要通过 ConfigMap 和 VolumeMount 将文件挂载到容器中。
  - 从 3scale 2.11 开始，需要使用 APIManager 自定义资源(CR)中引用的 secret。
- 对于 APIcast operator 部署：
  - 在以前的版本中 3scale 2.11 无法配置它。
  - 从 3scale 2.11 开始，需要使用 APIManager CR 中引用的 secret。
- 对于在 Docker 上部署的 APIcast 自我管理，需要将文件挂载到容器器。

日志记录选项有助于避免出现 API 中未正确格式化的日志问题。

以下是在所有服务中载入的策略示例：

custom_env.lua file

```
local cjson = require('cjson')
local PolicyChain = require('apicast.policy_chain')
local policy_chain = context.policy_chain

local logging_policy_config = cjson.decode([[
{
  "enable_access_logs": false,
  "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}"
}
]])

policy_chain:insert( PolicyChain.load_policy('logging', 'builtin', logging_policy_config), 1)

return {
  policy_chain = policy_chain,
  port = { metrics = 9421 },
}
```

4.1.19.1.1. 通过在容器中通过 ConfigMap 和 VolumeMount 挂载文件系统来配置所有 API 的日志记录策略

1. 使用 **custom_env.lua** 文件创建 ConfigMap：

```
$ oc create configmap logging --from-file=/path/to/custom_env.lua
```

2. 为 ConfigMap 挂载一个卷，如 **apicast-staging** ：

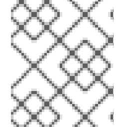
```
$ oc set volume dc/apicast-staging --add --name=logging --mount-path=/opt/app-root/src/config/custom_env.lua --sub-path=custom_env.lua -t configmap --configmap-name=logging
```

3. 设置环境变量：

```
$ oc set env dc/apicast-staging APICAST_ENVIRONMENT=/opt/app-root/src/config/custom_env.lua
```

4.1.19.1.2. 使用 APIManager CR 中引用的 secret 配置日志记录策略

基于 operator 的部署中从 3scale 2.11 开始，将日志策略配置为 secret，并在 APIManager CR 中引用 secret。



注意

以下流程只适用于 3scale Operator。但是，您可以使用这些步骤以类似的方式配置 APIcast operator。

先决条件

- 一个或多个使用 Lua 的代码定制的环境。

流程

1. 使用自定义环境内容创建 secret：

```
$ oc create secret generic custom-env --from-file=./custom_env.lua
```

2. 使用 APIcast 自定义环境配置和部署 APIManager CR：

apimanager.yaml content:

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-apicast-custom-environment
spec:
  apicast:
    productionSpec:
      customEnvironments:
        - secretRef:
            name: custom-env
    stagingSpec:
```

```
customEnvironments:
  - secretRef:
      name: custom-env
```

3. 部署 APIManager CR：

```
$ oc apply -f apimanager.yaml
```

如果 secret 不存在，Operator 会将 CR 标记为失败。对 secret 的更改将需要重新部署 pod/容器才能反映在 APIcast 中。

更新自定义虚拟环境

如果您需要修改自定义环境内容，有两个选项：

- 建议：使用不同的名称创建另一个 secret 并更新 APIManager CR 字段：

  ```
  customEnvironments[].secretRef.name
  ```

  Operator 会触发滚动更新载入新的自定义环境内容。

- 更新现有的 secret 内容，并重新部署 APIcast 将 **spec.apicast.productionSpec.replicas** 或 **spec.apicast.stagingSpec.replicas** 设置为 0， 然后切换到以前的值。

4.1.19.1.3. 为 Docker 上部署的 APIcast 自我管理的所有 API 配置日志策略

使用以下 docker 命令挂载 custom_env.lua 来运行 APIcast：

```
docker run --name apicast --rm -p 8080:8080 \
  -v $(pwd):/config \
  -e APICAST_ENVIRONMENT=/config/custom_env.lua \
  -e THREESCALE_PORTAL_ENDPOINT=https://ACCESS_TOKEN@ADMIN_PORTAL_DOMAIN \
  quay.io/3scale/apicast:master
```

以下是要考虑的 docker 命令的主要概念：

- 将当前的 Lua 文件共享至容器 **-v $(pwd):/config**。
- 将 APICAST_ENVIRONMENT 变量设置为存储在 **/config** 目录中的 Lua 文件。

4.1.19.2. 日志记录策略示例

以下是日志记录策略示例，请考虑以下事项：

- 如果启用了 **custom_logging** 或 **enable_json_logs** 属性，则会禁用默认访问日志。
- 如果启用了 **enable_json_logs**，将省略 **custom_logging** 字段。

禁用访问日志

```
{
 "name": "apicast.policy.logging",
 "configuration": {
```

```
    "enable_access_logs": false
  }
}
```

启用自定义访问日志

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "[{{time_local}}] {{host}}:{{server_port}} {{remote_addr}}:{{remote_port}} \"
{{request}}\" {{status}} {{body_bytes_sent}} ({{request_time}}) {{post_action_impact}}",
  }
}
```

使用服务标识符启用自定义访问日志

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.serializable.name}}",
  }
}
```

使用 JSON 格式配置访问日志

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "enable_json_logs": true,
    "json_object_config": [
      {
        "key": "host",
        "value": "{{host}}",
        "value_type": "liquid"
      },
      {
        "key": "time",
        "value": "{{time_local}}",
        "value_type": "liquid"
      },
      {
        "key": "custom",
        "value": "custom_method",
        "value_type": "plain"
      }
    ]
  }
}
```

仅为成功请求配置自定义访问日志

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}",
    "condition": {
      "operations": [
        {"op": "==", "match": "{{status}}", "match_type": "liquid", "value": "200"}
      ],
      "combine_op": "and"
    }
  }
}
```

自定义访问日志，其中响应状态与 200 或 500匹配

```
{
  "name": "apicast.policy.logging",
  "configuration": {
    "enable_access_logs": false,
    "custom_logging": "\"{{request}}\" to service {{service.id}} and {{service.name}}",
    "condition": {
      "operations": [
        {"op": "==", "match": "{{status}}", "match_type": "liquid", "value": "200"},
        {"op": "==", "match": "{{status}}", "match_type": "liquid", "value": "500"}
      ],
      "combine_op": "or"
    }
  }
}
```

4.1.19.3. 有关自定义日志记录的附加信息

对于自定义日志记录，您可以使用 Liquid 模板和导出的变量。这些变量包括：

- NGINX default 指令变量：log_format.例如： **{{remote_addr}}**。
- 响应和请求标头：
  - **{{req.headers.FOO}}**: 要在请求中获取 FOO 标头。
  - **{{res.headers.FOO}}**: 要检索响应上的 FOO 标头。
- 服务信息，如 **{{service.id}}**，以及这些参数提供的所有服务属性：
  - THREESCALE_CONFIG_FILE
  - THREESCALE_PORTAL_ENDPOINT

4.1.20. 维护模式

Maintenance Mode 策略允许您使用指定状态代码和消息拒绝传入的请求。它对于维护期或临时阻止 API 非常有用。

配置属性

下表列出了可能的属性和默认值：

属性	value	default	描述
status	整数，*可选*	503	响应代码
message	字符串，*可选*	503 服务不可用 - 维护	响应消息

维护模式策略示例

```
{
  "policy_chain": [
    {"name": "maintenance-mode", "version": "1.0.0",
    "configuration": {"message": "Be back soon..", "status": 503} },
  ]
}
```

为特定上游应用维护模式

```
{
    "name": "maintenance_mode",
    "version": "builtin",
    "configuration": {
        "message_content_type": "text/plain; charset=utf-8",
        "message": "Echo API /test is currently Unavailable",
        "condition": {
            "combine_op": "and",
            "operations": [
                {
                    "left_type": "liquid",
                    "right_type": "plain",
                    "op": "==",
                    "left": "{{ original_request.path }}",
                    "right": "/test"
                }
            ]
        },
        "status": 503
    }
}
```

有关如何配置策略的详情，请参考 文档中的 3scale API 管理 部分创建策略链。

4.1.21. NGINX Filter

NGINX 会自动检查某些请求标头并在无法验证这些标头时拒绝请求。例如，NGINX 拒绝具有 NGINX 无法验证的 **If-Match** 标头的请求。如果您希望 NGINX 跳过特定标头的验证，请添加 NGINX Filter 策略。

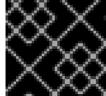
添加 NGINX Filter 策略时，您要为需要 NGINX 跳过验证指定一个或多个请求标头。对于您指定的每个标头，您可以指定是否在请求中保留标头。例如，以下 JSON 代码添加 NGINX 过滤器策略，以便它跳过 **if-Match** 标头的验证，但在转发到上游服务器的请求中保留 **If-Match** 标头。

```
{ "name": "apicast.policy.nginx_filters",
  "configuration": {
    "headers": [
      {"name": "If-Match", "append": true}
    ]
  }
}
```

下一个示例还跳过了 **If-Match** 标头的验证，但此代码指示 NGINX 在向上游服务器发送请求前删除 **If-Match** 标头。

```
{ "name": "apicast.policy.nginx_filters",
  "configuration": {
    "headers": [
      {"name": "If-Match", "append": false}
    ]
  }
}
```

无论您是否将指定的标头附加到上游服务器的请求中，当 NGINX 无法验证您指定的标头时，您避免了 NGINX **412** 响应代码。



重要

为标头修改策略 和 NGINX 过滤器策略指定相同的标头是潜在的冲突来源。

4.1.22. OAuth 2.0 通用 TLS 客户端身份验证

此策略为每个 API 调用执行 OAuth 2.0 通用 TLS 客户端身份验证。

OAuth 2.0 通用 TLS 客户端身份验证策略 **JSON** 示例如下所示：

```
{
  "$schema": "http://apicast.io/policy-v1/schema#manifest#",
  "name": "OAuth 2.0 Mutual TLS Client Authentication",
  "summary": "Configure OAuth 2.0 Mutual TLS Client Authentication.",
  "description": ["This policy executes OAuth 2.0 Mutual TLS Client Authentication ",
    "(https://tools.ietf.org/html/draft-ietf-oauth-mtls-12) for every API call."
  ],
  "version": "builtin",
  "configuration": {
    "type": "object",
    "properties": { }
  }
}
```

4.1.23. OAuth 2.0 令牌内省

OAuth 2.0 Token Introspection 策略允许使用令牌签发者(Red Hat Single Sign-On)的 Token Introtion Endpoint(Red Hat Single Sign-On)验证选项来验证用于通过 OpenID Connect(OIDC)身份验证选项的 JSON Web Token Token 令牌(JWT)令牌。

APIcast 支持 **auth_type** 字段中的以下身份验证类型，以确定 Token Introspection Endpoint 和调用此端点时使用的凭证 APIcast：

- **use_3scale_oidc_issuer_endpoint**: APIcast 使用客户端凭证、*客户端 ID* 和*客户端 Secret*，以及来自 Service Integration 页面上配置的 OIDC Issuer 设置中的 Token Introspection Endpoint。APIcast 从 **token_introspection_endpoint** 字段发现 Token Introspection 端点。此字段位于 OIDC 签发者返回的 **.well-known/openid-configuration** 端点中。
身份验证类型被设置为 **use_3scale_oidc_issuer_endpoint**:

```
"policy_chain": [
…
 {
   "name": "apicast.policy.token_introspection",
   "configuration": {
     "auth_type": "use_3scale_oidc_issuer_endpoint"
   }
 }
…
],
```

- **client_id+client_secret** ：此选项允许您指定不同的 Token Introspection Endpoint，以及*客户端 ID* 和*客户端 Secret* APIcast 用于请求令牌信息。使用这个选项时，请设置以下配置参数：
  - **client_id** ：设置令牌内省端点的客户端 ID。
  - **client_secret** ：设置令牌内省端点的客户端 Secret。
  - **introspection_url** ：设置内省端点 URL。
验证类型设置为 **client_id+client_secret**:

```
"policy_chain": [
…
 {
   "name": "apicast.policy.token_introspection",
   "configuration": {
     "auth_type": "client_id+client_secret",
     "client_id": "myclient",
     "client_secret": "mysecret",
     "introspection_url": "http://red_hat_single_sign-on/token/introspection"
   }
 }
…
],
```

无论 **auth_type** 字段中的设置是什么，APIcast 都会使用 Basic Authentication 来授权 Token Introspection 调用（**Authorization: Basic *<token>*** 头，其中 *<token>* 是 Base64 编码的 *<client_id>:<client_secret>* 设置）。

Edit Policy

Cancel

OAuth 2.0 Token Introspection

builtin - Configures OAuth 2.0 Token Introspection.

This policy executes OAuth 2.0 Token Introspection (https://tools.ietf.org/html/rfc7662) for every API call.

Enabled

max_ttl_tokens

Max TTL for cached tokens

max_cached_tokens

Max number of tokens to cache

auth_type*

client_id+client_secret

introspection_url*

Introspection Endpoint URL

client_id*

Client ID for the Token Introspection Endpoint

client_secret*

Client Secret for the Token Introspection Endpoint

Token Introspection Endpoint 的响应中包含 **active** 属性。APIcast 检查此属性的值。根据属性的值，APIcast 授权或拒绝调用：

- true
  调用已授权
- false
  该调用被拒绝，并显示 **Authentication Failed** 错误

该策略允许缓存令牌，以避免在相同 JWT 令牌的每个调用中调用 Token Introspection Endpoint。要为 Token Introspection Policy 启用令牌缓存，请将 **max_cached_tokens** 字段设置为从 **0** （禁用该功能）到 **10000** 直接的值。另外，您可以在 **max_ttl_tokens** 字段中将令牌的值从 **1** 设置为 **3600** 秒。

4.1.24. On Fail

作为 API 供应商，您可以将 On Fail 策略添加到 API 产品中。当 On Fail 策略位于策略链中，并且针对给定的 API 使用者请求执行策略会失败，APIcast 执行以下操作：

- 停止处理请求。
- 将您指定的状态代码返回到发送请求的应用程序，

当 APIcast 无法处理策略时，On Fail 策略很有用，这可能是因为配置不正确，或因为自定义策略中不合规的代码。如果没有策略链中的 On Fail 策略，APIcast 会跳过它无法应用的策略，在链中处理任何其他策略，并将请求发送到上游 API。在策略链中使用 On Fail 策略，APIcast 拒绝请求。

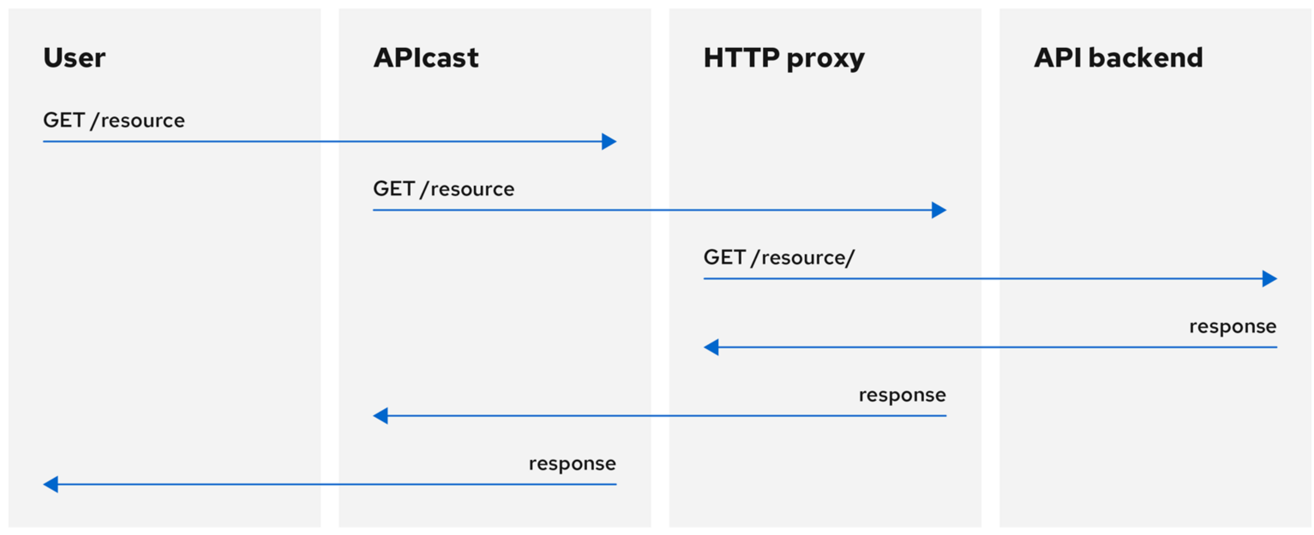
在策略链中，On Fail 策略可以处于任何位置。

在管理门户中，将 On Fail 策略添加到产品的策略链中。在策略链中，点策略指定您希望 APIcast 在应用 On Fail 策略时返回的状态代码。例如，您可以指定 **400**，这表示来自客户端的错误请求。

4.1.25. 代理服务

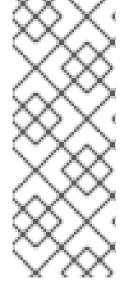
您可以使用 Proxy Service 策略定义一个通用 *HTTP 代理*，该代理将使用定义的代理发送 3scale 流量。在这种情况下，代理服务充当反向 HTTP 代理，APIcast 将流量发送到 HTTP 代理，代理随后将流量发送到 API 后端。

以下示例显示了流量流：



发送到 3scale 后端的所有 APIcast 流量都不使用代理。此策略只适用于代理以及 APIcast 和 API 后端之间的通信。

如果要通过代理发送所有流量，则必须使用 **HTTP_PROXY** 环境变量。



注意

- Proxy Service 策略禁用所有负载平衡策略，流量发送到代理。
- 如果定义了 **HTTP_PROXY**、**HTTPS_PROXY** 或 **ALL_PROXY** 参数，此策略将覆盖这些值。
- 代理连接不支持身份验证。您可以使用标头修改策略进行身份验证。

配置

以下示例显示了策略链配置：

```
"policy_chain": [
   {
    "name": "apicast.policy.apicast"
   },
   {
    "name": "apicast.policy.http_proxy",
    "configuration": {
        "all_proxy": "http://192.168.15.103:8888/",
        "https_proxy": "https://192.168.15.103:8888/",
        "http_proxy": "https://192.168.15.103:8888/"
    }
   }
]
```

如果未定义 **http_proxy** 或 **https_proxy**，则使用 **all_proxy** 值。

使用案例示例

代理服务器策略旨在应用更加精细的策略，并在 3scale 中使用 Apache Camel 通过 HTTP 应用更为精细的策略和转换。但是，您还可以将 Proxy Service 策略用作通用 HTTP 代理服务。有关通过 HTTPS 与 Apache Camel 集成的信息，请参阅 第 4.1.6 节 "Camel Service"。

项目示例

请参阅 GitHub 上的 camel-netty-proxy 示例。此项目显示 HTTP 代理，它将响应正文从 API 后端转换为大写。

4.1.26. 速率限制标头

当应用程序订阅具有速率限制的应用程序计划时，Rate Limit Headers 策略会添加 **RateLimit** 标头来响应消息。这些标头提供有关当前时间窗口中配置的请求配额和剩余的请求配额和秒的有用信息。

在产品的策略链中，如果您添加 Rate Limit Headers 策略，则必须在 3scale APIcast 策略之前。如果 3scale APIcast 策略早于 Rate Limit Headers 策略，则 Rate Limit Headers 策略不起作用。

RateLimit 标头

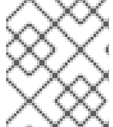
每个消息都添加了以下 **RateLimit** 标头：

- **RateLimit-Limit**
  在配置的时间窗口中显示请求总数，例如 **10** 个请求。

- **ratelimit-Remaining**
  在当前时间窗口中显示剩余的请求配额，例如 **5** 个请求。

- **RateLimit-Reset**
  在当前时间窗口中显示剩余的秒数，例如 **30** 秒。此标头的行为与 **Retry-After** 标头的 **delta-seconds** 表示法兼容。

默认情况下，当没有配置 Rate Limit Headers 策略或应用程序计划没有任何速率限值限制时，响应消息中没有速率限制标头。



注意

如果您请求没有速率限制的 API 指标，但父指标配置了限制，则速率限值标头仍然包含在响应中，因为应用父限制。

其他资源

- Internet-Draft: HTTP 的 RateLimit Header Fields
- 配置 3scale API 管理应用程序计划和速率限值
- 配置 3scale API 管理 API 指标

4.1.27. 响应/请求内容限制

作为 API 提供程序，您可以将 Response/Request Content Limits 策略添加到 API 产品。此策略允许您将请求的大小限制为上游 API，以及来自上游 API 的响应大小。如果没有此策略，请求/响应大小将无限制。

此策略有助于防止过载：

- 后端，因为它必须作用于过大的载荷。
- 最终用户（API 使用者），因为它收到的数据数量超过其可以处理的数据量。

在请求或响应中，3scale 需要 **content-length** 标头来应用 Response/Request Content Limits 策略。

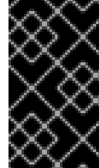
在管理门户中，在将 Response/Request Content Limits 策略添加到产品后，单击它以字节为单位指定限值。您可以指定请求限值或响应限值，或同时指定两者。默认值为 **0**，表示无限大小。

另外，您可以通过更新策略链配置文件来添加此策略，例如：

```
{
  "name": "apicast.policy.limits",
  "configuration":
  {
    "request": 100,
    "response": 100
  }
}
```

4.1.28. Retry

重试策略将请求数设置为上游 API。重试策略是为每个服务配置的，因此用户可以根据需要为任意数量或任意数量的服务启用重试，并为不同的服务配置不同的重试值。



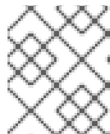
重要

从 3scale 2.14 开始，无法配置从策略重试的情况。这通过环境变量 **APICAST_UPSTREAM_RETRY_CASES** 控制，后者将重试请求应用到所有服务。有关此内容，请查看 APICAST_UPSTREAM_RETRY_CASES。

重试策略 **JSON** 的示例如下所示：

```
{
  "$schema": "http://apicast.io/policy-v1/schema#manifest#",
  "name": "Retry",
  "summary": "Allows retry requests to the upstream",
  "description": "Allows retry requests to the upstream",
  "version": "builtin",
  "configuration": {
    "type": "object",
    "properties": {
      "retries": {
        "description": "Number of retries",
        "type": "integer",
        "minimum": 1,
        "maximum": 10
      }
    }
  }
}
```

4.1.29. RH-SSO/Keycloak 角色检查



注意

当您将 RH-SSO/Keycloak 角色检查策略添加到 APIcast 策略链中时，将其放在 APIcast 和路由策略之前。

此策略在与 OpenID Connect 身份验证选项一起使用时添加角色检查。此策略验证红帽单点登录(RH-SSO)中发布的访问令牌中的域角色和客户端角色。当您要向 3scale 的每个客户端资源添加角色检查时，会指定 realm 角色。

有两种类型的角色检查策略配置中指定的 **type** 属性：

- **whitelist**
  这是默认值。使用 **whitelist** 时，APIcast 将检查 JWT 令牌中是否存在指定的范围，并且如果 JWT 没有范围，将拒绝调用。

- **blacklist**
  使用 **blacklist** 时，如果 JWT 令牌包含黑名单范围，APIcast 将拒绝调用。

同一策略中无法同时配置 **blacklist** 和 **whitelist** 检查，但您可以在 APIcast 策略链中添加多个 **RH-SSO/Keycloak 角色检查** 策略链。

您可以通过策略配置的 **scopes** 属性配置范围列表。

每个 **scope** 对象具有以下属性：

- resource
  由角色控制的资源端点。这个格式与映射规则相同。模式从字符串开头匹配，若要完全匹配，您必须在末尾附加 $。

- resource_type
  这可定义如何评估 **resource** 值。
  - 纯文本(**纯文本**)：评估 **resource** 值作为纯文本。示例： **/api/v1/products$**.
  - Liquid 文本(**liquid**)：允许在 **resource** 值中使用 Liquid。示例： **/resource_{{ jwt.aud }}** 管理对包含客户端 ID 的资源的访问。

- **方法** ：使用此参数根据 RH-SSO 中的用户角色，列出 APIcast 中允许的 HTTP 方法。例如，您可以允许使用以下方法：
  - 用于访问 **/resource1** 的 **role1** realm 角色。对于没有此 realm 角色的方法，您需要指定 **黑名单**。
  - **client1** 角色调用 **role1** 来访问 **/resource1**。
  - 用于访问 **/resource1** 的 **role1** 和 **role2** 域角色。指定 **realm_roles** 中的角色。您还可以指定各个角色的范围。
  - 应用客户端的 **role1** 角色 （这是访问令牌的接收者）来访问 **/resource1**。使用 **liquid** 客户端类型向客户端指定 JSON Web Token(JWT)信息。
  - 客户端角色，包括应用客户端的客户端 ID（访问令牌的接收者）来访问 **/resource1**。使用 **liquid** 客户端类型，将 JWT 信息指定为客户端角色的 **name**。
  - 名为 **role1** 的客户端角色，以访问资源，包括应用客户端 ID。使用 **liquid** 客户端类型为 **resource** 指定 JWT 信息。

- realm_roles
  使用它检查 realm 角色。请参阅红帽单点登录文档中的 Realm 角色。

  域角色存在于红帽单点登录发布的 JWT 中。

  ```
  "realm_access": {
    "roles": [
      "<realm_role_A>", "<realm_role_B>"
    ]
  }
  ```

  策略中必须指定实际角色。

  ```
  "realm_roles": [
    { "name": "<realm_role_A>" }, { "name": "<realm_role_B>" }
  ]
  ```

  以下是 **realm_roles** 数组中每个对象的可用属性：
  - name
    指定角色的名称。
  - name_type
    定义如何评估名称；值可以是 **plain**，也可以是 **liquid**。其工作方式与 **resource_type** 相同。

- **client_roles**
  使用 **client_roles** 检查客户端命名空间中的特定访问角色。请参阅红帽单点登录文档中的客户端角色。

  客户端角色存在于 JWT 中的 **resource_access** 声明下。

  ```
  "resource_access": {
    "<client_A>": {
      "roles": [
        "<client_role_A>", "<client_role_B>"
      ]
    },
    "<client_B>": {
      "roles": [
        "<client_role_A>", "<client_role_B>"
      ]
    }
  }
  ```

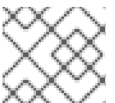
  指定策略中的客户端角色。

  ```
  "client_roles": [
    { "name": "<client_role_A>", "client": "<client_A>" },
    { "name": "<client_role_B>", "client": "<client_A>" },
    { "name": "<client_role_A>", "client": "<client_B>" },
    { "name": "<client_role_B>", "client": "<client_B>" }
  ]
  ```

  以下是 **client_roles** 数组中每个对象的可用属性：

  - **name**
    指定角色的名称。
  - **name_type**
    定义如何评估 **name** 值；该值可以是 **plain** 值，也可以是 **liquid** 值。其工作方式与 **resource_type** 相同。
  - **client**
    指定角色的客户端。如果未定义，此策略将 **aud** 声明用作客户端。
  - **client_type**
    定义如何评估 **client** 值；值可以是 **plain** 值或 **liquid** 值。其工作方式与 **resource_type** 相同。

4.1.30. 路由



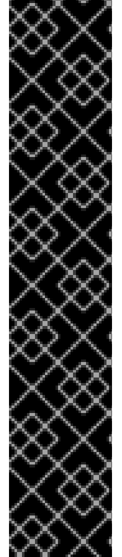
注意

即使路由策略处理请求时，也必须仍存在针对请求的对应映射规则。

Routing 策略允许您将请求路由到不同的目标端点。您可以定义目标端点，然后将从 UI 传入的请求路由到使用正则表达式的请求。

路由基于以下规则：

- 请求路径规则
- 标头规则
- 查询参数规则
- JSON Web 令牌(JWT)声明规则



重要

当您将路由策略添加到策略链时，路由策略必须始终紧接在标准的 3scale APIcast 策略前。换句话说，路由策略和 3scale APIcast 策略之间没有任何策略。这样可确保 APIcast 发送到上游 API 的请求中正确的 APIcast 输出。以下是正确的策略链的两个示例：

```
Liquid Context Debug
JWT Claim Check
Routing
3scale APIcast
```

```
Liquid Context Debug
Routing
3scale APIcast
JWT Claim Check
```

路由规则

- 如果存在多个规则，路由策略会应用第一个匹配项。您可以对这些规则进行排序。
- 如果没有规则匹配，策略不会更改上游，并使用服务配置中定义的已定义的私有基本 URL。

请求路径规则

这是当路径为 **/accounts** 时路由到 **http://example.com** 的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
            }
          ]
        }
      }
```

```
      ]
    }
  }
```

标头规则

这是当标头 **Test-Header** 的值为 **123** 时路由到 **http://example.com** 的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "header",
              "header_name": "Test-Header",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
    ]
  }
}
```

查询参数规则

这是当查询参数 **test_query_arg** 为 **123** 时路由到 **http://example.com** 的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "query_arg",
              "query_arg_name": "test_query_arg",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
    ]
  }
}
```

JWT 声明规则

若要基于 JWT 声明的值进行路由，链中需要有一个策略来验证 JWT 并将其存储在策略共享的上下文中。

这是当 JWT 声明 **test_claim** 的值为 **123** 时路由到 **http://example.com** 的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "operations": [
            {
              "match": "jwt_claim",
              "jwt_claim_name": "test_claim",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
    ]
  }
}
```

多操作规则

只有当所有上游都使用 'and' **combined_op** 评估为 true 时，或者当其中至少一个规则通过使用 'or' **combine_op** 评估为 true 时，规则可以有多个操作并路由到给定上游。**combine_op** 的默认值为 'and'。

这是当请求的路径为 **/accounts** 且标头 **Test-Header** 的值为 **123** 时路由到 **http://example.com** 的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "combine_op": "and",
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
            },
            {
              "match": "header",
              "header_name": "Test-Header",
              "op": "==",
              "value": "123"
            }
```

```
          ]
        }
      }
    ]
  }
}
```

这是当请求的路径为 **/accounts** 或标头 **Test-Header** 的值为 **123** 时路由到 **http://example.com** 的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "condition": {
          "combine_op": "or",
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
            },
            {
              "match": "header",
              "header_name": "Test-Header",
              "op": "==",
              "value": "123"
            }
          ]
        }
      }
    ]
  }
}
```

组合规则

规则可以组合在一起使用。当有多个规则时，上游选择是首批评估为 true 的规则之一。

这是包含多个规则的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://some_upstream.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/accounts"
```

```
          }
        ]
      }
    },
    {
      "url": "http://another_upstream.com",
      "condition": {
        "operations": [
          {
            "match": "path",
            "op": "==",
            "value": "/users"
          }
        ]
      }
    }
  ]
 }
}
```

catch-all 规则

没有操作的规则始终匹配。这对于定义概括性规则非常有用。

如果路径为 **/abc**，则此配置将请求路由到 **http://some_upstream.com**，如果路径是 **/def**，将请求路由到 **http://another_upstream.com**，最后，如果之前的规则没有评估为 true，则会将请求路由到 **http://default_upstream.com** ：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://some_upstream.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/abc"
            }
          ]
        }
      },
      {
        "url": "http://another_upstream.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/def"
            }
          ]
        }
```

```
          },
          {
            "url": "http://default_upstream.com",
            "condition": {
              "operations": []
            }
          }
        ]
      }
    }
```

支持的操作

支持的操作有 **==**, **!=** 和 **matches**。后者将字符串与正则表达式匹配，并使用 ngx.re.match来实施

这是使用 **!=** 的配置。当路径不是 **/accounts** 时，它会路由到 **http://example.com**:

```
    {
      "name": "routing",
      "version": "builtin",
      "configuration": {
        "rules": [
          {
            "url": "http://example.com",
            "condition": {
              "operations": [
                {
                  "match": "path",
                  "op": "!=",
                  "value": "/accounts"
                }
              ]
            }
          }
        ]
      }
    }
```

移动模板

可以将弹性模板用于配置的值。如果链中的策略将键 **my_var** 存储在上下文中，则可以使用动态值定义规则。

这是使用该值路由请求的配置：

```
    {
      "name": "routing",
      "version": "builtin",
      "configuration": {
        "rules": [
          {
            "url": "http://example.com",
            "condition": {
              "operations": [
                {
                  "match": "header",
```

```
          "header_name": "Test-Header",
          "op": "==",
          "value": "{{ my_var }}",
          "value_type": "liquid"
        }
      ]
    }
  }
 ]
 }
}
```

设置 host_header中使用的主机

默认情况下，当路由请求时，策略使用匹配的规则的 URL 主机设置 Host 标头。可以通过 **host_header** 属性指定不同的主机。

这是将 **some_host.com** 指定为 Host 标头主机的配置：

```
{
  "name": "routing",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "url": "http://example.com",
        "host_header": "some_host.com",
        "condition": {
          "operations": [
            {
              "match": "path",
              "op": "==",
              "value": "/"
            }
          ]
        }
      }
    ]
  }
}
```

4.1.31. SOAP

SOAP 策略与 HTTP 请求的 SOAPAction 或 Content-Type 标头中提供的 SOAP 操作 URI 匹配策略中指定的映射规则。

配置属性

属性	描述	值	必需？
pattern	此 **pattern** 属性允许您在 SOAPAction URI 中指定 APIcast 将寻找匹配项的字符串。	数据类型：字符串	是

属性	描述	值	必需？
metric_system_name	**metric_system_name** 属性允许您指定匹配模式将注册命中的 3scale 后端指标。	数据类型：字符串，必须是一个有效的 metric	是

策略对象示例

```
{
  "name": "soap",
  "version": "builtin",
  "configuration": {
    "mapping_rules": [
      {
        "pattern": "http://example.com/soap#request",
        "metric_system_name": "soap",
        "delta": 1
      }
    ]
  }
}
```

有关如何配置策略的详情，请参考 文档中的 3scale API 管理 部分创建策略链。

4.1.32. TLS 客户端证书验证

借助 TLS 客户端证书验证策略，APIcast 实施 TLS 握手，并根据白名单验证客户端证书。白名单包含由认证机构(CA)或纯客户端证书签名的证书。如果证书过期或无效，请求将被拒绝，且不会处理其他策略。

客户端连接到 APIcast 以发送请求并提供客户端证书。APIcast 根据策略配置验证传入请求中提供的证书的真实性。APIcast 也可以配置为使用自己的客户端证书，以在连接到上游时使用它。

设置 APIcast 以使用 TLS 客户端证书验证

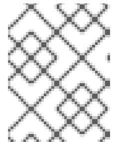
APIcast 需要配置为终止 TLS。按照以下步骤配置用户在 APIcast 上提供的客户端证书验证策略。

您必须有权访问 3scale 安装。您必须等待所有部署完成。

设置 APIcast 以使用策略

要设置 APIcast 并将其配置为终止 TLS，请按照以下步骤操作：

1. 您需要获取访问令牌并部署 APIcast 自我管理，如 使用 OpenShift 模板部署 APIcast 中所述 。



注意

需要 APIcast 自我管理的部署，因为 APIcast 实例需要重新配置，才能将一些证书用于整个网关。

2. 仅用于测试目的，您可以使用没有缓存和暂存环境以及 **--param** 标志的 lazy 加载器来进行测试。

```
$ oc new-app -f https://raw.githubusercontent.com/3scale/3scale-amp-openshift-templates/master/apicast-gateway/apicast.yml --param CONFIGURATION_LOADER=lazy --param DEPLOYMENT_ENVIRONMENT=staging --param CONFIGURATION_CACHE=0
```

3. 生成证书用于测试目的。另外，对于生产部署，您可以使用证书颁发机构提供的证书。

4. 使用 TLS 证书创建 Secret

```
$ oc create secret tls apicast-tls --cert=ca/certs/server.crt --key=ca/keys/server.key
```

5. 在 APIcast 部署中挂载 Secret

```
$ oc set volume dc/apicast --add --name=certificates --mount-path=/var/run/secrets/apicast --secret-name=apicast-tls
```

6. 将 APIcast 配置为为 HTTPS 开始侦听端口 8443

```
$ oc set env dc/apicast APICAST_HTTPS_PORT=8443
APICAST_HTTPS_CERTIFICATE=/var/run/secrets/apicast/tls.crt
APICAST_HTTPS_CERTIFICATE_KEY=/var/run/secrets/apicast/tls.key
```

7. 在服务中公开 8443

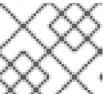
```
$ oc patch service apicast -p '{"spec":{"ports":
[{"name":"httpsproxy","port":8443,"protocol":"TCP"}]}}'
```

8. 删除默认路由

```
$ oc delete route api-apicast-staging
```

9. 将 **apicast** 服务作为路由公开

```
$ oc create route passthrough --service=apicast --port=https --hostname=api-3scale-apicast-staging.$WILDCARD_DOMAIN
```



注意

您要使用的每个 API 和每个 API 的域更改都需要这一步。

10. 通过在占位符中指定 [Your_user_key]，验证之前部署的网关是否正常工作并且配置已保存。

```
curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN?user_key=[Your_user_key] -v --cacert ca/certs/ca.crt
```

在策略链中配置 TLS 客户端证书验证

要在策略链中配置 TLS 客户端证书验证，您需要 3scale 登录凭据。此外，您需要使用 TLS 客户端证书验证策略配置 APIcast。

1. 要将 TLS 客户端证书验证策略添加到您的 API 中，请按照 3scale API 管理门户中启用策略 中所述的步骤，然后选择 TLS 客户端证书验证。

2. 单击 **TLS 客户端证书验证** 链接。

3. 若要启用该策略，选中 **Enabled** 复选框。

4. 若要添加证书到白名单，可单击加号 **+** 图标。

5. 指定包括 **-----BEGIN CERTIFICATE-----** 和 **-----END CERTIFICATE-----** 的证书。

6. 使用 TLS 客户端证书验证设置完 API 后，单击 **Update Policy**。

另外：

- 您可以通过单击加号 **+** 图标来添加更多证书。

- 您还可以通过单击上下箭头来重新组织证书。

要保存您的更改，请点击 **Update Policy Chain**。

验证 TLS 客户端证书验证策略的功能

要验证 TLS 客户端证书验证策略的功能，您需要 3scale 登录凭据。此外，您需要使用 TLS 客户端证书验证策略配置 APIcast。

您可以通过在占位符中指定 **[Your_user_key]** 来验证应用的策略。

```
curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN\?user_key\=[Your_user_key] -v --
cacert ca/certs/ca.crt --cert ca/certs/client.crt --key ca/keys/client.key

curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN\?user_key\=[Your_user_key] -v --
cacert ca/certs/ca.crt --cert ca/certs/server.crt --key ca/keys/server.key

curl https://api-3scale-apicast-staging.$WILDCARD_DOMAIN\?user_key\=[Your_user_key] -v --
cacert ca/certs/ca.crt
```

从白名单中删除证书

要从白名单中删除证书，您需要 3scale 登录凭据。您需要使用 TLS 客户端证书验证策略设置 APIcast。您需要通过在 策略链中配置 TLS 客户端证书验证将证书 添加到白名单。

1. 单击 **TLS 客户端证书验证** 链接。

2. 要从白名单中删除证书，请点击 **x** 图标。

3. 删除证书后，点击 **Update Policy**。

要保存您的更改，请点击 **Update Policy Chain**。

有关使用证书的更多信息，请参阅 红帽认证系统。

4.1.33. TLS 终止

本节提供有关传输层安全(TLS)终止策略的信息：从策略中移除概念、配置、验证和文件删除。

借助 TLS Termination 策略，您可以将 APIcast 配置为为每个 API 完成 TLS 请求，而无需为所有 API 使用单个证书。APIcast 在建立与客户端的连接前拉取配置设置；因此，APIcast 使用策略中的证书，并使 TLS 终止。此策略可与以下源配合工作：

- 存储在策略配置中。
- 存储在文件系统中。

默认情况下，策略链中不启用此策略。

在策略链中配置 TLS 终止

本节介绍了在策略链中配置 TLS 终止的先决条件和步骤，以及增强保密邮件(PEM)格式的证书。先决条件是：

- 用户签发的证书。
- PEM 格式的服务器证书。
- PEM 格式的证书私钥。

按照以下步骤操作：

1. 要将 TLS 终止策略添加到您的 API 中，请按照 启用标准策略 中所述的步骤操作，然后选择 TLS 终止。
2. 单击 **TLS 终止** 链接。
3. 若要启用该策略，选中 **Enabled** 复选框。
4. 若要将 TLS 证书添加到策略，可单击加号 **+** 图标。
5. 选择证书源：
    - 默认选择 **嵌入式证书**。上传这些证书：
        - **PEM 格式的证书私钥** ：点击 **Browse** 来选择和上传。
        - **PEM 格式的证书** ：点击 **Browse** 来选择和上传。
    - **来自文件系统的证书** - 选择并指定这些证书路径：
        - **证书的路径**
        - **证书私钥的路径**
6. 使用 TLS Termination 设置完 API 后，单击 **Update Policy**。

另外：

- 您可以通过单击加号 **+** 图标来添加更多证书。
- 您还可以通过单击上下箭头来重新组织证书。

要保存您的更改，请点击 **Update Policy Chain**。

验证 TLS 终止策略的功能

您必须有 3scale 登录凭证。您必须已使用 TLS Termination 策略配置了 APIcast。

如果策略可使用以下命令，则可以在命令行中测试：

```
curl “${public_URL}:${port}/?user_key=${user_key}" --cacert ${path_to_certificate}/ca.pem -v
```

–

其中：

- **public_URL**
  暂存公共基本 URL。
- **port**
  端口号。
- **user_key**
  要进行身份验证的用户密钥。
- **path_to_certificate**
  到本地文件系统中的 CA 证书的路径。

从 TLS 终止中删除文件

本节论述了从 TLS 终止策略中删除证书和密钥文件的步骤。

- 您需要 3scale 登录凭据。
- 您需要将证书添加到策略中，方法是使用 TLS Termination 策略配置 APIcast。

删除证书：

1. 单击 **TLS 终止** 链接。
2. 要删除证书和密钥，请单击 **x** 图标。
3. 删除证书后，点击 **Update Policy**。

要保存您的更改，请点击 **Update Policy Chain**。

4.1.34. Upstream

通过 Upstream 策略，您可以使用正则表达式来解析主机请求标头，并将私有基本 URL 中定义的上游 URL 替换为不同的 URL。

例如：

具有 regex **/foo** 的策略，URL 字段 **newexample.com** 会将 URL **https://www.example.com/foo/123/** 替换为 **newexample.com**

策略链参考：

属性	描述	值	必需？
regex	**regex** 属性允许您指定在搜索与请求路径匹配项时 Upstream 策略要使用的正则表达式。	数据类型：字符串，必须是一个有效的正则表达式语法	是

属性	描述	值	必需？
url	使用 **url** 属性，您可以在匹配项中指定替换 URL。请注意，Upstream 策略不会检查这个 URL 是否有效。	数据类型：字符串，确定这是有效的 URL	是

策略对象示例

```
{
  "name": "upstream",
  "version": "builtin",
  "configuration": {
    "rules": [
      {
        "regex": "^/v1/.*",
        "url": "https://api-v1.example.com",

      }
    ]
  }
}
```

有关如何配置策略的详情，请参考 文档中的 3scale API 管理 部分创建策略链。

4.1.35. 上游连接

Upstream Connection 策略允许您为每个 API 更改以下指令的默认值，具体取决于您在 3scale 安装中配置 API 后端服务器的方式：

- **proxy_connect_timeout**
- **proxy_send_timeout**
- **proxy_read_timeout**

配置上游连接策略：

- 您必须有权访问 3scale 安装。
- 您需要等待所有部署完成。

按照以下步骤操作：

1. 要将 Upstream Connection 策略添加到您的 API 中，请按照 3scale API 管理管理门户中启用策略中所述的步骤，然后选择 *Upstream Connection*。
2. 单击 **Upstream Connection** 链接。
3. 若要启用该策略，选中 **Enabled** 复选框。

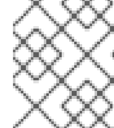
4. 配置与上游连接的选项：
    - **send_timeout**
    - **connect_timeout**
    - **read_timeout**
5. 当您使用 Upstream Connection 设置 API 后，点 **Update Policy**。

要保存您的更改，请点击 **Update Policy Chain**。

4.1.36. Upstream Mutual TLS

使用 Upstream Mutual TLS 策略，您可以根据配置中设置的证书在 APIcast 和上游 API 之间建立并验证 mutual TLS 连接。

启用 **verify** 字段后，策略还会验证来自上游 API 的服务器证书。**ca_certificates** 包含 Privacy Enhanced Mail(PEM)格式的证书，包括 **-----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----** APIcast 使用它来验证服务器。



注意

您必须启用 **verify** 字段，并填写 **ca_certificates** 来验证上游 API 的证书。如果没有启用 **verify** 字段，则只会在上游 API 中检查 APIcast 证书。

要在策略链中配置上游双向 TLS，您需要有权访问 3scale 安装。

1. 要将 Upstream Mutual TLS 策略添加到您的 API 中，请按照 3scale API 管理管理门户中启用策略中所述的步骤，然后选择 *Upstream Mutual TLS*。
2. 单击 **Upstream Mutual TLS** 链接。
3. 若要启用该策略，选中 **Enabled** 复选框。
4. 选择 **证书类型** ：
    - **path**
如果要指定证书的路径，如 OpenShift 生成的证书的路径。
    - **embedded**
如果要使用第三方生成的证书，请通过从您的文件系统上传证书。
5. 在 **Certificate** 中，指定客户端证书。
6. 在 **Certificate key** 中指定密钥。
7. 使用 Upstream Mutual TLS 设置完 API 后，点 **Update Policy Chain**。

推进您的更改：

1. 进入 **[Your_product] page > Integration > Configuration**。
2. 在 *APIcast Configuration* 下，点击 **Promote v# to Staging APIcast**。

V# 代表要提升的配置的版本号。

路径配置

使用 OpenShift 和 Kubernetes secret 的证书路径，如下所示：

```
{
  "name": "apicast.policy.upstream_mtls",
  "configuration": {
      "certificate": "/secrets/client.cer",
      "certificate_type": "path",
      "certificate_key": "/secrets/client.key",
      "certificate_key_type": "path"
  }
}
```

嵌入式配置

对 *http* 表单和文件上传使用以下配置：

```
{
  "name": "apicast.policy.upstream_mtls",
  "configuration": {
    "certificate_type": "embedded",
    "certificate_key_type": "embedded",
    "certificate": "data:application/pkix-cert;name=client.cer;base64,XXXXXXXXXXX",
    "certificate_key": "data:application/x-iwork-keynote-sffkey;name=client.key;base64,XXXXXXXX"
  }
}
```

有关 Upstream Mutual TLS 的其他字段 **ca_certificates** 和 **verify**，策略配置模式 的详细。

其他注意事项

Upstream mutual TLS 策略将覆盖 **APICAST_PROXY_HTTPS_CERTIFICATE_KEY** 和 **APICAST_PROXY_HTTPS_CERTIFICATE** 环境变量值。它使用策略设置的证书，因此这些环境变量不会起作用。

4.1.37. URL Rewriting

URL 重写策略允许您修改请求的路径和查询字符串。

与 3scale APIcast 策略结合使用时，如果在策略链中的 APIcast 策略之前放置 URL 重写策略，APIcast 映射规则将应用到修改的路径。如果在 APIcast 链中的 APIcast 后放置了 URL 重写策略，则映射规则将应用到原始路径。

该策略支持以下两组操作：

- **commands**
  要应用的命令列表来重写请求的路径。

- **query_args_commands**
  要应用的命令列表，用于重写请求的查询字符串。

重写路径的命令

以下是 **commands** 列表中每个命令的配置参数，其中包括：

- **op**

要应用的操作。可用的选项包括：**sub** 和 **gsub**。**sub** 操作仅用您指定的正则表达式替换第一个匹配项。**gsub** 操作会将所有匹配项替换为您指定的正则表达式。请参阅有关 sub 和 gsub 操作的文档。

- regex
  要匹配 Perl 的正则表达式。
- replace
  匹配项时使用的替换字符串。
- options
  这是可选的。定义如何执行 regex 匹配的选项。有关可用选项的详情，请查看 OpenResty Lua 模块项目文档中的 ngx.re.match 部分。
- break
  这是可选的。当启用复选框设置为 true 时，如果命令重新编写该 URL，它将是应用的最后一个，并且列表中的所有 posterior 命令将被丢弃。

重写查询字符串的命令

以下是 **query_args_commands** 列表中每个命令由以下部分组成的配置参数：

- op
  要应用到查询参数的操作。可用的选项如下：
  - add
    为现有参数添加一个值。
  - set
    创建 arg（如果没有设置），并在设置时替换其值。
  - push
    创建 arg（如果没有设置），并在设置时添加值。
  - delete
    删除 arg。
- arg
  操作所应用的查询参数名称。
- value
  指定用于查询参数的值。对于值类型"liquid"，值应当采用 **{{ variable_from_context }}** 格式。对于 **delete** 操作，不考虑该值。
- value_type
  这是可选的。定义如何评估查询参数值，可以是 **plain**（纯文本）或 **liquid**（用于 Liquid 模板）。更多信息请参阅 第 5.1 节 "在策略中使用变量和过滤器"。如果没有指定，则默认使用类型"plain"。

示例

URL 重写策略配置如下：

```
{
  "name": "url_rewriting",
  "version": "builtin",
  "configuration": {
```

```
    "query_args_commands": [
      {
        "op": "add",
        "arg": "addarg",
        "value_type": "plain",
        "value": "addvalue"
      },
      {
        "op": "delete",
        "arg": "user_key",
        "value_type": "plain",
        "value": "any"
      },
      {
        "op": "push",
        "arg": "pusharg",
        "value_type": "plain",
        "value": "pushvalue"
      },
      {
        "op": "set",
        "arg": "setarg",
        "value_type": "plain",
        "value": "setvalue"
      }
    ],
    "commands": [
      {
        "op": "sub",
        "regex": "^/api/v\\d+/",
        "replace": "/internal/",
        "options": "i"
      }
    ]
  }
```

发送到 APIcast 的原始请求 URI：

```
https://api.example.com/api/v1/products/123/details?
user_key=abc123secret&pusharg=first&setarg=original
```

应用 URL 重写后 APIcast 发送到 API 后端的 URI：

```
https://api-backend.example.com/internal/products/123/details?
pusharg=first&pusharg=pushvalue&setarg=setvalue
```

应用以下转换：

1. 子字符串 **/api/v1/** 匹配唯一的路径重写命令，它被 **/internal/** 替换。
2. 已删除 **user_key** 查询参数。
3. 值 **pushvalue** 作为 **pusharg** 查询参数的额外值添加。
4. 查询参数 **setarg** 的 **original** 值替换为配置的值 **setvalue**。

5. 命令 **add** 没有被应用，因为原始 URL 中不存在查询参数 **addarg**。

有关如何配置策略的详情，请参考 文档中的 3scale API 管理 部分创建策略链。

4.1.38. 使用 Captures 重写 URL

URL 重写策略是 URL 重写策略的替代选择，允许在将 API 请求传递给 API 后端前重写 API 请求的 URL。

URL 使用 Captures 策略检索 URL 中的参数，并在重写 URL 中使用其值。

该策略支持 **transformations** 配置参数。这是一个对象列表，用于描述将哪些转换应用到请求 URL。每个调整对象由两个属性组成：

- **match_rule**
  该规则与传入请求 URL 匹配。它可以包含 **{nameOfArgument}** 格式的命名参数；这些参数可以在重写 URL 中使用。将 URL 与 **match_rule** 进行比较，作为正则表达式。匹配指定参数的值必须只包含以下字符（在 PCRE regex 表示法中）：**[\w-.~%!$&'()*,;=@:]**。可以在 **match_rule** 表达式中使用其他 regex 令牌，如 **^** 表示字符串开头，**$** 代表字符串末尾。

- **模板**
  使用重写原始 URL 的 URL 模板；它可以使用 **match_rule** 中的指定参数。

原始 URL 的查询参数与 **template** 中指定的查询参数合并。

示例

使用 Captures 策略的 URL 重写配置如下：

```
{
  "name": "rewrite_url_captures",
  "version": "builtin",
  "configuration": {
    "transformations": [
      {
        "match_rule": "/api/v1/products/{productId}/details",
        "template": "/internal/products/details?id={productId}&extraparam=anyvalue"
      }
    ]
  }
}
```

发送到 APIcast 的原始请求 URI：

```
https://api.example.com/api/v1/products/123/details?user_key=abc123secret
```

应用 URL 重写后 APIcast 发送到 API 后端的 URI：

```
https://api-backend.example.com/internal/products/details?
user_key=abc123secret&extraparam=anyvalue&id=123
```

4.1.39. Websocket

Websocket 策略启用 WebSocket 协议连接到上游 API。如果您计划启用 WebSocket 协议，请考虑以下几点：

- WebSocket 协议不支持 JSON Web 令牌。
- WebSocket 协议不允许额外的标头。
- WebSocket 协议不属于 HTTP/2 标准。

对于启用了 WebSocket 连接的给定上游 API，您可以将其后端定义为 **http[s]** 或 **ws[s]**。

如果您在策略链中添加了 Websocket 策略，请确保 Websocket 策略在 3scale APIcast 策略之前。

4.2. 3SCALE API 管理标准策略中的策略链

对于每个 API 产品，您可以指定策略链。策略链执行以下操作：

- 指定 APIcast 适用于请求的策略。
- 为这些策略提供配置信息。
- 决定 APIcast 应用策略的顺序。

要在链中正确排序策略，了解 APIcast 如何将策略应用到 API 消费者请求。

4.2.1. APIcast NGINX 阶段如何处理 3scale API 管理策略

3scale API 网关或 APIcast 使用 NGINX 代理 Web 服务器应用策略。当 APIcast 收到来自 API 使用者的请求时，APIcast APIcast 会在一系列 NGINX 阶段处理请求。在每个 NGINX 阶段中，APIcast 可以通过应用这些策略来修改原始请求：

- 上游 API 策略链中的策略。策略链是策略的顺序列表。默认情况下，上游 API 的策略链包含 **3scale APIcast** 策略。API 供应商可在 3scale 产品的策略链中添加策略。APIcast 将上游 API 策略链中的策略应用到仅发送到上游 API 的 API 使用者请求。
- 全球 3scale 策略链中的策略。API 供应商您可以设置 3scale 环境变量来更新全局策略链。APIcast 将全局策略链中的策略应用到所有 API 消费者请求。

如果同一策略位于上游 API 策略链和全局策略链中，则上游 API 策略链中的策略配置具有优先权。

在 APIcast 执行所有 NGINX 阶段所需的处理后，APIcast 会向上游 API 发送请求。因此，为了实现所需的行为，务必要理解 NGINX 阶段处理策略的顺序，因为处理可以修改 API 使用者请求。

NGINX 阶段的顺序和描述

当 APIcast 从 API 使用者接收请求时，APIcast 会通过在上游 API 的策略链和全局策略链中应用策略来处理请求。每个 3scale 策略定义一个或多个功能。APIcast 在一系列 NGINX 阶段执行策略功能。在每个阶段中，NGINX 运行所应用策略中定义的任何功能，并在该阶段指定执行。下表列出了运行策略功能的 NGINX 阶段。额外的 NGINX 阶段（在这个表中没有列出）执行不受策略链中策略顺序的处理。

NGINX 阶段（按顺序排列）	本阶段中的处理描述
rewrite	运行修改请求目标 URI 的任何功能。
access	运行验证客户端的授权以发出请求的任何功能。

NGINX 阶段（按顺序排列）	本阶段中的处理描述
content	生成要发送到上游 API 的请求内容。 NGINX 在 **content** 阶段仅应用一个策略。如果策略链中多个策略在请求内容 NGINX 上运行，则仅应用链中最接近的策略。这一点非常重要，因为内置的 3scale APIcast 策略始终在策略链中，它需要在 **内容** 阶段处理 NGINX。 例如，3scale APIcast 策略和 Upstream 策略都会更新请求，以指定上游 API 的路径。NGINX 在 **content** 阶段处理这些功能。如果 3scale APIcast 策略在 Upstream 策略之前，则 NGINX 使用上游 API 配置来将其路径添加到修改后的请求。如果 Upstream 策略在 3scale APIcast 策略之前，NGING 会评估 Upstream 策略表达式。当有匹配项时，NGINX 会在修订请求中相应地更改上游 API 路径。
balancer	运行任何负载平衡功能。
header_filter	运行处理请求标头的任何功能。
body_filter	运行处理请求正文的任何功能。
post_action	在 NGINX 同时运行标题和正文功能后运行任何处理请求的功能。
log	生成有关请求的日志信息。
metrics	对从 Prometheus 端点接收的任何数据执行操作。

执行不受策略顺序影响的 NGINX 阶段示例：

- 当 APIcast 启动时，NGINX 执行与 **init** 阶段关联的任务。
- 当 APIcast worker 启动时，NGINX 执行与 **init_worker** 阶段关联的任务。
- 当 APIcast 终止 HTTPS 连接时，NGINX 执行与 **ssl_certificate** 阶段关联的任务。

NGINX 运行策略功能的顺序

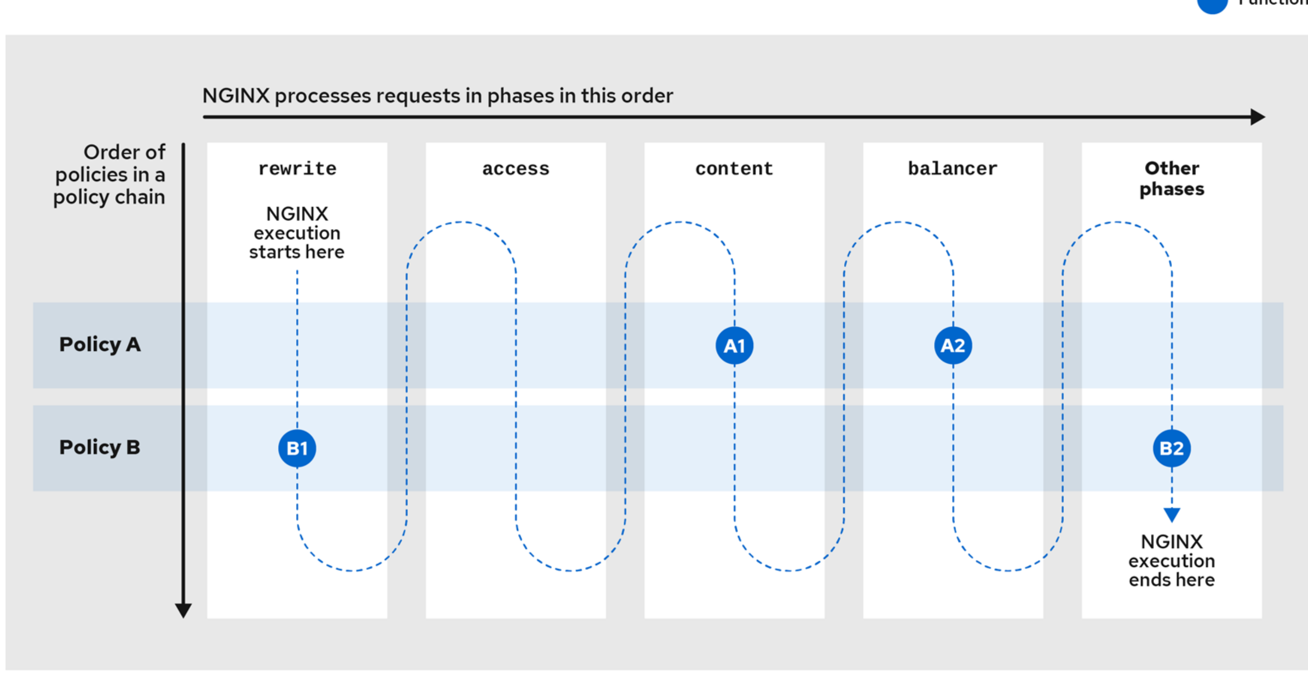
API 供应商可以在 3scale 产品中添加一个或多个策略来形成策略链。在每个阶段中，NGINX 处理那些指定在该阶段执行的策略功能。每个策略功能指定 APIcast 在一个 NGINX 阶段中应如何更改其默认行为。例如，在 **header_filter** 阶段中，GINX 进程指定 **header_filter** 以及可能对请求标头操作的功能。在每个阶段中，NGINX 按策略链中的顺序处理相关功能。

策略可以通过 **context** 对象共享数据。策略可以读取和修改每个阶段的 **context** 对象。

NGINX 执行策略功能的顺序取决于以下内容：

- 策略链中策略的位置
- 处理特定策略功能的 NGINX 阶段

要获得所需行为，您必须正确指定策略链顺序，因为应用策略的结果会根据策略链中的位置而有所不同。下图显示了 NGINX 应用策略的顺序示例。

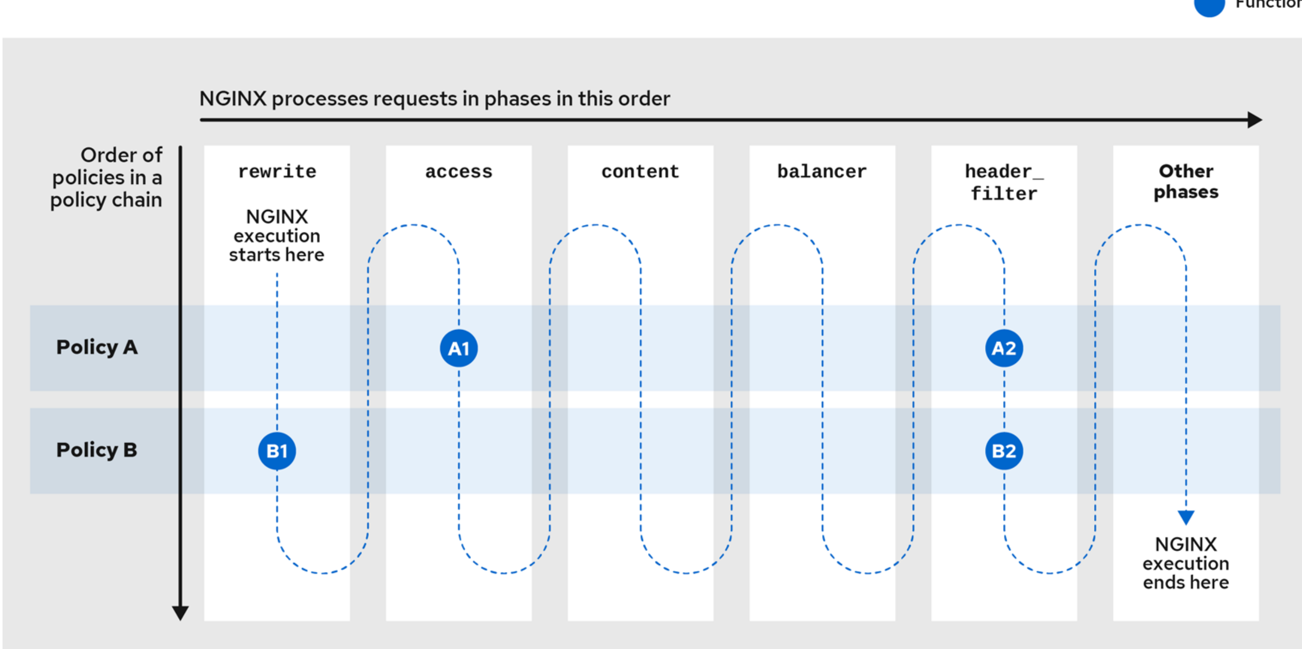


在上图中，策略 A 是策略链中的第一个。但是，NGX 首先在策略 B 中处理某个功能，因为该函数与 NGINX 的第一阶段相关，即 **rewrite** 阶段。

现在考虑一个包含策略 A、带有以下功能的策略 B 的产品策略链：

- 策略y A 指定：
  - 在 **access** 阶段运行 NGINX 的功能 **A1**
  - NGINX 在 **header_filter** 阶段运行的功能 **A2**
- 策略 B 指定：
  - NGINX 在 **rewrite** 阶段运行的功能 **B1**
  - NGINX 在 **header_filter** 阶段运行的功能 **B2**

下图显示了 NGINX 运行该产品的策略功能的顺序。



当 APIcast 收到对此产品公开的上游 API 的访问请求时，APIcast 会检查产品的策略链并运行功能，如下表所述：

NGINX 阶段（按顺序排列）	NGINX 在此阶段运行的功能
rewrite	运行 B 为 **rewrite** 阶段指定的函数 **B1**。
access	运行函数 **A1** 该策略 A 为 **access** 阶段指定。
content	策略 A 和策略 B 指定在 **balancer** 阶段执行的功能。
balancer	策略 A 和策略 B 指定在 **balancer** 阶段执行的功能。
header_filter	策略链指定策略 A，然后指定策略 B。因此，此阶段运行函数 **A2**，为 **header_filter** 阶段指定策略 A，然后运行函数 **B2**，为 **header_filter** 阶段指定策略 B。
body_filter	策略 A 和策略 B 指定在这个阶段执行的功能。
post_action	策略 A 和策略 B 指定在这个阶段执行的功能。
log	策略 A 和策略 B 指定在这个阶段执行的功能。

在本例中，策略 A 最初在策略链中，但策略 B 中的功能是 NGINX 运行的第一个功能。这是因为，策略 B 指定 **rewrite** 阶段中 NGINX 进程的功能 **B1**，这在其它阶段之前。

再举一个例子，请考虑此策略链：

1. URL Rewriting

2. **3scale APIcast**（分配给所有产品的默认策略）

URL Rewriting 策略修改请求的目标路径。APIcast 在 **rewrite** 阶段运行 **URL Rewriting** 功能。**3scale APIcast** 策略定义了 APIcast 在 **rewrite** 阶段运行的功能，以及 APIcast 在三个其他阶段运行的功能。当 **URL Rewriting** 策略是第一个时，**3scale APIcast** 策略将规则映射到重写的路径。如果 **3scale APIcast** 策略是第一个，**URL Rewriting** 策略是第二个，则 **3scale APIcast** 策略将规则映射到原始路径。

其他资源

- 运行 3scale API 管理标准策略功能的 NGINX 阶段
- 3scale API 管理标准策略和处理它们的 NGINX 阶段

4.2.2. 在 3scale API 管理管理门户中修改策略链

修改 3scale 管理门户中的产品策略链，作为 APIcast 网关配置的一部分。

流程

1. 登录 3scale。
2. 导航到您要为其配置策略链的 API 产品。
3. 在 **[your_product_name] > Integration > Policies**中，点 *Add policy*。
4. 在 **Policy Chain** 部分下，使用箭头图标来重新排序策略链中的策略。
5. 单击 **Update Policy Chain** 以保存策略链。

后续步骤

在管理门户的左侧导航面板中，有一个警告，表示您有没有提升到 APIcast 的 **Configuration** 改变。将策略链更新提升到 Staging APIcast，并根据需要测试更新。确认所需行为后，将更新提升到 Production APIcast。如果 **APICAST_CONFIGURATION_CACHE** 环境变量设置为一个大于零（默认值）的数字（默认值），则 APIcast 使用更新的配置需要这个秒数。

4.2.3. 在 JSON 配置文件中创建 3scale API 管理策略链

如果使用 APIcast 的原生部署，您可以创建一个 JSON 配置文件来控制外部的策略链。

JSON 配置文件策略链包含一个由以下信息组成的 JSON 数组：

- 带有 **id** 值的 **services** 对象，用于指定策略链按编号应用到哪个服务。
- **proxy** 对象，其中包含 **policy_chain** 对象和后续对象。
- **policy_chain** 对象，其中包含定义策略链的值。
- 单独的 **policy** 对象，用于指定策略和配置策略所需的 **name** 和 **configuration** 数据

以下是自定义策略 **sample_policy_1** 和 API 内省标准策略 **token_introspection** 的策略链示例：

```
{
  "services":[
    {
      "id":1,
```

```
      "proxy":{
        "policy_chain":[
          {
            "name":"sample_policy_1", "version": "1.0",
            "configuration":{
              "sample_config_param_1":["value_1"],
              "sample_config_param_2":["value_2"]
            }
          },
          {
            "name": "token_introspection", "version": "builtin",
            "configuration": {
              introspection_url:["https://tokenauthorityexample.com"],
              client_id:["exampleName"],
              client_secret:["secretexamplekey123"]
          },
          {
             "name": "apicast", "version": "builtin",
          }
        ]
      }
    }
  ]
}
```

所有策略链必须包含内置策略 **apicast**。在策略链中放置 **apicast** 策略会影响策略行为。

4.2.4. 运行 3scale API 管理标准策略功能的 NGINX 阶段

下表列出了标准策略的主要 NGINX 阶段，用于定义 NGINX 在该阶段中运行的功能。表按照 NGINX 进程的顺序列出阶段。

策略链可包含特定阶段中 NGINX 进程的多个策略。在这种情况下，请确保链中策略的顺序是处理 API 请求以获得所需结果的正确顺序。表以字母顺序列出策略。

NGINX 阶段（按顺序排列）	定义在这个阶段处理的功能的标准策略
rewrite	3scale APIcast 3scale Referrer Anonymous Access Echo Header Modification NGINX Filter SOAP Upstream URL Rewriting URL Rewriting with Captures Websocket

NGINX 阶段（按顺序排列）	定义在这个阶段处理的功能的标准策略
access	3scale APIcast 3scale Batcher Camel Proxy Content Cache Edge Limiting IP Check JWT Claim Check RH-SSO/Keycloak Role Check Maintenance Mode OAuth 2.0 Mutual TLS Client Authentication OAuth 2.0 Token 内省 Rate Limit Headers Response/Request Content Limits Routing TLS Client Certificate Validation Upstream
content	3scale APIcast Liquid Context Debug Rate Limit Headers Routing Upstream
balancer	Upstream Mutual TLS
header_filter	CORS 请求处理 标头修改 响应/请求内容限制 HTTP Response Code Overwrite
body_filter	响应/请求内容限制
post_action	3scale APIcast 自定义指标
log	边缘限制 日志

其他资源

- APIcast NGINX 阶段如何处理 3scale API 管理策略
- 3scale API 管理标准策略和处理它们的 NGINX 阶段

4.2.5. 3scale API 管理标准策略和处理它们的 NGINX 阶段

下表列出了标准策略以及运行该策略功能或功能的 NGINX 阶段或阶段。使用此表在策略链中正确排序策略，为上游 API 生成正确的请求。

标准策略	运行策略功能的 NGINX 阶段
3scale APIcast	**Init** **重写** 访问 **内容** **post_action** APIcast 将 **3scale APIcast** 策略应用到所有请求。
Anonymous Access（匿名访问）	**rewrite**
3scale Auth 缓存	在策略链中，此策略的位置无关紧要。
3scale Batcher	**access**
3scale Referrer	**rewrite**
Camel Service	**access**
条件策略	在策略链中，此策略的位置无关紧要。
内容缓存	**access**
CORS 请求处理	**header_filter**
自定义指标	**post_action**
Echo	**rewrite**
边缘限制	**access** **log**
标头修改	**rewrite** **header_filter**
HTTP 响应代码覆盖	**header_filter**
IP 检查	**access**
JWT 申索检查	**access**
Liquid Context Debug	**content**
日志记录	**log**
维护模式	**access**
NGINX Filter	**rewrite**

标准策略	运行策略功能的 NGINX 阶段
OAuth 2.0 通用 TLS 客户端身份验证	access
OAuth 2.0 令牌内省	access
代理服务	在策略链中，此策略的位置无关紧要。
速率限制标头	access +content
响应/请求内容限制	access header_filter body_filter
Retry	在策略链中，此策略的位置无关紧要。
RH-SSO/Keycloak 角色检查	access
路由	访问 内容
SOAP	rewrite
TLS 客户端证书验证	access
TLS 终止	ssl_certificate
Upstream	重写 访问 内容
上游连接	在策略链中，此策略的位置无关紧要。
Upstream Mutual TLS	balancer
URL Rewriting	rewrite
使用 Captures 重写 URL	rewrite
Websocket	rewrite

其他资源

- APIcast NGINX 阶段如何处理 3scale API 管理策略
- 运行 3scale API 管理标准策略功能的 NGINX 阶段

4.3. 使用 API 修改代理策略链

要在策略链中管理策略，您可以使用帐户管理 API，而不使用 3scale 管理门户。使用帐户管理 API （称为 API），您可以更改控制 API 流量的代理策略链。您可以添加、删除、重新排序或修改策略，将整个功能视为一个端点，称为 *Proxy Policies Chain Update* 。使用 *Proxy Policies Chain Update* 端点调用 API：

```
PUT /admin/api/services/{service_id}/proxy/policies.json
```

对端点的调用必须在请求正文中包含 **access_token** 和 **policies_config** 参数。**policies_config** request body 参数应该是 URL 编码的 JSON 数组。数组中的每个元素代表一个策略配置。

Proxy Policies Chain Update 端点返回更新的代理策略链。无效的输入会导致错误。

要查看策略链，请对帐户管理 API 使用以下 **GET** 调用：

```
GET /admin/api/services/{service_id}/proxy/policies.json
```

GET 调用示例策略链输出

```
{
  "policies_config": [
    {
      "name": "cors",
      "version": "builtin",
      "configuration": {
        "allow_headers": [],
        "allow_methods": [
          "GET"
        ],
        "allow_origin": "https://example.com",
        "allow_credentials": true
      },
      "enabled": true
    },
    {
      "name": "apicast",
      "version": "builtin",
      "configuration": {},
      "enabled": true
    }
  ]
}
```

在前面的 JSON 响应中，policy_config 属性的有效负载是一个数组，代表调用 *Proxy Policies Chain Update* 端点中的 **policies_config** 参数的预期值。

4.3.1. 使用 curl 命令更新策略链

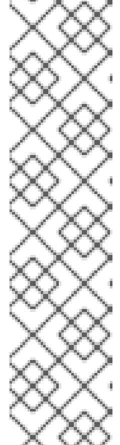
以下示例演示了如何使用 **curl** 命令和 **jq** 工具读取和更新代理策略链。将占位符值 **{admin_portal_url}**、**{service_id}** 和 **{access_token}** 替换为代表您的环境的值。

4.3.1.1. 在 curl 请求中提供 policies_config 内联

流程

1. 获取当前的策略链：

```
$ curl -s "{admin_portal_url}/admin/api/services/{service_id}/proxy/policies.json?access_token={access_token}" | jq '.policies_config' -c
```



注意

- **-s** 选项启用"silent"模式，以抑制不属于请求响应的输出。
- **jq '.policies_config'** 从响应中的 **policies_config** JSON 属性中提取策略链数组。
- **jq** 工具的 **-c** 选项以紧凑模式输出，以避免多行。

该命令返回一个在策略链中显示 CORS 和 APIcast 策略的响应，例如：

```
[{"name":"cors","version":"builtin","configuration":{"allow_headers":[],"allow_methods":["GET","POST","PUT"],"allow_origin":"https://example.com","allow_credentials":true},"enabled":true},{"name":"apicast","version":"builtin","configuration":{},"enabled":true}]
```

2. 通过在链中添加、删除或重新排序策略来编辑策略链，或者更改其配置。

3. 更新策略链。

   在以下 curl 命令示例中，CORS 策略已从链中删除，但您仍可以对策略链进行其他更改。

```
$ curl -X PUT "{admin_portal_url}/admin/api/services/{service_id}/proxy/policies.json" -d 'access_token={access_token}' -d 'policies_config=[{"name":"apicast","version":"builtin","configuration":{},"enabled":true}]'
```

4.3.1.2. 从文件提供 policies_config 内容

流程

1. 将当前策略链保存到文件中：

```
curl -s "{admin_portal_url}/admin/api/services/{service_id}/proxy/policies.json?
access_token={access_token}" | jq '.policies_config' > policies_config.json
```

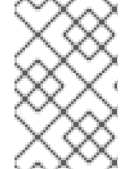
2. 通过在链中添加、删除或重新排序策略来编辑 policies_config.json 文件中的策略链，或者更改其配置。

3. 更新策略链：

```
$ curl -X PUT "{admin_portal_url}/admin/api/services/{service_id}/proxy/policies.json"
-d 'access_token={access_token}' --data-urlencode
policies_config@policies_config.json
```

4.4. 自定义 3SCALE API 管理 APICAST 策略

配置自定义策略以修改 APIcast 行为。首先，定义一个策略链来配置 APIcast 策略，包括您的自定义策略；然后，将策略链添加到 APIcast。



注意

红帽 3scale 提供了一种添加自定义策略的方法，但不支持自定义策略。

APIcast 的自定义策略取决于您的 3scale 部署的配置：

- 将自定义策略添加到这些 APIcast 自我管理的部署：OpenShift 上的 APIcast 和您安装的容器化环境中的 APIcast。

- 您无法将自定义策略添加到 APIcast 托管。

警告

切勿直接在生产网关上进行策略更改。始终测试您的更改。

4.4.1. 关于 3scale API 管理 APIcast 部署的自定义策略

您可以完全创建自定义 APIcast 策略，或修改标准策略。

要创建自定义策略，您必须了解以下内容：

- 策略用 Lua 编写。
- 策略必须遵循，并且必须放在正确的文件目录中。
- 策略行为受到策略链中的放置方式的影响。
- 完全支持添加自定义策略的接口，但不支持自定义策略本身。

4.4.2. 在 3scale API 管理嵌入式 APIcast 中添加自定义策略

要将自定义 APIcast 策略添加到内部部署中，您必须构建包含自定义策略的 OpenShift 镜像，并将其添加到您的部署中。3scale 提供了一个示例存储库，您可以使用作为框架来创建和添加自定义策略到内部部署。

此示例存储库包含自定义策略的正确目录结构，以及用于创建镜像流和 BuildConfig 的模板，用于构建包含您创建的任何自定义策略的新 APIcast OpenShift 镜像。

警告

构建 apicast-custom-policies 时，构建过程会将新镜像推送到 amp-apicast:latest 标签。当此镜像流上有镜像更改时，*apicast-staging* 和 *apicast-production* 标签都配置为自动启动新部署。为了避免对暂存或生产服务出现任何中断，请取消选择 *"Automatically start a new deployment"* 复选框来禁用自动部署。或者，为生产环境配置不同的镜像流标签，如 amp-apicast:production。

流程

1. 使用您在 Creating a registry service account 中创建的凭证创建一个 docker-registry secret，具体如下：

   - 将 your-registry-service-account-username 替换为以 12345678|username 创建的用户名。
   - 将 your-registry-service-account-password 替换为用户名下面的密码字符串，位于 *Token Information* 选项卡下。
   - 为每个镜像流所在并使用 *registry.redhat.io* 的新命名空间 创建一个 docker-registry secret。

     运行这个命令来创建 docker-registry secret：

```
$ oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username="your-registry-service-account-username" \
  --docker-password="your-registry-service-account-password"
```

2. 使用 策略示例分叉公共存储库，或使用其内容创建私有存储库。您需要在 Git 存储库中提供自定义策略的代码，供 OpenShift 构建该镜像。请注意，为了使用私有 Git 存储库，您必须在 OpenShift 中设置机密。

3. 在本地克隆存储库，为您的策略添加实施，并将更改推送到您的 Git 存储库。

4. 更新 openshift.yml 模板。特别是，更改以下参数：

   a. 策略 BuildConfig 中的 spec.source.git.uri: https://github.com/3scale/apicast-example-policy.git - 将它更改为 Git 存储库位置。

   b. 自定义策略 BuildConfig 中的 spec.source.images[0].paths.sourcePath: /opt/app-root/policies/example - 把 example 改为您在存储库中的 policies 目录下添加的自定义策略名称。

   c. （可选）更新 OpenShift 对象名称和镜像标签。但是，您必须确保更改是一致的。例如：apicast-example-policy BuildConfig 构建并推送 apicast-policy:example 镜像，该

镜像随后被 apicast-custom-policies BuildConfig 用作源。因此，标签应当相同。

5. 运行以下命令来创建 OpenShift 对象：

```
$ oc new-app -f openshift.yml --param AMP_RELEASE=2.14
```

6. 如果构建没有自动启动，请运行以下两个命令：如果您更改了它，请将 apicast-example-policy 替换为您自己的 BuildConfig 名称，如 apicast-<name>-policy。等待第一个命令完成，然后执行第二个命令。

```
$ oc start-build apicast-example-policy
$ oc start-build apicast-custom-policies
```

如果内置 APIcast 镜像跟踪 amp-apicast:latest 镜像流中的更改，则 APIcast 的新部署将启动。apicast-staging 重启后，导航到 Integration > Policies，然后点击 Add Policy 按钮来查看列出的自定义策略。选择并配置后，单击 Update Policy Chain，使自定义策略在暂存 APIcast 中正常工作。

4.4.3. 在另一个 OpenShift Container Platform 中添加自定义策略到 3scale API 管理

您可以通过从集成的 OpenShift Container Platform 注册表获取包含自定义策略的镜像，将自定义策略添加到 OpenShift Container Platform (OCP)上的 APIcast。

流程

1. 将策略添加到 APIcast 内置。

2. 如果您没有在 OpenShift 主集群中部署 APIcast 网关，请 建立对主 OpenShift 集群上的内部注册表的访问。

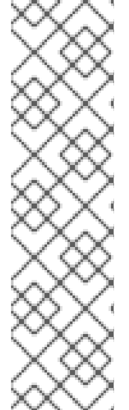
3. 下载 3scale 2.14 APIcast OpenShift 模板。

4. 要修改模板，请将默认 镜像 目录替换为内部注册表中的完整镜像名称。

```
image: <registry>/<project>/amp-apicast:latest
```

5. 使用 OpenShift 模板部署 APIcast，并指定您的自定义镜像：

```
$ oc new-app -f customizedApicast.yml
```



注意

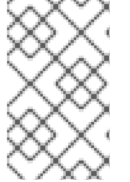
当自定义策略添加到 APIcast 并构建新镜像时，当 APIcast 使用镜像部署时，这些策略将自动显示在管理门户中。现有服务可以在可用策略列表中看到此新策略，因此可以在任何策略链中使用。

从镜像中删除自定义策略并重启 APIcast 时，该策略将不再在列表中可用，因此您无法再将它添加到策略链中。

4.4.4. 在 3scale API 管理自定义策略中包含外部 Lua 依赖项

您可以将外部 Lua 依赖项添加到自定义策略中，以便 APIcast 可以使用 3scale 镜像中没有的 Lua 库。

此处的步骤演示了如何使用自定义策略示例来将响应正文从 JSON 转换为 XML。自定义策略示例需要 xml2lua XML 解析器，该解析器使用 Lua 编写。完整的示例显示了用于构建和测试的简短示例，但您无法仅遵循示例过程来部署自定义策略。要部署具有外部 Lua 依赖项的自定义策略，您必须执行此流程中的步骤，并在 另一个 OpenShift Container Platform 中将自定义策略添加到 3scale API 管理。



注意

JSON 到 XML 自定义策略只是示例。它不可用于生产环境。

先决条件

- 3scale 自定义策略。
- 访问外部 Lua 库。

流程

1. 在包含自定义策略的目录中，添加一个可识别外部 Lua 库的文件。

文件的名称必须是 Roverfile。在 JSON 到 XML 自定义 策略示例中，Roverfile 具有此内容：

```
luarocks {
 group 'production' {
  module { 'xml2lua' },
 }
}
```

lua-rover 是 LuaRocks 的打包程序。lua-rover 为依赖项提供传输锁定。LuaRocks 是 Lua 模块的软件包管理器。

2. 在包含自定义策略的目录中，添加一个 lua-rover 锁定文件。

   文件的名称必须是 Roverfile.lock。在 JSON to XML 自定义策略示例中，Roverfile.lock 具有此内容：

```
xml2lua 1.5-2||productionbash-4.4
```

   Roverfile 和 Roverfile.lock 一起启用 APIcast 或 3scale 操作器来获取依赖的库。

3. 在定义自定义策略的文件中，添加一行来指定 Lua 依赖项。JSON to XML 自定义策略示例指定这一行：

```
local xml2lua = require("xml2lua")
```

4. 在用于构建自定义策略的 Dockerfile 中，复制 Roverfile 和 Roverfile.lock，并运行 rover install。JSON to XML 自定义策略示例将以下行添加到其 Dockerfile 中：

```
COPY Roverfile .
COPY Roverfile.lock .

RUN rover install --roverfile=/opt/app-root/src/Roverfile
```

   您的 Dockerfile 可以使用 APIcast 或 3scale operator 来构建策略。

5. 在自定义策略的 Makefile 中，指定任何自定义策略的 build 目标。

例如，构建 目标可能类似如下：

```
TARGET_IMAGE="apicast/json_to_xml:latest"
# IP="http://localhost:8080"

build:
 docker build . --build-arg IMAGE=registry.redhat.io/3scale-amp2/apicast-gateway-
rhel8:3scale2.14 -t $(TARGET_IMAGE)
```

后续步骤

部署具有外部 Lua 依赖项的自定义策略的其余步骤与部署其他自定义策略的步骤相同。也就是说，您需要将镜像推送到存储库，并将 APIcast 镜像替换为您刚才构建的镜像。

其他资源

- 在另一个 OpenShift Container Platform 中添加自定义策略到 3scale API 管理
- APIManager CRD Reference, APIcastSpec image 参数
- APIcast 自定义资源引用镜像参数

第 5 章 将策略链与 APICAST 原生部署集成

对于原生 APIcast 部署，您可以通过使用 THREESCALE_CONFIG_FILE 环境变量指定配置文件来集成 自定义策略链。以下示例指定了配置文件 example.json ：

```
THREESCALE_CONFIG_FILE=example.json bin/apicast
```

5.1. 在策略中使用变量和过滤器

一些 标准策略 支持 Liquid 模板化，它不仅允许使用纯字符串值，也允许在请求上下文中使用变量。

要使用上下文变量，将其名称打包在 {{ 和 }} 中，例如： {{ uri }}。如果变量是对象，您也可以访问其属性，例如： {{ somevar.attr }}。

以下是所有策略中的标准变量：

- **uri**

  此路径中排除的查询参数请求的路径。嵌入式 NGINX 变量的值 $uri。

- **主机**

  请求的主机，这是嵌入式 NGINX 变量 $host 的值。

- **remote_addr**

  客户端的 IP 地址，这是嵌入式 NGINX 变量 $remote_addr 的值。

- 标头

  包含请求标头的对象。使用 {{headers['Some-Header']}} 获取特定的标头值。

- **http_method**

  请求方法：GET、POST 等。

这些标准变量在请求的上下文中使用，但策略可以在上下文中添加更多变量。阶段是指 APIcast 具有的所有执行步骤。在以下情况下，策略链中的所有策略都可以使用变量：

- 在同一阶段中，如果在策略中添加了变量，然后在添加后在以下策略中使用。
- 如果在阶段中添加变量，则可在后续阶段使用此变量。

以下是标准 3scale APIcast 策略添加到上下文中的变量示例：

- JWT ：用于 OpenID Connect 身份验证的 JWT 令牌解析 JSON 有效负载。
- 凭证 ：保存应用程序凭证的对象。例如："app_id": "972f7b4f", "user_key": "13b668c4d1e10eaebaa5144b4749713f"。
- service：一个对象，用于存储处理当前请求的服务的配置。示例：服务 ID 将为 {{ service.id }}。

有关上下文中可用对象和值的完整列表，请参阅 Liquid 上下文调试。

变量配合 Liquid 模板使用。示例： {{ remote_addr }}、{{ headers['Some-Header'] }}、{{ jwt.aud }}.支持这些值的变量的策略具有一个特殊参数，通常使用 _type 后缀，如 value_type、name_type，接受两个值：纯文本"plain"，以及"liquid"用于 liquid 模板。

APIcast 还支持 Liquid 过滤器，可应用于变量的值。过滤器将 NGINX 功能应用到 Liquid 变量的值。

过滤器在变量输出标签 {{ }} 中放置，紧随变量的名称或通过竖线字符 | 和过滤器的名称来排列的字面值。示例：

- {{ 'username:password' | encode_base64 }}，其中 username:password 是变量。
- {{ uri | escape_uri }}。

有些过滤器不需要参数，因此您可以使用空字符串而不是 变量。示例： {{ '' | utctime }} 将以 UTC 时区返回当前时间。

过滤器可以按如下所示链接：{{ variable | function1 | function2 }}。示例： {{ '' | utctime |scap_uri }}.

以下是可用功能列表：

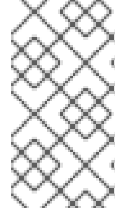
- escape_uri
- unescape_uri
- encode_base64
- decode_base64
- crc32_short
- crc32_long
- hmac_sha1
- md5
- md5_bin

- sha1_bin
- quote_sql_str
- today
- time
- now
- localtime
- utctime
- cookie_time
- http_time
- parse_http_time

第 6 章 使用 FUSE 中的策略扩展转换 3SCALE API 管理消息内容

您可以使用红帽 Fuse 为红帽 3scale API 管理创建高度灵活的策略扩展。您可以通过在 OpenShift 上的 Fuse 中创建策略扩展，然后将它们配置为 3scale 管理门户中的策略。使用 APIcast Camel 代理策略，您可以对请求和响应消息内容执行复杂的转换，例如 XML 到 JSON，它们在 Apache Camel 集成框架中实施。

此外，您可以在 Camel 中动态添加或修改自定义策略扩展，而不是重新构建和重新部署静态 APIcast 容器镜像。您可以使用任何使用 Camel 域特定语言(DSL)编写的 Camel 企业集成模式(EIP)来实施 APIcast 策略扩展。这可让您使用熟悉的编程语言（如 Java 或 XML）编写策略扩展。本主题的示例使用 Camel Netty4 HTTP 组件在 Java 中实施 HTTP 代理。



注意

如果您已在 3scale API 后端中使用 Fuse Camel 应用程序，则不需要此功能。在这种情况下，您可以使用现有的 Fuse Camel 应用程序来执行转换。

所需的软件组件

您必须在同一 OpenShift 集群中部署以下 Red Hat 集成组件：

- OpenShift 7.10 上的 Fuse。
- 3scale 内部部署 2.14.
- APIcast 嵌入式（默认暂存和生产）或 APIcast 自我管理。

您可以在 3scale 之外的其他 OpenShift 项目中部署自定义 Fuse 策略，但这不是必需的。但是，您必须确保两个项目间的通信成为可能。详情请参阅使用 OpenShift SDN 配置网络策略。

其他资源

- OpenShift 上的 Fuse 指南

6.1. 在 FUSE 中集成 APICAST 与 APACHE CAMEL 转换

您可以将 APIcast 与在 OpenShift 上的 Fuse 中编写为 Apache Camel 应用程序转换进行集成。当在 3scale 中配置和部署策略扩展后，3scale 流量将经由 Camel 策略扩展，该扩展将转换消息内容。在这种情况下，Camel 充当反向 HTTP 代理，APIcast 将 3scale 流量发送到 Camel，然后 Camel 会将流量发送到 API 后端。

本主题的示例使用 Camel Netty4 HTTP 组件创建 HTTP 代理：

- 通过 HTTP 代理协议接收的请求将转发到目标服务，并将 HTTP 正文转换为大写。
- 目标服务的响应是通过将其转换为大写，然后返回到客户端来处理的。
- 本例显示了 HTTP 和 HTTPS 用例所需的配置。

先决条件

- 您必须在同一 OpenShift 集群上部署 Fuse on OpenShift 7.10 和 3scale 2.14。有关安装详情，请参阅：
  - OpenShift 上的 Fuse 指南.
  - 安装 3scale API 管理。
- 您必须具有集群管理员特权才能在 OpenShift 和 3scale 上安装 Fuse 并创建项目。但是，您可以创建部署配置、部署容器集，或者创建具有每个项目编辑访问权限的服务。

流程

1. 使用 Camel netty4-http 组件以 Java 编写 Apache Camel 应用程序，以实施 HTTP 代理。然后，您可以使用任何 Camel 组件转换消息。

   以下简单示例从服务执行请求和响应的大写转换：

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Locale;
```

```
import org.apache.camel.Exchange;
import org.apache.camel.Message;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.RouteDefinition;

public class ProxyRoute extends RouteBuilder {

   @Override
   public void configure() throws Exception {
      final RouteDefinition from;
      if (Files.exists(keystorePath())) {
         from = from("netty4-http:proxy://0.0.0.0:8443?
ssl=true&keyStoreFile=/tls/keystore.jks&passphrase=changeit&trustStoreFile=/tls/keys
tore.jks"); (1)
      } else {
         from = from("netty4-http:proxy://0.0.0.0:8080");
      }

      from
         .process(ProxyRoute::uppercase)
         .toD("netty4-http:"
            + "${headers." + Exchange.HTTP_SCHEME + "}://" (2)
            + "${headers." + Exchange.HTTP_HOST + "}:"
            + "${headers." + Exchange.HTTP_PORT + "}"
            + "${headers." + Exchange.HTTP_PATH + "}")
         .process(ProxyRoute::uppercase);
   }

   Path keystorePath() {
      return Path.of("/tls", "keystore.jks");
   }

   public static void uppercase(final Exchange exchange) { (3)
      final Message message = exchange.getIn();
      final String body = message.getBody(String.class);
      message.setBody(body.toUpperCase(Locale.US));
   }

}
```

1

在这个简单示例中，如果您的 Java 密钥存储文件挂载于 /tls/keystore.jks，侦听端口将设置为 8443。

2

3scale 调用 Camel 代理策略时，会根据 3scale 中为后端 API 配置的值自动设置 HTTP_SCHEME、HTTP_HOST、HTTP_PORT 和 HTTP_PATH 标头的值。

3

这个简单示例将消息内容转换为大写。您可以使用 Camel Enterprise Integration Patterns 对请求和响应消息内容（例如 XML 到 JSON）执行更复杂的转换。

2. 在 OpenShift 上部署您的 Camel 应用程序并将其公开为服务。如需了解更多详细信息，请参阅在 OpenShift 的 Fuse 上创建和部署应用程序。

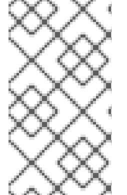
其他资源

- Apache Camel 组件参考 - Netty4 HTTP 组件

6.2. 配置使用 OPENSHIFT 上 FUSE 中的 APACHE CAMEL 创建的 APICAST 策略扩展

在 OpenShift 上使用 Fuse 实施 Apache Camel 转换后，您可以使用 3scale 管理门户将其配置为 APIcast 策略链中的策略扩展。

策略扩展允许您配置 3scale 产品以使用 Camel HTTP 代理。此服务用于通过 HTTP 代理发送 3scale 流量，以在第三方代理中执行请求响应修改。在这种情况下，第三方代理是使用 OpenShift 中的 Fuse 实施 Apache Camel。您还可以配置 APIcast，以使用 TLS 安全地连接到 Camel HTTP 代理服务。



注意

策略扩展代码在 OpenShift 上的 Fuse 中的 Apache Camel 应用程序中实施，无法从 3scale 修改或删除。

先决条件

- 您必须在同一 OpenShift 集群上部署 Fuse on OpenShift 7.10 和 3scale 2.14。有关安装详情，请参阅：
  - OpenShift 上的 Fuse 指南
  - 安装 3scale API 管理
- 您必须已在 OpenShift 的 Fuse 中使用 Apache Camel 应用程序实施了 APIcast 策略扩展。请查看 第 6.1 节 “在 Fuse 中集成 APIcast 与 Apache Camel 转换”

- 您必须已在 OpenShift pod 中部署 Apache Camel 应用程序，并将其作为服务公开。如需了解更多详细信息，请参阅在 OpenShift 的 Fuse 上创建和部署应用程序。

流程

1. 在 3scale Admin Portal 中，选择 **Integration** > **Policies**。

2. 选择 **POLICIES** > **Add policy** > **Camel Service**。

3. 在相应字段中输入用于连接到 Camel HTTP 代理服务的 OpenShift 路由：

   - **HTTPS_ PROXY** ：使用 **http** 协议和 TLS 端口连接到 Camel HTTP 代理，例如：

     ```
     http://camel-proxy.my-3scale-management-project.svc:8443
     ```

   - **HTTP _ PROXY** ：使用 **http** 协议和端口连接到 Camel HTTP 代理，例如：

     ```
     http://camel-proxy.my-3scale-management-project.svc:8080
     ```

   - **ALL_ PROXY** ：当未指定协议时，使用 **http** 协议和端口连接到 Camel HTTP 代理，例如：

     ```
     http://camel-proxy.my-3scale-management-project.svc:8080
     ```

4. 将更新的策略配置提升到您的暂存或生产环境。例如，点 **Promote v.3 to Staging APIcast**.

5. 使用 3scale curl 命令测试 APIcast 策略配置，例如：

   ```
   curl "https://testapi-3scale-apicast-staging.myuser.app.dev.3sca.net:443/?user_key=MY_USER_KEY" -k
   ```

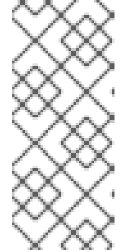
   APIcast 建立一个新的 TLS 会话，用于连接 Camel HTTP 代理。

6. 确认消息内容已转换，本例中将转换为大写。

7. 如果要绕过 APIcast 并直接使用 TLS 测试 Camel HTTP 代理，则必须使用自定义 HTTP 客户端。例如，您可以使用 netcat 命令：

```
$ print "GET https://mybackend.example.com HTTP/1.1\nHost:
mybackend.example.com\nAccept: */*\n\n" | ncat --no-shutdown --ssl my-camel-proxy
8443
```

本例使用 GET 后的完整 URL 创建 HTTP 代理请求，并使用 ncat --ssl 参数在端口 8443 上指定 与 my-camel-proxy 主机的 TLS 连接。



注意

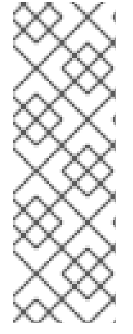
您不能使用 curl 或其他通用 HTTP 客户端直接测试 Camel HTTP 代理，因为代理不支持使用 CONNECT 方法进行 HTTP 隧道。通过 CONNECT 使用 HTTP 隧道时，传输是端到端加密，不允许 Camel HTTP 代理调节载荷。

其他资源

- 第 4.1.6 节 “Camel Service”

第 7 章 APICAST 环境变量

APIcast 环境变量允许您修改 APIcast 的行为。以下值是受支持的环境变量：



注意

- 不支持或已弃用的环境变量不会被列出
- 有些环境变量功能可能已移到 APIcast 策略

- all_proxy, ALL_PROXY
- APICAST_ACCESS_LOG_BUFFER
- APICAST_ACCESS_LOG_FILE
- APICAST_BACKEND_CACHE_HANDLER
- APICAST_CACHE_MAX_TIME
- APICAST_CACHE_STATUS_CODES
- APICAST_CONFIGURATION_CACHE
- APICAST_CONFIGURATION_LOADER
- APICAST_CUSTOM_CONFIG
- APICAST_ENVIRONMENT

- APICAST_EXTENDED_METRICS
- APICAST_HTTPS_CERTIFICATE
- APICAST_HTTPS_CERTIFICATE_KEY
- APICAST_HTTPS_PORT
- APICAST_HTTPS_PROXY_PROTOCOL
- APICAST_HTTPS_VERIFY_DEPTH
- APICAST_HTTP_PROXY_PROTOCOL
- APICAST_LARGE_CLIENT_HEADER_BUFFERS
- APICAST_LOAD_SERVICES_WHEN_NEEDED
- APICAST_LOG_FILE
- APICAST_LOG_LEVEL
- APICAST_MANAGEMENT_API
- APICAST_MODULE
- APICAST_OIDC_LOG_LEVEL

- APICAST_PATH_ROUTING
- APICAST_PATH_ROUTING_ONLY
- APICAST_POLICY_LOAD_PATH
- APICAST_PROXY_HTTPS_CERTIFICATE
- APICAST_PROXY_HTTPS_CERTIFICATE_KEY
- APICAST_PROXY_HTTPS_PASSWORD_FILE
- APICAST_PROXY_HTTPS_SESSION_REUSE
- APICAST_REPORTING_THREADS
- APICAST_RESPONSE_CODES
- APICAST_SERVICE_CACHE_SIZE
- APICAST_SERVICE_${ID}_CONFIGURATION_VERSION
- APICAST_SERVICES_LIST
- APICAST_SERVICES_FILTER_BY_URL
- APICAST_UPSTREAM_RETRY_CASES

- APICAST_WORKERS
- BACKEND_ENDPOINT_OVERRIDE
- HTTP_KEEPALIVE_TIMEOUT
- http_proxy HTTP_PROXY
- https_proxy HTTPS_PROXY
- no_proxy NO_PROXY
- OPENSSL_VERIFY
- OPENTRACING_CONFIG
- OPENTRACING_HEADER_FORWARD
- OPENTRACING_TRACER
- RESOLVER
- THREESCALE_CONFIG_FILE
- THREESCALE_DEPLOYMENT_ENV
- THREESCALE_PORTAL_ENDPOINT

ALL_PROXY

Default: 没有值

值 ：字符串

示例:http://forward-proxy:80

定义要在未指定特定于协议的代理时用于连接服务的 HTTP 代理。不支持身份验证。

APICAST_ACCESS_LOG_BUFFER

Default: 没有值

value: 正整数

允许以字节的块的形式包含访问日志写入。结果为更少的系统调用，这可以提高网关的性能。

APICAST_ACCESS_LOG_FILE

Default: stdout

定义将存储访问日志的文件。

APICAST_BACKEND_CACHE_HANDLER

Default: strict

Values: strict | resilient

弃用 ：使用 缓存策略。

定义当后端不可用时授权缓存的行为方式。当后端不可用时，请严格删除缓存的应用。弹性仅在从后端获取授权时才会这样做。

APICAST_CACHE_MAX_TIME

默认： 1m

Value: 字符串

在系统中选择要缓存的响应时，此变量的值表示要缓存的最长时间。如果未设置 cache-control 标头，则缓存的时间将是定义的标头。

此值的格式由 proxy_cache_valid NGINX 指令定义。

此参数仅供使用内容缓存策略的 API 使用，并且请求符合缓存条件。

APICAST_CACHE_STATUS_CODES

默认： 200, 302

Value: 字符串

当上游的响应代码与此环境变量中定义的一个状态代码匹配时，响应内容将缓存在 NGINX 中。缓存时间取决于其中一个值：标头缓存时间值或 APICAST_CACHE_MAX_TIME 环境变量定义的最长时间。

此参数仅供使用内容缓存策略的 API 使用，并且请求符合缓存条件。

APICAST_CONFIGURATION_CACHE

Default ：0

值 ：整数

指定配置要存储的时间间隔（以秒为单位）。该值应设置为 0（不兼容 APICAST_CONFIGURATION_LOADER的引导值）或超过 60 个。例如，如果 APICAST_CONFIGURATION_CACHE 设为 120，则网关将每 2 分钟从 API 管理器重新加载配置（120 秒）。值 < 0 禁用重新加载。

APICAST_CONFIGURATION_LOADER

值 ：boot | lazy

默认 ： lazy

定义如何加载配置。启动将在网关启动时向 API 管理器请求配置。lazy 将根据每个传入请求按需加载（保证对每个请求 APICAST_CONFIGURATION_CACHE 应该为 0 完全刷新）。

APICAST_CUSTOM_CONFIG

弃用 ：使用 策略。

定义实施自定义逻辑的 Lua 模块的名称，覆盖现有的 APIcast 逻辑。

APICAST_ENVIRONMENT

Value: string[:]

示例 ：production:cloud-hosted

APIcast 应加载以冒号(:)分隔的环境（或路径）列表。此列表可以替代 CLI 上的 -e 或 --environment 参数，例如，存储在容器镜像中作为默认环境。CLI 上传递的任何值都会覆盖此变量。

APICAST_EXTENDED_METRICS

默认 ：false

值 ：布尔值

示例 ："true"

启用 Prometheus 指标的附加信息。以下指标有 service_id 和 service_system_name 标签，它们提供了有关 APIcast 的更多深入详情：

- **total_response_time_seconds**
- **upstream_response_time_seconds**
- **upstream_status**

APICAST_HTTPS_CERTIFICATE

Default: 没有值

HTTPS 的 PEM 格式带有 X.509 证书的文件路径。

APICAST_HTTPS_CERTIFICATE_KEY

Default: 没有值

使用 PEM 格式的 X.509 证书 secret 密钥的文件路径。

APICAST_HTTPS_PORT

Default: 没有值

控制哪些端口 APIcast 应开始侦听 HTTPS 连接。如果此冲突与 HTTP 端口冲突，它将仅用于 HTTPS。

APICAST_HTTPS_PROXY_PROTOCOL

Default: false **Values:** boolean **Example:** true

此参数为 HTTPS 侦听器启用代理协议。

APICAST_HTTPS_VERIFY_DEPTH

默认: 1

值 ：正整数

定义客户端证书链的最大长度。如果此参数的值有 1，则可以在客户端证书链中包含额外证书。例如，root 证书认证机构。

APICAST_HTTP_PROXY_PROTOCOL

默认： false

值 ： 布尔值

示例： true

此参数为 HTTP 侦听器启用代理协议。

APICAST_LARGE_CLIENT_HEADER_BUFFERS

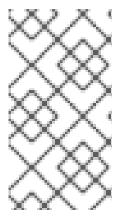
默认： 4 8k

Value: 字符串

设置用于读取大型客户端请求标头的最大缓冲数和大小。

这个值的格式由 large_client_header_buffers NGINX 指令定义。

APICAST_LOAD_SERVICES_WHEN_NEEDED



注意

在 3scale 2.14 中，删除了 APICAST_LOAD_SERVICES_WHEN_NEEDED 环境变量。现在，它被默认处于活跃状态。请参阅 THREESCALE_PORTAL_ENDPOINT。

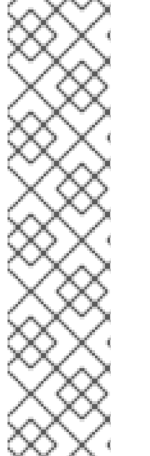
默认 ：*false*

Values:

- true 或 1 用于 *true*
- false、0 或空用于 *false*

当配置了多个服务时，可以使用此选项。但是，其性能取决于其他因素，例如服务数量、APIcast 和 3scale 管理门户之间的延迟，以及配置的生存时间(TTL)。

默认情况下，APIcast 每次从管理门户下载其配置时加载所有服务。启用这个选项后，配置将使用延迟加载。APIcast 将仅加载为请求的主机标头中指定的主机配置的主机。



注意

- 应用的 APICAST_CONFIGURATION_CACHE 定义的缓存。
- 当 APICAST_CONFIGURATION_LOADER 是 boot 时将禁用此选项。
- 与 APICAST_PATH_ROUTING 不兼容。

APICAST_LOG_FILE

默认 ： stderr

定义包含 OpenResty 错误日志的 文件。文件供 error_log 指令中的 bin/apicast 使用。文件路径可以是绝对的，也可以是相对于 APIcast 前缀目录。请注意，默认前缀目录为 APIcast。如需更多信息，请参阅 NGINX 文档。

APICAST_LOG_LEVEL

默认 ：warn

值 ：debug | info | notice | warn | error | crit | alert | emerg

指定 OpenResty 日志的日志级别。

APICAST_MANAGEMENT_API

Values:

- disabled ：完全禁用，只侦听端口

- status ：仅为健康检查启用 /status/ 端点

- debug ：会打开完整的 API

管理 API 功能强大，可以控制 APIcast 配置。您应该只为调试启用 debug 级别。

APICAST_MODULE

默认 ：apicast

弃用 ：使用 策略。

指定实施 API 网关逻辑的主 Lua 模块的名称。自定义模块可以覆盖默认 apicast.lua 模块的功能。请参阅有关如何使用模块的示例。

APICAST_OIDC_LOG_LEVEL

值 ：debug | info | notice | warn | error | crit | alert | emerg

默认 ：err

允许为与 OpenID Connect 集成相关的日志设置日志级别。

APICAST_PATH_ROUTING

值 ：

- true 或 1 用于 true
- false、0 或空用于 false

当此参数设置为 *true* 时，除了基于主机的默认路由外，网关还将使用基于路径的路由。API 请求将从请求的 Host 标头值与*公共基础 URL* 匹配的服务列表中路由到具有匹配映射规则的第一个服务。

APICAST_PATH_ROUTING_ONLY

值 ：

- true 或 1 用于 true
- false、0 或空用于 false

当此参数设置为 *true* 时，网关将使用基于路径的路由，并且不会回退到基于主机的默认路由。API 请求将从请求的 Host 标头值与*公共基础 URL* 匹配的服务列表中路由到具有匹配映射规则的第一个服务。

此参数的优先级高于 APICAST_PATH_ROUTING。如果启用了 APICAST_PATH_ROUTING_ONLY，APIcast 将只执行基于路径的路由，无论 APICAST_PATH_ROUTING 的值为何。

APICAST_POLICY_LOAD_PATH

默认:APICAST_DIR/policies

Value: string[:]

示例 ：~/apicast/policies:$PWD/policies

冒号(:)分隔的路径列表，APIcast 应该在其中查找策略。它可用于首先从开发目录加载策略或加载示例。

APICAST_PROXY_HTTPS_CERTIFICATE

Default:

值 ：字符串

示例 ：/home/apicast/my_certificate.crt

APIcast 与上游连接时将使用的客户端 SSL 证书的路径。请注意，此证书将用于配置中的所有服务。

APICAST_PROXY_HTTPS_CERTIFICATE_KEY

Default:

值 ：字符串

示例 ：/home/apicast/my_certificate.key

客户端 SSL 证书密钥的路径。

APICAST_PROXY_HTTPS_PASSWORD_FILE

Default:

值 ：字符串

示例 ：/home/apicast/passwords.txt

使用 APICAST_PROXY_HTTPS_CERTIFICATE_KEY 指定 SSL 证书密钥的密语文件的路径。

APICAST_PROXY_HTTPS_SESSION_REUSE

默认 ： on

值：

- on：重用 SSL 会话。
- off：不重复使用 SSL 会话。

APICAST_REPORTING_THREADS

Default：0

值：整数 >= 0

实验性：在极端负载下，可能会有无法预计的性能并丢失报告。

大于 0 的值将启用对后端的带外报告。这是一个全新的 实验 功能，用于提高性能。客户端不会看到后端延迟，一切都将异步处理。这个值决定了在客户端因增加延迟而节流前可以同时运行多少异步报告。

APICAST_RESPONSE_CODES

默认：<empty>(*false*)

值：

- true 或 1 用于 true
- false、0 或空用于 false

当设置为 true 时，APIcast 将记录 3scale 中 API 后端返回的响应代码。如需更多信息，请参阅为 API 设置和评估 3scale API 管理响应代码日志。

APICAST_SERVICE_CACHE_SIZE

默认: 1000

值 ：整数 >= 0

指定 APIcast 可存储在内部缓存中的服务数量。高值会影响性能，因为 Lua 的 lru 缓存将初始化所有条目。

APICAST_SERVICE_${ID}_CONFIGURATION_VERSION

将 ${ID} 替换为实际的服务 ID。该值应当是您可以在管理门户的配置历史记录中看到的配置版本。将它设置为特定版本将阻止它自动更新，并且始终使用该版本。

APICAST_SERVICES_LIST

value ：以逗号分隔的服务 ID 列表

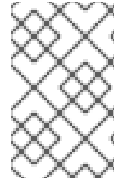
APICAST_SERVICES_LIST 环境变量用于过滤您在 3scale API Manager 中配置的服务。这仅应用网关中特定服务的配置，从而丢弃未在列表中指定的服务标识符。您可以在 Products > [Your_product_name] > Overview 下的 Admin Portal 中找到您的产品的服务标识符，然后参阅 Configuration、Methods 和 Settings 以及 *API 调用的 ID*。

APICAST_SERVICES_FILTER_BY_URL

值 ：PCRE（Perl 兼容正则表达式）， 如 *.example.com。

过滤 3scale API Manager 中配置的服务。

此过滤器与*公共基本 URL* 匹配，可以是暂存或生产。与过滤器不匹配的服务将被丢弃。如果无法编译正则表达式，则不会加载任何服务。



注意

如果服务不匹配但包含在 “APICAST_SERVICES_LIST”一节 中，则不会丢弃该服务。

例 7.1. 应用到后端端点的 Regexp 过滤器

Regexp 过滤器 http://.*.foo.dev 应用到以下后端端点：

1. http://staging.foo.dev

2. http://staging.bar.dev

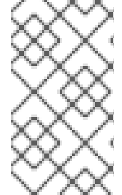
3. http://prod.foo.dev

4. http://prod.bar.dev

在这种情况下，1 和 3 在 APIcast 中配置，2 和 4 将被丢弃。

APICAST_UPSTREAM_RETRY_CASES

值: error | timeout | invalid_header | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | http_429 | non_idempotent | off



注意

这只在配置了重试策略并且指定何时应重试对上游 API 的请求时使用。它接受与 Nginx 的 PROXY_NEXT_UPSTREAM 模块 相同的值。

APICAST_WORKERS

默认 : auto

值: integer | auto

这是 nginx worker_processes 指令 中使用的值。默认情况下，APIcast 使用 auto，但使用 1 的开发环境除外。

BACKEND_ENDPOINT_OVERRIDE

从配置覆盖后端端点的 URI。在 OpenShift 外部部署 AMP 时非常有用.示例：https://backend.example.com.

HTTP_KEEPALIVE_TIMEOUT

默认: 75

值 ：正整数

示例 ：*1*

此参数设置一个超时，期间 keep-alive 客户端连接将在服务器端保持打开状态。零值禁用 keep-alive 客户端连接。

默认情况下，网关会禁用 HTTP_KEEPALIVE_TIMEOUT。此配置允许使用默认值为 75 秒 的 NGINX 中保留超时。

HTTP_PROXY,HTTP_PROXY

Default: 没有值

值 ：字符串

示例:http://forward-proxy:80

定义用于连接 HTTP 服务的 HTTP 代理。不支持身份验证。

HTTPS_PROXY,HTTPS_PROXY

Default: 没有值

值 ：字符串

示例:https://forward-proxy:443

定义用于连接 HTTPS 服务的 HTTP 代理。不支持身份验证。

NO_PROXY,NO_PROXY

Default: 没有值

值: string\[,<string>\]; *

示例 ：foo,bar.com,.extra.dot.com

定义不应代理请求的主机名和域名的逗号分隔列表。设置为单个 * 字符（匹配所有主机）有效禁用代理。

OPENSSL_VERIFY

值 ：

- 0,false: 禁用对等验证

- 1, true ：启用对等验证

控制 OpenSSL 对等验证.它默认为 off，因为 OpenSSL 无法使用系统证书存储。它需要自定义证书捆绑包并将其添加到可信证书中。

建议您使用 https://github.com/openresty/lua-nginx-module#lua_ssl_trusted_certificate 并指向由 export-builtin-trusted-certs 生成的证书捆绑包。

OPENTRACING_CONFIG

此环境变量用于决定 opentracing tracer 的配置文件，如果未设置 OPENTRACING_TRACER，则忽略此变量。

每个 tracer 都有一个默认配置文件：* jaeger: conf.d/opentracing/jaeger.example.json

通过使用此变量设置文件路径，您可以选择挂载与默认情况下提供的不同的配置。

示例 ：/tmp/jaeger/jaeger.json

OPENTRACING_HEADER_FORWARD

默认 ：uber-trace-id

此环境变量控制用于转发 Opentracing 信息的 HTTP 标头，此 HTTP 标头将转发到上游服务器。

OPENTRACING_TRACER

示例 ：jaeger

此环境变量控制将加载的追踪库，现在只有一个打开追踪器可用，即 jaeger。

如果为空，将禁用 opentracing 支持。

RESOLVER

允许指定将由 OpenResty 使用的自定义 DNS 解析器。如果 RESOLVER 参数为空，则会自动发现 DNS 解析器。

THREESCALE_CONFIG_FILE

带有网关配置的 JSON 文件的路径。您必须提供 THREESCALE_PORTAL_ENDPOINT 或 THREESCALE_CONFIG_FILE 才能成功运行网关。从这两个环境变量中，THREESCALE_CONFIG_FILE 优先

要使用网关配置来构建文件，根据服务数量，有两个替代方案：

- 使用可用的 3scale API 端点：
  - *Proxy Config Show*、*Proxy Config Show Latest*，或 *Proxy Configs 列表*。您必须知道服务的 ID。使用以下选项：
    - 使用 *Proxy Configs List* 供应商端点：< schema>://<admin-portal>/admin/api/account/proxy_configs/<env>.json

    - 端点会返回供应商的所有存储代理配置，而不只返回每个服务的最新内容。
    - 迭代 JSON 中返回的 proxy_configs 数组。
    - 选择 proxy_config.content，其 proxy_config.version 是具有相同 proxy_config.content.id 的所有 proxy_configs 最高。ID 是服务之一。
  - 然后，迭代服务以构建配置文件。在这一步中，使用可用的 3scale API 端点或等同的 3scale toolbox 命令。

当使用容器镜像部署网关时：

1. 将 文件配置为只读卷。
2. 指定指示挂载卷的位置的路径。

您可以在 Example 文件夹中找到 示例 配置文件。

THREESCALE_DEPLOYMENT_ENV

默认 ：production

值 ： stage | production

此环境变量的值定义从中下载配置的环境；在使用新 APIcast 时是 3scale staging 或 production。

这个值也会在对 3scale Service Management API 的 authorize/report 请求的 header X-3scale-User-Agent 中使用。3scale 仅将它用于统计数据目的。

THREESCALE_PORTAL_ENDPOINT

以以下格式包含密码和门户端点的 URI：

<schema>://<password>@<admin-portal>.

其中：

- <password> 可以是 3scale 帐户管理 API 的供应商密钥或访问令牌。

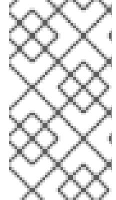
- <admin-portal > 是登录到 3scale 管理门户的 URL 地址。

示例 ：https://access-token@account-admin.3scale.net.

当提供了 THREESCALE_PORTAL_ENDPOINT 环境变量并且 APICAST_CONFIGURATION_LOADER=boot 时，网关会在初始化时从 3scale 下载配置。配置包括在 [Your_product_name] > Integration 下 API 集成页面中提供的所有设置。

您还可以使用此环境变量 创建具有主管理门户的单一网关。

要成功运行网关，您必须提供 THREESCALE_PORTAL_ENDPOINT 或 THREESCALE_CONFIG_FILE。请注意，THREESCALE_CONFIG_FILE 优先于 THREESCALE_PORTAL_ENDPOINT。



注意

在 3scale 2.14 中，删除了 APICAST_LOAD_SERVICES_WHEN_NEEDED 环境变量。现在，它被默认处于活跃状态。

默认情况下，获取配置。以下是逻辑规格：

- THREESCALE_PORTAL_ENDPOINT 不在 URL 中包含路径。
  - APICAST_CONFIGURATION_LOADER=boot

> http://${THREESCALE_PORTAL_ENDPOINT}/admin/api/account/proxy_configs/<env>.json?version=latest

/admin/api/account/proxy_configs/${env} 端点被分页。

- APICAST_CONFIGURATION_LOADER=lazy

> http://${THREESCALE_PORTAL_ENDPOINT}/admin/api/account/proxy_configs/<env>.json?host=<hostname_of_the_request>&version=latest

/admin/api/account/proxy_configs/${env} 端点被分页。

- THREESCALE_PORTAL_ENDPOINT 在 URL 中包含一个带有 master 端点的路径，即 /master/api/proxy/configs

  - APICAST_CONFIGURATION_LOADER=boot

> http://${THREESCALE_PORTAL_ENDPOINT}/<env>.json

这适用于来自 master 端点的所有服务，这与当前的行为相同。

  - APICAST_CONFIGURATION_LOADER=lazy

> http://${THREESCALE_PORTAL_ENDPOINT}/<env>.json?host=<hostname_of_the_request>

这适用于与 master 端点的主机匹配的所有服务，这与当前的行为相同。

表 7.1. 附加到 ${THREESCALE_PORTAL_ENDPOINT} 的路径：

	APICAST_CONFIGURATION_LOADER=boot	APICAST_CONFIGURATION_LOADER=lazy
端点没有路径	/admin/api/account/proxy_configs/${env}.json?version=version	/admin/api/account/proxy_configs/${env}.json?host=host&version=version
端点有一个路径	/${env}.json	/${env}.json?host=host

第 8 章 配置 APICAST 以获得更好的性能

本文档提供了在 APIcast 中调试性能问题的一般准则。它还介绍了可用的缓存模式，并解释了它们如何帮助提高性能，以及分析模式的详细信息。内容由以下部分构成：

- 第 8.1 节 “常规指南”
- 第 8.2 节 “默认缓存”
- 第 8.3 节 “异步报告线程”
- 第 8.4 节 “3scale API 管理 Batcher 策略”

8.1. 常规指南

在典型的 APIcast 部署中，需要考虑三个组件：

- APIcast。
- 3scale 后端服务器，用于授权请求并跟踪使用情况。
- 上游 API。

在 APIcast 中遇到性能问题时：

- 识别负责这些问题的组件。
- 测量上游 API 的延迟，以确定 APIcast 加上 3scale 后端服务器的延迟。
- 利用您用于运行基准测试的相同工具，执行新的测量，但指向 APIcast，而不是直接指向上游 API。

比较这些结果可让您了解 APIcast 和 3scale 后端服务器带来的延迟。

在带有自我管理的 APIcast 的托管(SaaS)安装中，如果 APIcast 和 3scale 后端服务器引入的延迟较高：

1. 从部署了 APIcast 的同一机器向 3scale 后端服务器发出请求。
2. 测量延迟.

3scale 后端服务器公开返回版本： https://su1.3scale.net/status 的端点。相比之下，授权调用需要更多资源，因为它验证键、限值和队列后台作业。虽然 3scale 后端服务器在几毫秒内执行这些任务，但它的工作要比检查 /status 端点所执行的版本更多。例如，如果对 /status 的请求从 APIcast 环境中大约需要 300 毫秒，则每个未缓存的请求将花费更多时间。

8.2. 默认缓存

对于没有缓存的请求，这些是事件：

1. APIcast 从匹配的映射规则中提取用量指标。
2. APIcast 将指标加上应用凭据发送到 3scale 后端服务器。
3. 3scale 后端服务器执行以下操作：
    a. 检查应用密钥，并且报告的指标使用量是否在定义的限制之内。
    b. 对后台作业排队，以增加所报告指标的使用量。
    c. 响应 APIcast 请求是否应授权。

4. 如果请求被授权，请求将转至上游.

在这种情况下，请求不会到达上游，直到 3scale 后端服务器响应为止。

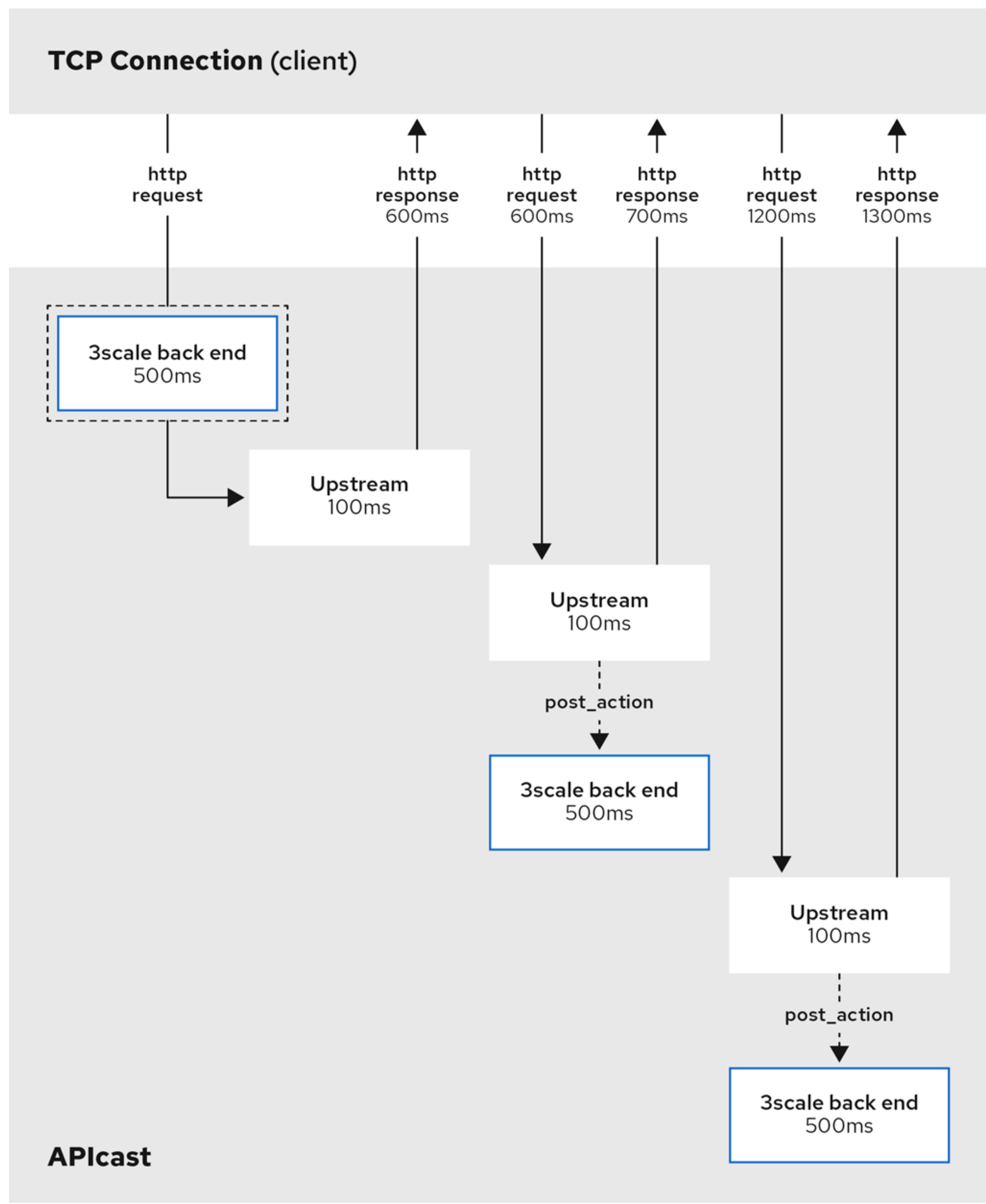
另一方面，使用默认启用的缓存机制：

- APIcast 将存储在缓存中，如果已授权，则向 3scale 后端服务器授权调用的结果。
- 下一个具有相同凭据和指标的请求将使用该缓存的授权，而不是前往 3scale 后端服务器。
- 如果请求未获得授权，或者 APIcast 首次收到凭据，APIcast 将同步调用 3scale 后端服务器，如上方所述。

缓存身份验证后，APIcast 首先调用上游，然后在名为 *post 操作*的 阶段中，它将调用 3scale 后端服务器并在缓存中存储授权，使它准备好下一请求。请注意，对 3scale 后端服务器的调用不会引入任何延迟，因为它不会在请求时间发生。但是，在同一连接中发送的请求将需要等待后续 *操作* 阶段结束。

想象一下，客户端使用 *keep-alive* 并每秒发送请求的情况。如果上游响应时间为 100 ms，3scale 后端服务器的延迟为 500 毫秒，客户端每次都会获得 100 毫秒的响应。上游响应和报告总共需要 600 毫秒。这在下一请求到来前额外提供了 400 毫秒。

下图说明了默认的缓存行为。缓存机制的行为可以使用 缓存策略 来更改。



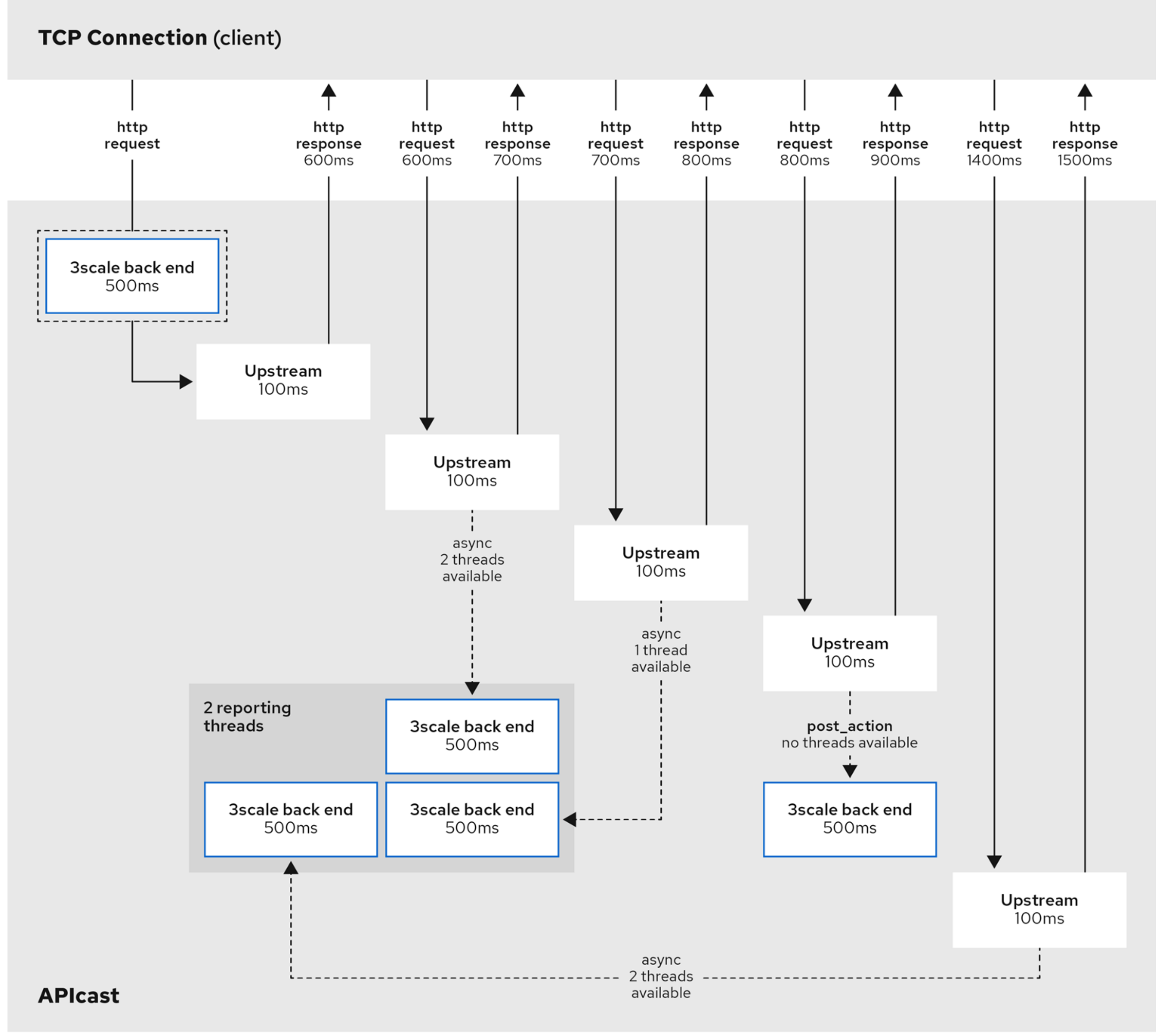
8.3. 异步报告线程

APIcast 具备启用对 3scale 后端服务器授权的线程池的功能。启用此功能后，APIcast 首先调用 3scale 后端服务器，以验证应用和指标是否与映射规则匹配。这与使用默认启用的缓存机制时类似。不同之处在于，只要池中存在可用报告线程，对 3scale 后端服务器的后续调用就完全异步报告。

报告线程对整个网关而言是全局的，在所有服务之间共享。进行第二个 TCP 连接时，只要已缓存授权，它也会完全异步。如果没有免费的报告线程，同步模式将返回标准异步模式，并在后的操作阶段报告。

您可以使用 APICAST_REPORTING_THREADS 环境变量启用此功能。

下图说明了异步报告线程池的工作方式。

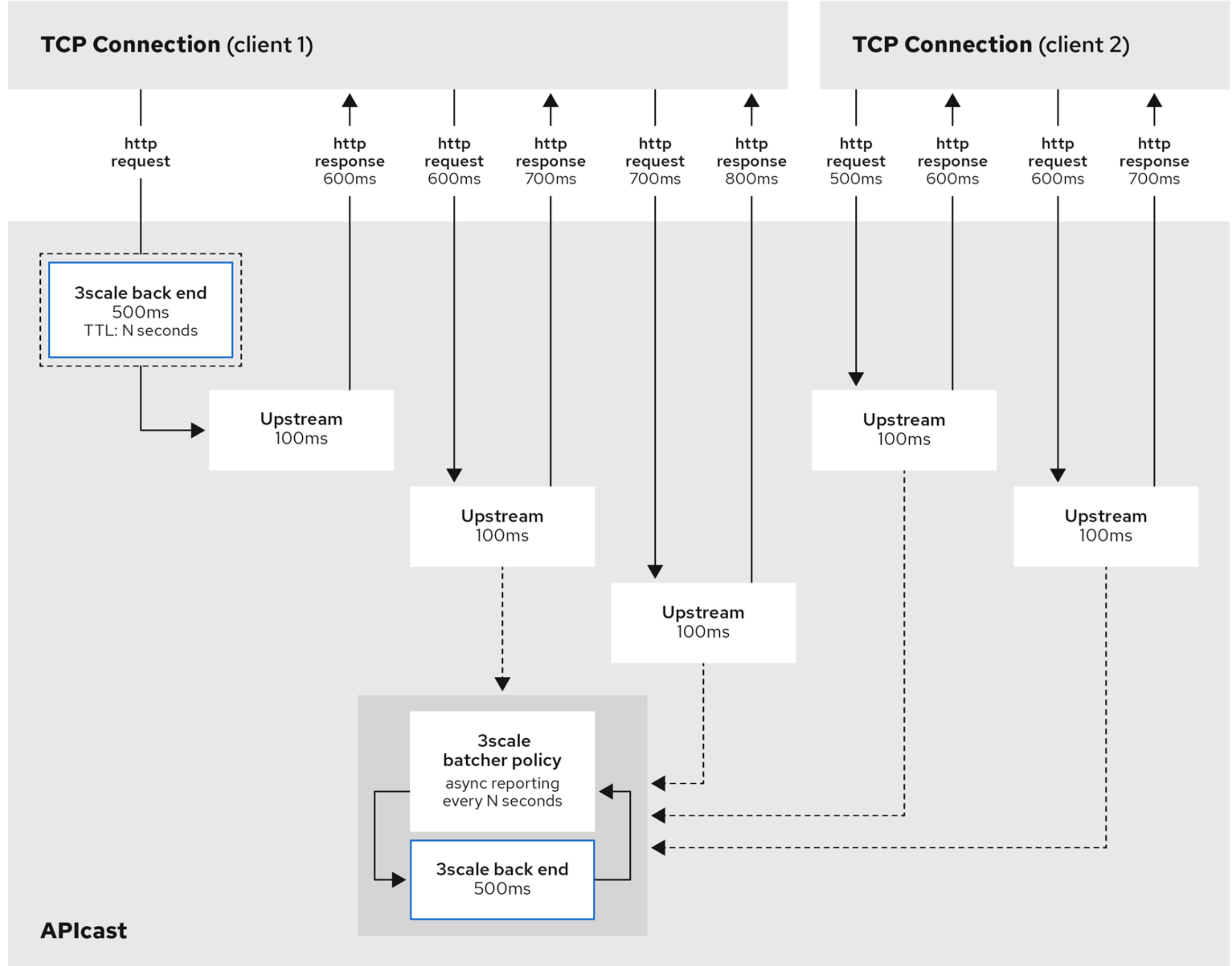


8.4. 3SCALE API 管理 BATCHER 策略

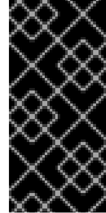
默认情况下，APIcast 会为接收的每个请求对 3scale 后端服务器执行一个调用。3scale API 管理批处理器策略的目标是通过显著减少向 3scale 后端服务器发出的请求数量来缩短延迟并提高吞吐量。为实现

这一目标，此策略会缓存授权状态和批处理报告。

详情请参阅 3scale API 管理 Batcher 策略。下图说明了策略的工作原理。



第 9 章 向 PROMETHEUS 公开 3SCALE API 管理 APICAST METRICS



重要

在这个 3scale 发行版本中，不支持 Prometheus 安装和配置。另外，您可以使用 Prometheus 的社区版本 视觉化 API 服务的指标和警报。

9.1. 关于 PROMETHEUS

Prometheus 是一个开源系统监控工具包，可用于监控 Red Hat OpenShift 环境中部署的 Red Hat 3scale API Management APIcast 服务。

如果要使用 Prometheus 监控服务，您的服务必须公开 Prometheus 端点。此端点是一个 HTTP 接口，公开指标列表和指标的当前值。Prometheus 定期提取这些目标定义的端点，并将收集的数据写入其数据库中。

9.1.1. Prometheus 查询

在 Prometheus UI 中，您可以使用 Prometheus Query Language(PromQL)编写查询来提取指标信息。使用 PromQL，您可以实时选择和聚合时间序列数据。

例如，您可以使用以下查询为指标名称 http_requests_total 选择 Prometheus 在最后 5 分钟内记录的所有值：

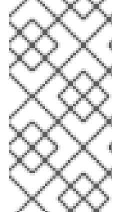
```
http_requests_total[5m]
```

您可以通过为指标指定 *label*（键：值对）来进一步定义或过滤查询的结果。例如，您可以查询以下查询，为指标名称 http_requests_total 和 job 标签设置为 integration 的所有时间序列选择 Prometheus 在最后 5 分钟内记录的所有值：

```
http_requests_total{job="integration"}[5m]
```

查询的结果可以显示为一个图形，在 Prometheus 的表达式浏览器中作为表格数据查看，或者通过使用 Prometheus HTTP API 由外部系统使用。Prometheus 提供数据的图形视图。要获得更强大的图形仪表板来查看 Prometheus 指标，Grafana 是一种常见选择。

您还可以使用 PromQL 语言在 Prometheus alertmanager 工具中配置警报。



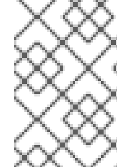
注意

Grafana 是社区支持的功能。红帽生产服务级别协议(SLA)不支持部署 Grafana 来监控 3scale API 管理产品。

9.2. APICAST 与 PROMETHEUS 集成

以下部署选项提供了与 Prometheus 的 APIcast 集成：

- 自我管理 APIcast - 带 3scale 托管或内部 API 管理器。
- 3scale 内部部署的嵌入式 APIcast.



注意

托管 API Manager 和托管的 APIcast 不提供与 Prometheus 的 APIcast 集成。

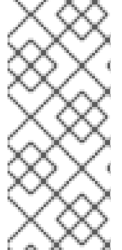
默认情况下，Prometheus 可以监控 表 9.2 “3scale API 管理 APIcast 的 Prometheus 默认指标” 中列出的 APIcast 指标。

9.2.1. 其他选项

另外，如果您有集群管理员对 OpenShift 集群的访问权限，您可以扩展 total_response_time_seconds, upstream_response_time_seconds, and upstream_status metrics to include service_id 和 service_system_name 标签。要扩展这些指标，使用以下命令将 APICAST_EXTENDED_METRICS OpenShift 环境变量设置为 true ：

```
oc set env dc/apicast APICAST_EXTENDED_METRICS=true
```

如果您使用 3scale Batcher 策略（在 第 4.1.3 节 “3scale API 管理 Batcher” 中介绍），Prometheus 还可以监控 表 9.3 “3scale API 管理 APIcast 批处理策略的 Prometheus Metrics” 中列出的指标。



注意

如果指标没有值，Prometheus 会隐藏指标数据。例如，如果 nginx_error_log 没有报告错误，Prometheus 不会显示 nginx_error_log 指标。nginx_error_log 指标只有在有值时才可见。

其他资源

有关 Prometheus 的详情，请参考 Prometheus： Getting Started。

9.3. 3SCALE API 管理 APICAST 的 OPENSHIFT 环境变量

要配置 Prometheus 实例，您可以设置 表 9.1 “3scale API 管理 APIcast 的 Prometheus 环境变量” 中描述的 OpenShift 环境变量。

表 9.1. 3scale API 管理 APIcast 的 Prometheus 环境变量

环境变量	描述	默认
APICAST_EXTENDED_METRICS	一个布尔值，启用 Prometheus 指标的额外信息。以下指标有 **service_id** 和 **service_system_name** 标签，它们提供了有关 APIcast 的更多深入详情： • total_response_time_seconds • upstream_response_time_seconds • upstream_status	false

其他资源

有关设置环境变量的详情，请查看相关的 OpenShift 指南：

- OpenShift 4： 应用程序
- OpenShift 3.11：开发人员指南

- Red Hat 3scale API Management Platform 支持的配置

9.4. 3SCALE API 管理 APICAST 指标公开给 PROMETHEUS

在将 Prometheus 设置为监控 3scale APIcast 后，它默认可以监控 表 9.2 “3scale API 管理 APIcast 的 Prometheus 默认指标” 中列出的指标。

只有在您使用 3scale Batcher 策略时，表 9.3 “3scale API 管理 APIcast 批处理策略的 Prometheus Metrics” 中列出的指标才可用。

表 9.2. 3scale API 管理 APIcast 的 Prometheus 默认指标

指标	描述	类型	标签
nginx_http_connections	HTTP 连接数	gauge	state(accepted,active,handled,reading,total,waiting,writing)
nginx_error_log	APIcast 错误	计数	**level(debug,info,notice,warn,error,crit,alert,emerg)**
openresty_shdict_capacity	worker 共享字典的能力	gauge	**dict**（每个字典对应一个）
openresty_shdict_free_space	worker 共享字典的可用空间	gauge	**dict**（每个字典对应一个）
nginx_metric_errors_total	管理指标的 Lua 库错误数	计数	*none*
total_response_time_seconds	向客户端发送响应所需的时间（以秒为单位） 注 ：要访问 **service_id** 和 **service_system_name** 标签，您必须将 **APICAST_EXTENDED_METRICS** 环境变量设置为 true，如 第 9.2 节 “APIcast 与 Prometheus 集成” 所述。	histogram	**service_id**,**service_system_name**

指标	描述	类型	标签
upstream_response_time_seconds	来自上游服务器的响应时间（以秒为单位） **注**：要访问 **service_id** 和 **service_system_name** 标签，您必须将 **APICAST_EXTENDED_METRICS** 环境变量设置为 **true**，如 第 9.2 节 “APIcast 与 Prometheus 集成” 所述。	histogram	**service_id**,**service_system_name**
upstream_status	来自上游服务器的 HTTP 状态 **注**：要访问 **service_id** 和 **service_system_name** 标签，您必须将 **APICAST_EXTENDED_METRICS** 环境变量设置为 **true**，如 第 9.2 节 “APIcast 与 Prometheus 集成” 所述。	计数	**status**, **service_id**, **service_system_name**
threescale_backend_calls	授权并报告请求到 3scale 后端(Apisonator)	计数	**endpoint**(**authrep**, **auth**, **report**), **status**(**2xx**, **4xx**, **5xx**)

表 9.3. 3scale API 管理 APIcast 批处理策略的 Prometheus Metrics

指标	描述	类型	标签
apicast_status	APIcast 向客户端发送的响应状态数	计数	status
batching_policy_auths_cache_hits	按 3scale 批处理策略的 auths 缓存中的内容	计数	*none*
batching_policy_auths_cache_misses	3scale 批处理策略的 auths 缓存中丢失	计数	*none*

指标	描述	类型	标签
content_caching	通过内容缓存策略的请求数	计数	Status (**MISS**,**BYPASS**,**EXPIRED**,**STALE**,**UPDATING**,**REVALIDATED**,**HIT**)

第 10 章 将 OPENTELEMETRY SDK 与 APICAST 集成

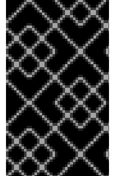
OpenTelemetry SDK 与 APIcast 集成可启用遥测数据导出，以深入了解系统性能和行为。APIcast 依赖于 NGINX OpenTelemetry 追踪库。此集成有助于识别和解决性能问题，从而提高了系统稳定性。

先决条件

- 跟踪 Collector 支持 APIcast exporter 跟踪程序。
  - APIcast 中唯一实现 导出器 是 gRPC 上的 OpenTelemetry 协议(OTLP) （远程过程调用） OTLP/gRPC。
  - APIcast 不使用 OTLP over HTTP (OTLP/HTTP)。
  - 如果现有收集器不支持 APIcast OTLP/gRPC 跟踪，则需要 OpenTelemetry Collector 作为追踪代理。

10.1. 用于部署 OTP 和 GRPC TRACE 的 JAEGER 服务示例

Jaeger 1.35 或更高版本支持 trace 收集器，其中包括 APIcast exporter 功能。现在，Jaeger 可以通过原生的 OpenTelemetry Protocol (OTLP) over gRPC (Remote Procedure Calls) OTLP/gRPC 接收来自 OpenTelemetry SDK 的 trace 数据。



重要

以下示例不适用于生产环境。

部署 Jaeger 的示例

```
oc apply -f - <<EOF
```

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: jaeger
  labels:
    app: jaeger
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jaeger
  template:
    metadata:
      labels:
        app: jaeger
    spec:
      containers:
        - name: jaeger
          image: jaegertracing/all-in-one:latest
          env:
            - name: JAEGER_DISABLED
              value: "false"
            - name: COLLECTOR_OTLP_ENABLED
              value: "true"
          imagePullPolicy: Always
          ports:
            - containerPort: 16686
            - containerPort: 4317
```

```
apiVersion: v1
kind: Service
metadata:
  name: jaeger
  labels:
    app: jaeger
spec:
  ports:
    - port: 16686
      name: http
    - port: 4317
      name: internal
  selector:
    app: jaeger
EOF
```

Jaeger 服务实例将在端口 4317 和 3scale 管理门户（端口 16686）中侦听 OTLP/gRPC 跟踪。

追踪标头示例：

```
"Traceparent": "00-4335058ae8ec72f9636d8c0da08c62be-137a4beaae638572-01",
```

10.2. 配置 APICAST 进行追踪

为确保 3scale API 的可扩展且可靠的网关基础架构，您可以手动配置 APIcast 进行追踪。这将有助于确保 API 服务的可靠且可扩展的网关基础架构。

流程

1. 为 APIcast 工具创建配置文件：

   - 配置文件规格在 NGINX 检测库存储库中定义。

```
$ oc apply -f - <<EOF
```

```
apiVersion: v1
kind: Secret
metadata:
  name: otel-config
type: Opaque
stringData:
  config.json: |
    # "otlp" is the only supported exporter in APIcast
    exporter = "otlp"
    processor = "simple"
    [exporters.otlp]
    # Alternatively the OTEL_EXPORTER_OTLP_ENDPOINT environment variable
can also be used.
    host = "${COLLECTOR_HOST}"
    port = ${COLLECTOR_PORT}
    # Optional: enable SSL, for endpoints that support it
    # use_ssl = true
    # Optional: set a filesystem path to a pem file to be used for SSL encryption
    # (when use_ssl = true)
    # ssl_cert_path = "/path/to/cert.pem"
    [processors.batch]
    max_queue_size = 2048
    schedule_delay_millis = 5000
    max_export_batch_size = 512
    [service]
    name = "apicast" # Opentelemetry resource name
EOF
```

2. 定义指定 openTelemetry 属性的 *APIcast* 自定义资源(CR)。在 CR 定义中，将 openTelemetry.tracingConfigSecretRef.name 属性设置为包含 openTelemetry 配置详情的

secret 的名称。以下示例显示有关配置 openTelemetry 的内容：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  ...
  openTelemetry:
    enabled: true
    tracingConfigSecretRef:
      name: "$NAME_OF_SECRET"
```

10.3. 其他资源

- APIcast 环境变量
- W3C 支持的传播类型

部分 II. API 版本控制

第 11 章 API 版本

红帽 3scale API 管理允许 API 版本控制。当您使用 3scale 管理 API 时，可以通过三种方法正确发布 API：以下是您可以在 3scale 网关中对 API 进行版本的示例，它根据 3scale 架构提供了额外的功能。

11.1. 目标

本指南旨在为您提供足够信息，以便在 3scale 中实施 API 版本系统。

假设您有一个用于查找歌曲的 API。用户可以通过不同的关键字搜索自己喜欢的歌曲：艺术家、歌曲、歌曲标题、相簿标题等等。假设您有 API 的初始版本(v1)，现在您已开发了一个新的改进版本(v2)。

以下小节介绍了使用 3scale 实施 API 版本系统的三种最典型的方法：

- URL 版本控制.
- 端点版本控制.
- 自定义标头版本控制。

11.2. 先决条件

- 在使用此快速开始指南前，请通过 3scale 完成第一步。

11.3. URL 版本

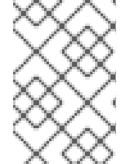
如果您具有不同的用于搜索歌曲的端点（通过工件，按歌曲标题等），使用 URL 版本作为 URI 的一部分包括 API 版本，例如：

1. api.songs.com/v1/songwriter
2. api.songs.com/v2/songwriter

3. **api.songs.com/v1/song**

4. **api.songs.com/v2/song**

5. **以此类推**



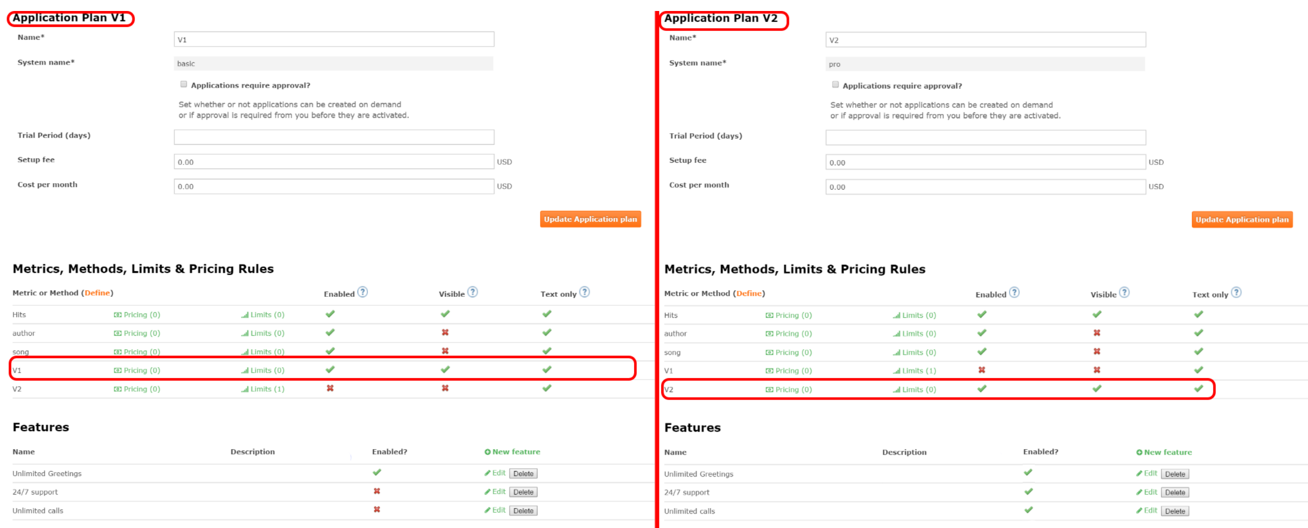
注意

使用此方法时，自 v1 准备对 API 进行版本之后，您应已进行了规划。

然后，3scale 网关将从 URI 中提取端点和版本。这种方法允许您为任何版本/端点组合设置应用计划。然后，您可以将指标与这些计划和端点关联，您可以绘制各个版本上每个端点的使用情况。

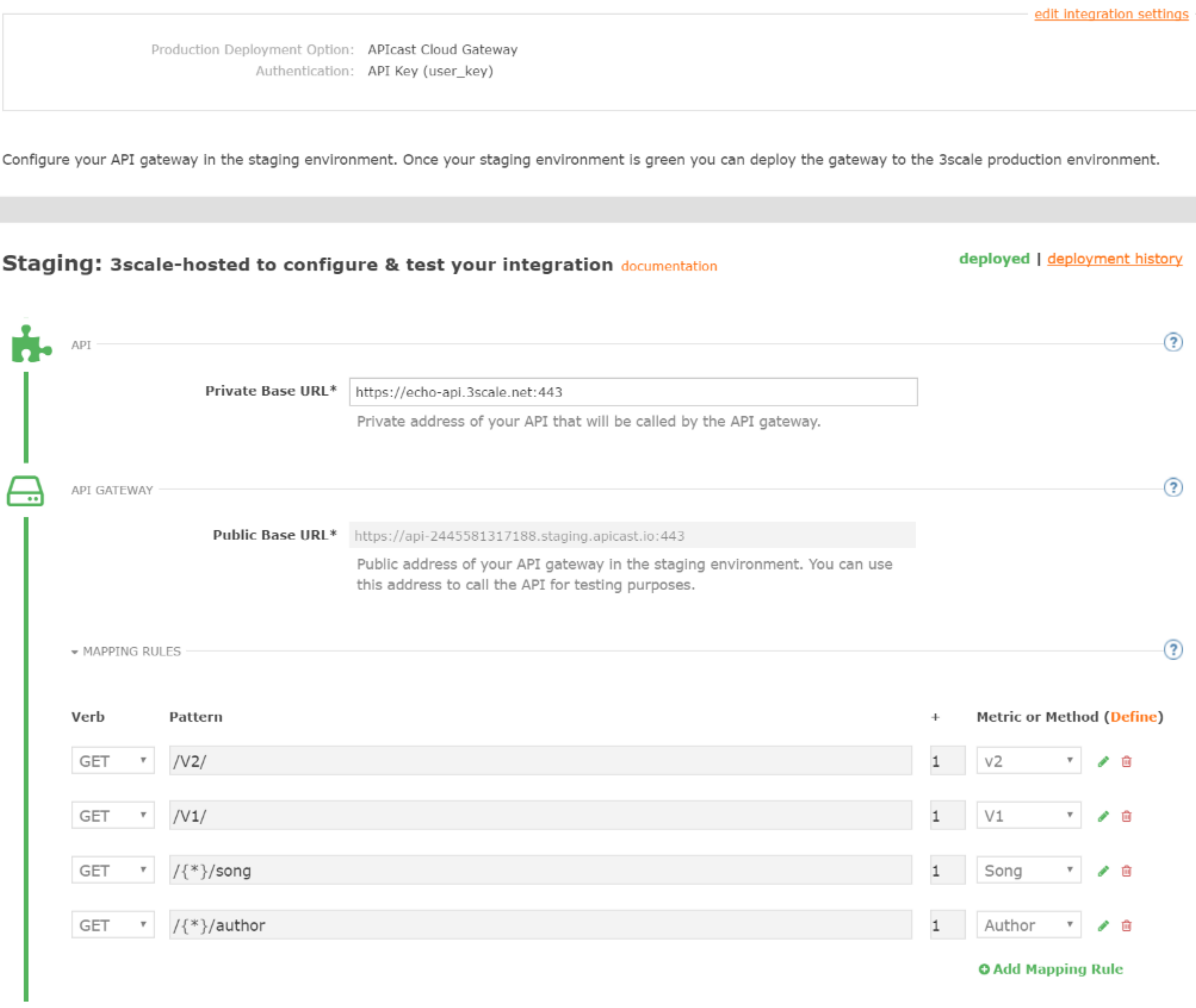
以下屏幕截图显示了 3scale 的灵活性。

图 11.1. 版本计划功能



唯一需要做的是进入 3scale 管理门户中的 [your_API_name] > Integration > Configuration，并将您的 URI 映射到您的指标，如下图所示。

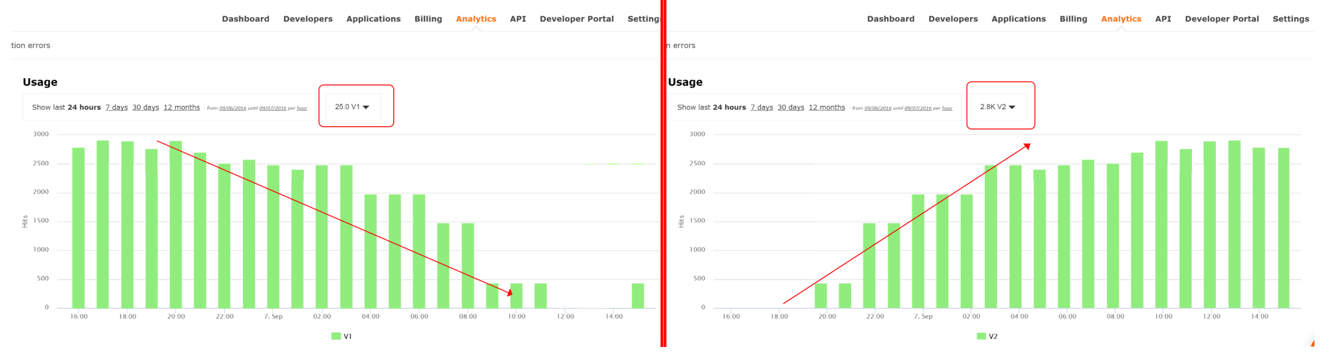
图 11.2. 将 URI 映射到指标



您现在有两个不同的 API 版本，各自启用了不同的功能。您对它们的使用具有完全的控制权和可见性。

如果您要与应移至 API v2 的所有用户通信，您可以发送内部备注要求他们这样做。您可以监控谁进行了移动，并了解 v1 上的活动在 v2 活动增加期间如何减少。通过在授权调用中向 3scale 添加指标，您可以看到整体流量到达 v1 与 v2 端点，并了解何时可以安全地弃用 v1。

图 11.3. 版本控制



如果某些用户使用 v1，则只能过滤出那些用户，以发送有关切换到 v2 的另一个内部备注。

3scale 提供三步方法来发送弃用通知。

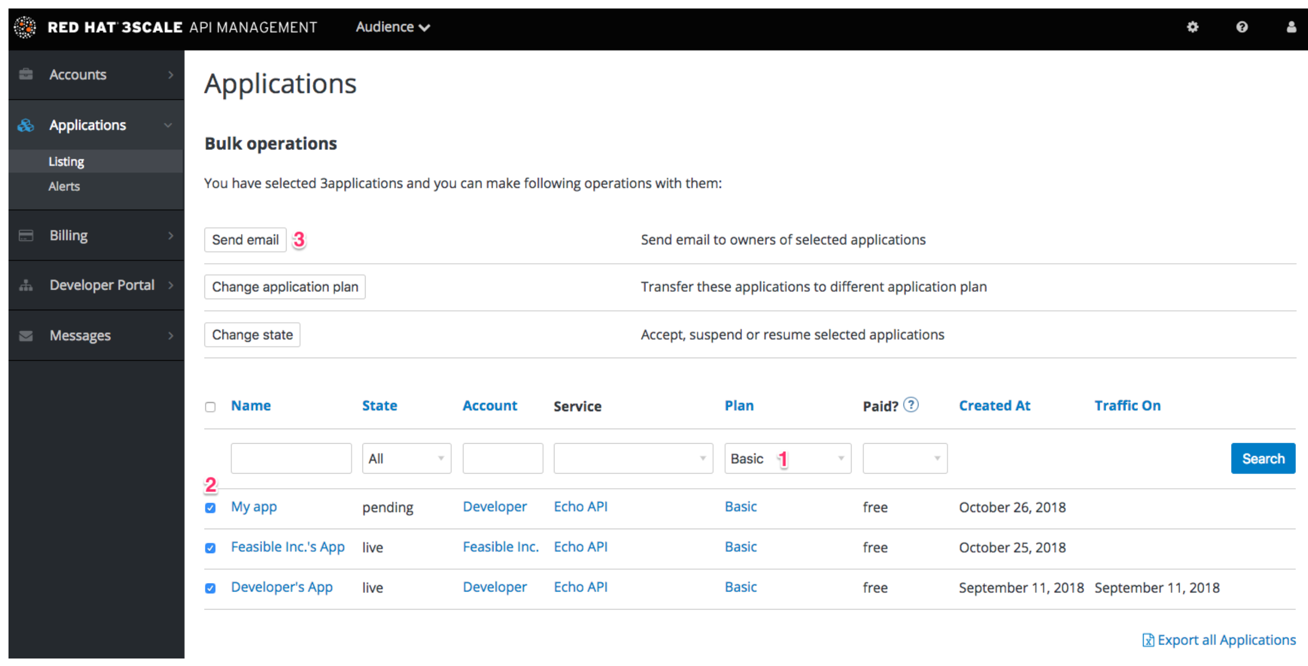
1. 导航到 Audience > Applications > Listing，并根据您要发送弃用备注的应用程序计划过滤列表并点击 Search。

2. 单击 multiselector，以选择该特定版本的所有应用。显示新的选项，并允许您执行批量操作，如 发送电子邮件、更改应用程序计划和更改状态。

3. 点 Send email，然后按照步骤将弃用通知发送到所选应用程序的所有者。

下图提供了一个可视化参考。

图 11.4. 发送弃用备注



对于对端点发出的每个 authrep 调用，您仅进行身份验证一次，但为端点报告两次，一次报告一次用于 API 版本。没有可重复性，因为调用只能一次被验证。对于您对特定 API 版本的任何端点进行的调用，您可以将点击聚合到以版本号（v1、v2 等）命名的方便指标上，您可以使用它来相互比较完整的版本流量。

11.4. 端点版本控制

通过端点版本控制，您可以为每个 API 版本使用不同的端点，如 api.cons.com/author_v1。网关从端点本身提取端点和版本。此方法以及上述方法允许 API 提供程序将外部 URL 映射到内部 URL。

端点版本方法只能通过内部部署方法执行，因为它需要使用作为内部配置一部分提供的 LUA 脚本来重写 URL。

EXTERNAL		INTERNAL
api.songs.com/songwriter_v1	可被重写为	internal.songs.com/search_by_songwriter
api.songs.com/songwriter_v2	可被重写为	internal.songs.com/songwriter

几乎所有（映射、应用程序计划功能等）的工作方式都与之前的方法完全相同。

11.5. 自定义标头版本

使用自定义标头版本控制时，您可以使用标头（即 "x-api-version"）而不是 URI 来指定版本。

然后，网关从该路径和标头中提取端点。就像以前一样，您可以分析和视觉化任何您想要的路径/版本组合。这种方法存在一些不便，无论您使用的 API 管理系统是什么。如需了解更多信息，请参阅 API 版本方法。以下是 3scale 工作原理的几个要点：

- 与前面的方法一样，自定义标头版本控制只能应用到内部托管 API，因为它需要一些请求标头解析/处理才能正确路由 authrep 调用。这种类型的自定义处理只能通过 Lua 脚本进行。

- 使用此方法时，很难实现之前方法的细粒度功能分隔。

- 这种方法最重要的优势在于，开发人员指定的 URL 和端点永远不会更改。当开发人员希望从一个 API 版本切换到另一个 API 版本时，他们只需要更改标头。其他所有功能都相同。

部分 III. API 身份验证

第 12 章 身份验证模式

在本教程结束时，您将了解如何在 API 上设置身份验证模式，以及对应用程序与 API 通信的影响。

根据您的 API，您可能需要使用不同的身份验证模式来发布凭据以访问您的 API。它们可以从 API 密钥到 openAuth 令牌和自定义配置。本教程介绍了如何从可用的标准身份验证模式中进行选择。

12.1. 支持的验证模式

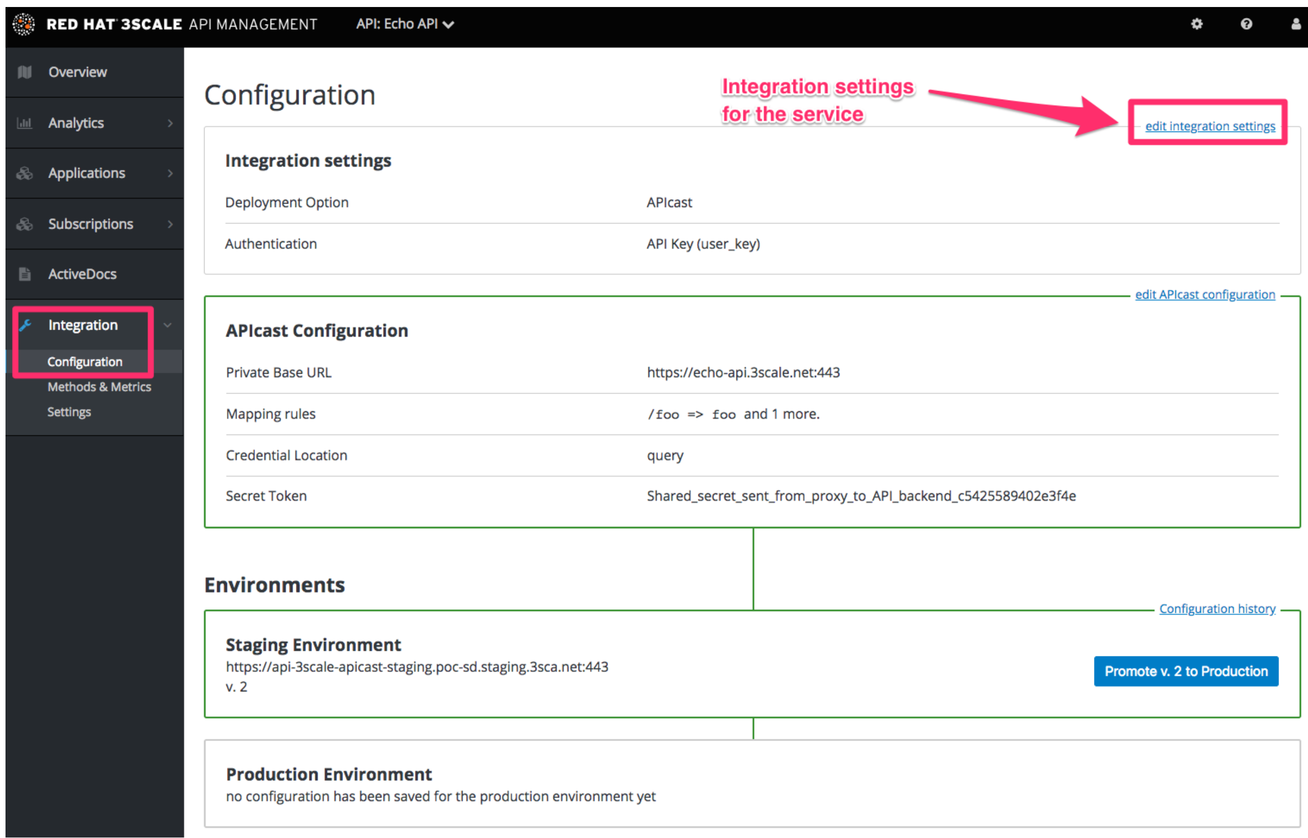
Red Hat 3scale API Management 支持以下开箱即用的身份验证模式：

- 标准 API 键：使用一个随机字符串或哈希值作为标识符和 secret 令牌。
- 应用程序标识符和密钥对 ：可驱逐的标识符和可变的 secret key 字符串。
- OpenID Connect

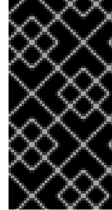
12.2. 设置身份验证模式

12.2.1. 选择服务的验证模式

导航到您要处理的 API 服务（在这种情况下，可能只有一个名为 API 的服务）。前往 Integration 部分。



您运行的每个服务都可以使用不同的身份验证模式，但每个服务只能使用一种模式。



重要

在注册了凭证后，您不得更改身份验证模式，因为服务的行为可能会变得无法预测。要更改身份验证模式，我们建议创建新服务并迁移客户。

12.2.2. 选择要使用的 Authentication 模式

若要选择身份验证模式，可滚动到 AUTHENTICATION 部分。在这里，您可以选择以下选项之一：

- API 密钥(user_key)
- App_ID 和 App_Key Pair
- OpenID Connect

12.2.3. 确保您的 API 接受正确类型的凭证

根据所选的凭证类型，您可能需要接受 API 调用中的不同参数（密钥字段、ID 等）。这些参数的名称可能与 3scale 中内部使用的参数不同。如果在 3scale 后端调用中使用了正确的参数名称，3scale 身份验证将正确运行。

12.2.4. 创建用于测试凭证的应用程序

为确保凭据集正常工作，您可以创建一个新应用来签发凭据以使用 API。导航到管理门户控制面板的 Accounts 区域，单击要使用的帐户，再单击 新应用程序。

填写表单并单击 save 将创建一个包含凭据的新应用，以使用 API。现在，您可以使用这些凭证来调用 API，并且会根据 3scale 中注册的应用程序列表检查记录。

12.3. 标准验证模式

3scale 支持以下部分中详述的身份验证模式。

12.3.1. API 密钥

支持最简单的凭证形式是单一 API 模型。在这里，每个具有 API 权限的应用程序都有单个（唯一）长字符串；例如：

```
API-key = 853a76f7c8d5f4a1ee8bf10a4e0d1f13
```

默认情况下，key 参数的名称是 user_key。您可以使用该标签或选择另一个标签，如 API-key。如果选择其他标签，则需要在向 3scale 发出授权调用前映射该值。字符串同时充当标识符和机密令牌，供 API 使用。建议您仅在安全要求低或 API 调用的 SSL 安全性的环境中使用这些模式。以下是可以在令牌和应用程序上执行的操作：

- 应用 Suspend ：这会挂起应用程序对 API 的访问，并且会暂停对带有相关密钥的 API 的所有调用。
- 应用程序 Resume：取消应用暂停操作的影响。
- 主要重新生成：此操作为应用生成一个新的随机字符串密钥，并将它与应用相关联。立即执行此操作后，对前一个令牌的调用将不再被接受。

后一操作可以从管理门户中的 API 管理以及（如果允许）从 API Developers 用户控制台触发。

12.3.2. App_ID 和 App_Key 对

API Key Pattern 将应用程序的身份和 secret 用量令牌组合在一个令牌中，但这种模式将两者分开：

- 使用 API 的每个应用都发出不可变的初始标识符，称为 应用 ID(App ID)。App ID 是恒定的，不一定是保密的。
- 此外，每个应用程序都可以有一到五个 应用密钥 (App_Keys)。每个密钥都直接与 App_ID 关联，应视为机密。

```
app_id = 80a4e03 app_key = a1ee8bf10a4e0d1f13853a76f7c8d5f4
```

在默认设置中，开发人员可以为每个应用创建最多五个密钥。这允许开发人员创建新密钥，将它添加到代码中，重新部署其应用，然后禁用旧密钥。这不会以 API 密钥重新生成的方式造成应用程序停机。

统计信息和速率限值始终保持在应用程序 ID 的粒度级别，而不是每个 API 密钥。如果开发人员希望跟踪两组统计数据，他们应创建两个应用，而不是两个键。

也可以更改系统上的模式，并允许在没有应用密钥的情况下创建应用程序。在这种情况下，3scale 系统将仅根据 App ID 验证访问权限（不进行密钥检查）。此模式可用于小部件类型场景，或者对用户而非应用程序应用速率限制。在大多数情况下，您可能希望 API 强制每个应用程序存在至少一个应用程序密钥。此设置包括在 [your_API_name] > Integration > Settings 中。

12.3.3. OpenID Connect

有关 OpenID Connect 身份验证的详情，请查看 OpenID Connect 集成 章节。

12.4. 参考器过滤

3scale 支持转译器过滤器功能，可用于将应用程序可以访问 API 的 IP 地址或域名列入白名单。API 客户端在 Referrer 标头中指定 referrer 值。使用 Referrer 标头的目的请参考 RFC 7231，第 5.5.2 节： 推荐器。*Referer* 是 RFC 和 标头中引用的官方拼写错误。

要使 Rerer Filtering 功能正常工作，您必须在服务策略链中启用 APIcast Referrer policy。

要启用 Adrer Filtering 功能，请转至 [your_API_name] > Applications > Settings > Usage Rules。选择 Require referr filter 并点击 Update Product。

3scale BY RED HAT

Dashboard Developers Applications Billing Analytics API Developer Portal

Overview ActiveDocs

Definition
Integration
Application Plans
Settings
Alerts

Echo API > Settings

DEFAULT SERVICE PLAN

Default plan

Default

Default service plan (if any) is contracted automatically on sign up.

APPLICATION REQUIREMENTS

Developers can manage applications

Developers with access to your API will be able to manage applications and their access keys.

Require referrer filtering

Developers with access to your API must indicate allowed domain / IP referrers.

Enable custom keys

Allows you to create custom keys for developers

有权访问您的 API 的开发人员必须配置开发人员门户允许的域/IP 引用器。

Referrer Filters

If you are developing a server based application you typically need to add IP addresses, if it is widget based you typically need to add domain names. Specify allowed referrer domains or IP addresses. Wildcards (*.example.org) are also accepted.

Add

At most 5 referrer filters are allowed.

developer.example.com Delete

169.34.21.42 Delete

在 Admin Portal on the application details 页中，所有属于该服务的应用程序详情页面中会显示一个新的 Referencerer Filters 部分。此处，管理员也可以为此应用程序配置允许参考器标头值的白名单。

Referrer Filters

Specify allowed referrer domains or IP addresses. Wildcards (*.example.org) are also accepted.

Add Filter

developer.example.com Delete

169.34.21.42 Delete

您可以为每个应用程序设置最多五个引用者值。

*该值只能包含拉丁字母、数字和特殊字符 *、. 和 -。* 可用于通配符值。如果值设为 *，则允许任何引用器值，因此将绕过引用器检查。*

*当 **Require referrer** 过滤 功能和 **3scale API** 管理推荐器 策略时，授权可以正常工作：*

1. *没有指定推荐过滤器的应用程序通常只使用提供的凭证获得授权。*

2. *对于设置有引用器过滤器值的应用程序，**APIcast** 从请求的引用者标头中提取 引用器 值，并将它作为 **AuthRep**（授权和报告）请求中的 引用器 **param** 发送到 **Service Management API**。下表显示了针对引用器过滤参数的不同组合的 **AuthRep** 响应。*

referrer 参数通过？	为应用配置了参考器过滤器？	引用器参数值	HTTP 响应	响应正文
是	是	匹配引用器过滤器	200 OK	**<status><authorized>true</authorized></status>**

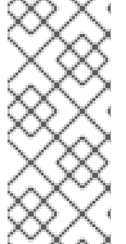
referrer 参数通过？	为应用配置了参考器过滤器？	引用器参数值	HTTP 响应	响应正文
是	否	匹配引用器过滤器	200 OK	**<status><authorized>true</authorized></status>**
是	是	不匹配引用器过滤器	409 冲突	**<status><authorized>false</authorized><reason>referrer "test.example.com" is not allowed</reason>** (**test.example.com** 是一个示例)
是	否	不匹配引用器过滤器	200 OK	**<status><authorized>true</authorized></status>**
是	是	*	200 OK	**<status><authorized>true</authorized></status>**
是	否	*	200 OK	**<status><authorized>true</authorized></status>**
否	是	–	409 冲突	**<status><authorized>false</authorized><reason>referrer is missing</reason>**
否	否	–	200 OK	**<status><authorized>true</authorized></status>**

未经 AuthRep 授权的调用将被 APIcast 拒绝，并显示"Authorization Failed"错误。您可以在服务集

成页面上配置确切的状态代码和错误消息。

第 13 章 将 3SCALE API 管理与 OPENID CONNECT 身份提供程序集成

为验证 API 请求，红帽 3scale API 管理可以与符合 OpenID Connect 规格的 身份提供程序集成。为了与 3scale 完整兼容，身份提供程序可以是 Red Hat Single Sign-On(RH-SSO) 或实施 默认 Keycloak 客户端注册的 第三方身份提供程序。为了与 3scale API 网关(APIcast)兼容，可以使用任何实现 OpenID Connect 的用户身份供应商。



注意

3scale 不使用 RFC 7591 Dynamic Client Registration Mechanism。对于 3scale 和 OpenID Connect 身份提供程序之间的完全兼容性，它依赖于默认 Keycloak 客户端注册。

OpenID Connect 的基础是 OAuth 2.0 授权框架 (RFC 6749)。OpenID Connect 在 API 请求中使用 JSON Web Token(JWT)(RFC 7519) 以验证该请求。当您将 3scale 与 OpenID Connect 身份提供程序集成时，进程有两个主要部分：

- APIcast 在请求中解析和验证 JWT。如果成功，APIcast 验证 API 消费者客户端应用的身份。
- 3scale Zync 组件将 3scale 应用程序详情与 OpenID Connect 身份提供程序同步。

当 RH-SSO 是 OpenID Connect 身份提供程序时，3scale 支持这两个集成点。请参阅 支持的 Configurations 页面中的 RH-SSO 版本。但是，RH-SSO 并不是必需的。您可以使用支持 OpenID Connect 规格和默认 Keycloak 客户端注册的任何身份提供程序。APIcast 集成与 RH-SSO 和 ForgeRock 测试。

以下小节提供了配置 3scale 以使用 OpenID Connect 身份提供程序的信息和说明：

- 集成 3scale API 管理和 OpenID Connect 身份提供程序概述
- APIcast 如何处理 JSON Web 令牌
- 3scale API 管理 Zync 如何将应用程序详情与 OpenID Connect 身份提供程序同步

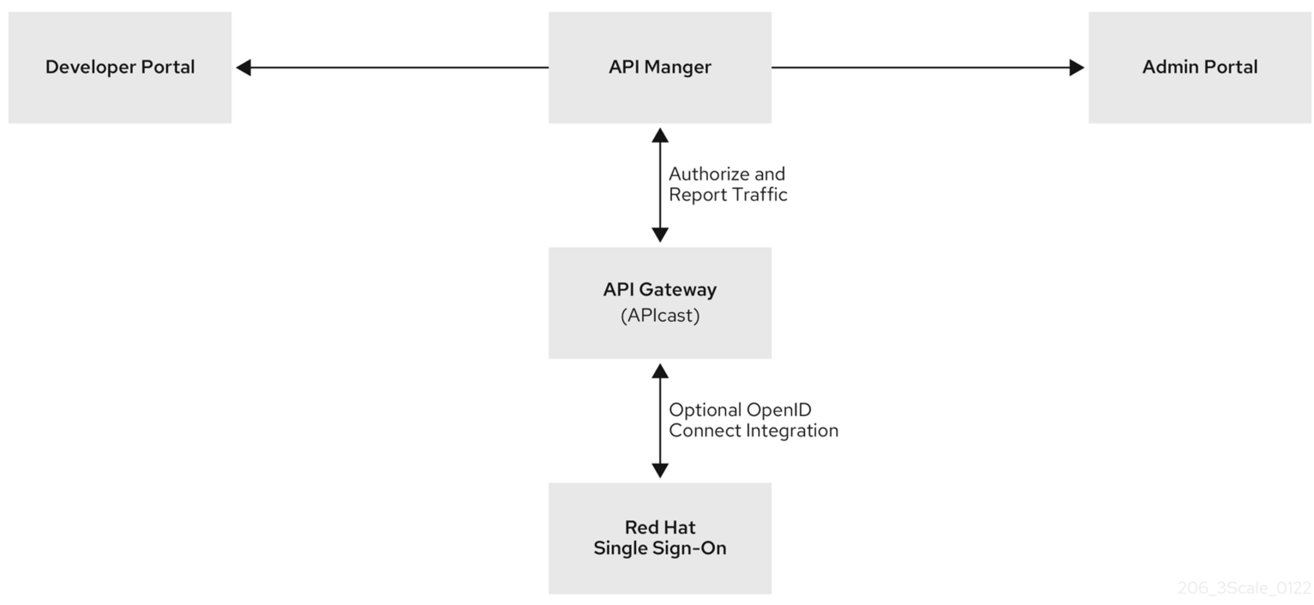
- *将 3scale API 管理与红帽单点登录集成为 OpenID Connect 身份提供程序*
- *将 3scale API 管理与第三方 OpenID Connect 身份提供程序集成*
- *使用 OpenID Connect 身份提供程序测试 3scale API 管理*
- *3scale API 管理与 OpenID Connect 身份提供程序集成的示例*

13.1. 集成 3SCALE API 管理和 OPENID CONNECT 身份提供程序概述

每个主要的 Red Hat 3scale API Management 组件都参与身份验证，如下所示：

- *APIcast 验证 API 消费者应用提供的身份验证令牌的真实性。在默认的 3scale 部署中，APIcast 可以做到这一点，因为它实现了 API 产品的 OpenID Connect 配置自动发现。*
- *API 供应商使用 Admin Portal 设置身份验证流程。*
- *如果 3scale 管理的 API 没有使用标准 API 密钥或应用程序标识符和密钥对验证请求，那么 API 供应商必须与 OpenID Connect 身份提供程序集成。在下图中，OpenID Connect 身份提供程序是红帽单点登录。*
- *通过配置了身份验证和实时开发人员门户，API 使用者使用开发者门户订阅应用程序计划，提供对特定 3scale API 产品的访问。*
- *当 OpenID Connect 与 3scale 集成时，订阅会触发 API 消费者应用程序的配置流，以获取来自 OpenID Connect 身份提供程序的 JSON Web 令牌 (JWT)。API 供应商在配置 API 产品以使用 OpenID Connect 时指定这个流。*

图 13.1. 显示带有 OpenID Connect 身份提供程序的主要 3scale 组件



订阅应用程序计划后，API 使用者会从集成的 OpenID Connect 身份提供程序中接收身份验证凭据。这些凭据支持对 API 消费者应用发送到上游 API 的请求进行身份验证，这是 API 使用者有权访问的 3scale API 产品提供的 API。

凭证包括客户端 ID 和客户端 secret。由 API 使用者创建的应用使用这些凭证从 OpenID Connect 身份提供程序中获取 JSON Web 令牌 (JWT)。当您配置 3scale 与 OpenID Connect 集成时，您可以选择 API 消费者应用如何获取 JWT 的流。在使用带有 Red Hat Single Sign-On 的默认 授权代码 流的 API 消费者应用程序中，应用程序必须执行以下操作：

1. 在第一个请求上游 API 后端前，启动使用 OpenID Connect 身份提供程序的 OAuth 授权流。授权代码流将最终用户重定向到 Red Hat Single Sign-On。最终用户登录，以获取授权代码。

2. 交换 JWT 的授权代码。

3. 身份验证时从红帽单点登录接收 JWT。

4. 发送包含 JWT 的 API 请求到上游 API 后端。

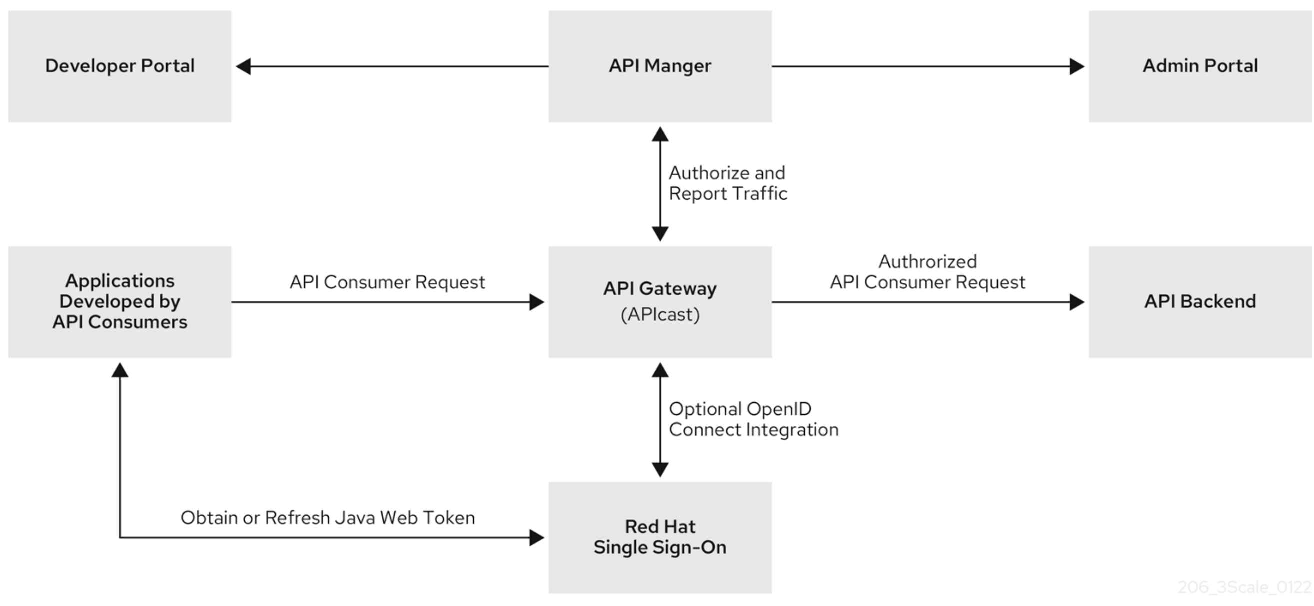
5. 使用同一 JWT 发送后续的 API 请求，直到其过期。

6.

刷新 JWT 或向 OpenID Connect 身份提供程序发送新请求，以获取新的 JWT。需要哪个操作取决于 OpenID Connect 身份提供程序。

APIcast 从 API 用户接收请求并检查请求中的 JWT。如果 APIcast 验证 JWT，APIcast 会将请求（包括 JWT）发送到上游 API 后端。

图 13.2. 显示 OpenID Connect 身份提供程序是 Red Hat Single Sign-On，但可以配置其他 OpenID Connect 身份提供程序。



13.2. APICAST 如何处理 JSON WEB 令牌

APIcast 通过检查 JSON Web 令牌 (JWT) 来处理每个请求，该请求在对 API 消费者应用进行身份验证时返回了 OpenID Connect 身份提供程序。该请求包含由集成的 OpenID Connect 身份提供程序发布的格式的 JWT。JWT 必须位于 Authorization 标头中，它必须使用 Bearer 模式。例如，标头应类似如下：

```
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2lkcC5leGFtcGxlLmNvbSIsInN1Yi
6ImFiYzEyMyIsIm5iZiI6MTUzNzg5MjQ5NCwiZXhwIjoxNTM3ODk2MDk0LCJpYXQiOjE1Mzc4OTI0OT
QsImp0aSI6ImlkMTIzNDU2IiwidHlwIjoiQmVhcmVyIn0.LM2PSmQ0k8mR7eDS_Z8iRdGta-Ea-
pJRrf4C6bAiKz-Nzhxpm7fF7oV3BOipFmimwkQ_-mw3kN--
oOc3vU1RE4FTCQGbzO1SAWHOZqG5ZUx5ugaASY-
hUHIohy6PC7dQl0e2NlAeqqg4MuZtEwrpESJW-
VnGdljrAS0HsXzd6nENM0Z_ofo4ZdTKvIKsk2KrdyVBOcjgVjYongtppR0cw30FwnpqfeCkuATeINN5OK
HXOibRA24pQyIF1s81nnmxLnjnVbu24SFE34aMGRXYzs4icMI8sK65eKxbvwV3PIG3mM0C4ilZPO26d
oP0YrLfVwFcqEirmENUAcHXz7NuvA
```

有很多开源工具可安全解码 JWT。请注意，您不会在公共 Web 工具中解码 JWT。在解码的 JWT 中，您可以看到令牌有三个部分：

- 标头提供有关如何对令牌进行格式的信息，以及用于为令牌签名的算法的信息。

- 有效负载标识发送请求的 API 使用者。详情可包含此 API 使用者可以执行的读取和写入操作、API 消费者的电子邮件地址以及有关 API 消费者的其他信息。

- 签名是一个加密签名，表示令牌没有被更改。

APIcast 检查 JWT 是否有以下特征：

- 完整性

  JWT 是否被恶意用户更改？签名是否有效？

  JWT 包含了一个签名，令牌接收器可以验证该令牌，以确保已知签发者签名。此验证还可保证其内容仍然尽可能创建。3scale 支持基于公钥/私钥对的 RSA 签名。签发者使用私钥对 JWT 进行签名。APIcast 使用公钥验证令牌。APIcast 使用 OpenID Connect Discovery 获取可用于验证 JWT 签名的 JSON Web 密钥(JWK)。

- 时间

  当令牌可以接受处理时，当前的时间是多久？JWT 是否已过期？换句话说，APIcast 会检查 JWT nbf （而非时间之前）和 exp（过期时间）声明。

- 发布者

  已知为 APIcast 的 OpenID Connect 身份提供程序发布的 JWT 吗？换句话说，APIcast 验证 JWT 中指定的签发者是与 OpenID Connect Issuer 字段中配置的 API 供应商相同的签发者。签发者规格是集成 3scale 和 OpenID Connect 身份提供程序的步骤的一部分。这是 JWT iss 声明。

- 客户端 ID

  令牌是否包含 3scale 客户端应用程序 ID，它对于 APIcast 已知？此客户端 ID 必须与在流程中指定的 API 供应商与 OpenID Connect 身份提供程序集成的步骤中的 ClientID Token Claim

匹配。这是 JWT azp （JWT 签发到的授权方）声明和 aud (听众) 声明。

如果有任何 JWT 验证或授权检查失败，APIcast 会返回 Authenication 失败 错误。否则，APIcast 将请求发送到 3scale 上游 API 后端。Authorization 标头保留在请求中，因此 API 后端也可以使用 JWT 检查用户和客户端身份。

13.3. 3SCALE API 管理 ZYNC 如何将应用程序详情与 OPENID CONNECT 身份提供程序同步

Zync 是一个 3scale 组件，可可靠地将数据推送到 OpenID Connect 身份提供程序中。在此交互中，3scale 应用与 OpenID Connect 身份提供程序客户端对应。换句话说，Zync 与 OpenID Connect 身份提供程序通信，以创建、更新和删除 OpenID Connect 客户端。

Zync 实现 Keycloak 默认客户端注册。使用此 API 意味着客户端代表特定于 Keycloak 和 Red Hat Single Sign-On。身份提供程序会返回客户端 ID 和客户端 secret，它们是 3scale 应用的身份验证凭据。

每次创建、更新或删除应用程序时，Zync 与 OpenID Connect 身份提供程序通信，以相应地更新对应的客户端。成功同步需要给定 3scale 产品的以下设置：

- 身份验证机制为 OpenID Connect。
- OpenID Connect 签发者类型是：
  - 当 Red Hat Single Sign-On 是 OpenID Connect 身份提供程序时，Red Hat Single Sign-On。使用这个签发者类型，Zync 会将客户端注册请求发送到 Keycloak/Red Hat Single Sign-On 默认客户端注册 API。
  - 其他 OpenID Connect 身份提供程序的 REST API。使用这个签发者类型，Zync 会发送客户端注册请求，如 Zync REST API 示例中所示。
- 如下 URL 是 OpenID Connect Issuer:http://id:/api_endpoint。

当部署到 OpenShift 集群时，有两个 Zync 进程：

- zync

  Zync 是一个 REST API，从 system-sidekiq 接收通知，并将后台作业排队到 zync-que。有针对新的、更新和删除 3scale 应用程序的通知。

- zync-que

  zync-que 处理这些后台作业，它们与 system-app 和 OpenID Connect 身份提供程序通信。例如，当 Red Hat Single Sign-On 是配置的 OpenID Connect 身份提供程序时，Zync 创建、更新和删除 Red Hat Single Sign-On 域中的客户端。

13.4. 将 3SCALE API 管理与红帽单点登录集成为 OPENID CONNECT 身份提供程序

作为 API 供应商，您可以将 Red Hat 3scale API Management 与 Red Hat Single Sign-On 集成，作为 OpenID Connect 身份提供程序。这里记录的流程适用于需要 OpenID Connect 进行身份验证 API 请求的 3scale API 产品。

此流程的一部分是在 3scale Zync 和 Red Hat Single Sign-On 之间建立 SSL 连接，因为 Zync 与红帽单点登录通信以交换令牌。如果您没有在 Zync 和红帽单点登录之间配置 SSL 连接，则会为侦听的任何人打开令牌。

3scale 2.2 及更新的版本支持带有 SSL_CERT_FILE 环境变量的 Red Hat Single Sign-On 的自定义 CA 证书。此变量指向证书捆绑包的本地路径。将 3scale 与红帽单点登录集成为 OpenID Connect 身份提供程序包括按以下顺序配置以下元素：

- 如果 Red Hat Single Sign-On 没有使用 由可信证书颁发机构(CA)发布的证书，您必须将 3scale Zync 配置为使用自定义 CA 证书。如果 Red Hat Single Sign-On 使用由可信 CA 发布的证书，则不需要此项。

- 配置 Red Hat Single Sign-On 以具有 3scale 客户端。

- 配置 3scale 以使用 Red Hat Single Sign-On。

先决条件

- Red Hat Single Sign-On 服务器必须通过 HTTPS 提供，且必须通过 zync-que 访问。要测试这一点，您可以从 zync-que pod 中运行 curl https://rhsso-fqdn，例如：

```
$ oc rsh -n $THREESCALE_PROJECT $(oc get pods -n $THREESCALE_PROJECT --field-selector=status.phase==Running -o name | grep zync-que) /bin/bash -c "curl -v https://<rhsso-fqdn>/auth/realms/master"
```

- OpenShift 集群管理员权限。
- 您要配置 OpenID Connect 与红帽单点登录集成的 3scale API 产品。

详情请查看以下部分：

- 配置 3scale API 管理 Zync 以使用自定义证书颁发机构证书
- 配置红帽单点登录使其具有 3scale API 管理客户端
- 配置 3scale API 管理以使用 Red Hat Single Sign-On

13.4.1. 配置 3scale API 管理 Zync 以使用自定义证书颁发机构证书

当 Red Hat Single Sign-On 使用由可信证书颁发机构(CA)发布的证书时，则不需要此项。但是，如果 Red Hat Single Sign-On 没有使用 受信任 CA 发布的证书，您必须配置 Red Hat 3scale API Management Zync，然后才能将 Red Hat Single Sign-On 配置为具有 3scale 客户端，并在配置 3scale 以使用 Red Hat Single Sign-On 前。

流程

1. 以 .pem 格式获取 CA 证书链，并将每个证书保存为单独的文件，例如：customCA1.pem、CustomCA2.pem 等等。

2. 测试每个证书文件，以确认它是有效的 CA。例如：

```
$ openssl x509 -in customCA1.pem -noout -text | grep "CA:"
```

   这个输出为 CA:TRUE 或 CA:FALSE。您希望每个证书文件为 CA:TRUE。如果输出为 CA:FALSE，则证书不是有效的 CA。

3. 使用以下 cURL 命令验证每个证书文件。例如：

```
$ curl -v https://<secure-sso-host>/auth/realms/master --cacert customCA1.pem
```

将 <secure-sso-host > 替换为 Red Hat Single Sign-On 主机的完全限定域名。

预期的响应是 Red Hat Single Sign-On 域的 JSON 配置。如果验证失败，您的证书可能不正确。

4. 在 zync-que pod 上收集 /etc/pki/tls/cert.pem 文件的现有内容：

```
$ oc exec <zync-que-pod-id> -- cat /etc/pki/tls/cert.pem > zync.pem
```

5. 将每个自定义 CA 证书文件的内容附加到 zync.pem，例如：

```
$ cat customCA1.pem customCA2.pem ... >> zync.pem
```

6. 将新文件附加到 zync-que pod 作为 configmap 对象：

```
$ oc create configmap zync-ca-bundle --from-file=./zync.pem
$ oc set volume dc/zync-que --add --name=zync-ca-bundle --mount-path
/etc/pki/tls/zync/zync.pem --sub-path zync.pem --source='{"configMap":{"name":"zync-ca-bundle","items":[{"key":"zync.pem","path":"zync.pem"}]}}'
```

这会完成将证书捆绑包添加到 zync-que pod 中。

7. 验证证书是否已附加，内容是否正确：

```
$ oc exec <zync-que-pod-id> -- cat /etc/pki/tls/zync/zync.pem
```

8. 在 Zync 上配置 SSL_CERT_FILE 环境变量以指向新的 CA 证书捆绑包：

```
$ oc set env dc/zync-que SSL_CERT_FILE=/etc/pki/tls/zync/zync.pem
```

13.4.2. 配置红帽单点登录使其具有 3scale API 管理客户端

在 OpenShift Red Hat Single Sign-On 仪表板中，将 Red Hat Single Sign-On 配置为具有 Red Hat 3scale API Management 客户端。这是特殊的管理客户端。每次 API 使用者订阅开发人员门户中的 API 时，3scale 使用您在此流程中创建的 Red Hat Single Sign-On 管理客户端来为 API 消费者应用程序创建客户端。

流程

1. 在 Red Hat Single Sign-On 控制台中，为 3scale 客户端创建一个域，或选择包含 3scale 客户端的现有域。https://access.redhat.com/documentation/zh-cn/red_hat_single_sign-on/7.5/html/getting_started_guide/creating-first-realm_

2. 在新的或选定域中，点左侧导航面板中的 Clients。

3. 点 Create 创建新客户端。

4. 在 Client ID 字段中，指定可帮助您将此客户端识别为 3scale 客户端的名称，如 oidc-issuer-for-3scale。

5. 将 Client Protocol 字段设置为 openid-connect。

6. 保存新客户端。

7. 在新客户端的设置中，设置以下内容：

   - Access Type 为 confidential。
   - Standard Flow Enabled 为 OFF。
   - Direct Access Grants Enabled 为 OFF。
   - Service Accounts Enabled 为 ON。此设置可让此客户端发布服务帐户。

a. 点 **Save**。

8. 为客户端设置服务帐户角色：

   a. 导航到客户端的 **Service Account Roles** 选项卡。

   b. 在 **Client Roles** 下拉列表中点 **realm-management**。

   c. 在 **Available Roles** 窗格中，选择 **manage-clients** 并通过点 **Add selected >>** 来分配角色。

9. 请注意客户端凭证：

   a. 记录客户端 ID(<client_id>)。

   b. 导航到客户端的 **Credentials** 选项卡，并记录 **Secret** 字段(<client_secret>)。

10. 要协助测试授权流，请在域中添加用户：

    a. 在窗口的左侧，展开 **Users**。

    b. 单击 **Add user**。

    c. 输入用户名，将 **Email Verified** 设置为 **ON**，然后点 **Save**。

    d. 在 **Credentials** 选项卡上，设置密码。在两个字段中输入密码，将 **Temporary** 开关设置为 **OFF**，以避免在下次登录时重置密码，然后点 **Set password**。

    e. 出现弹出窗口时，点 **Set password**。

13.4.3. 配置 3scale API 管理以使用 Red Hat Single Sign-On

通过在 3scale 管理门户中指定集成设置，配置 Red Hat 3scale API 管理以使用红帽单点登录。

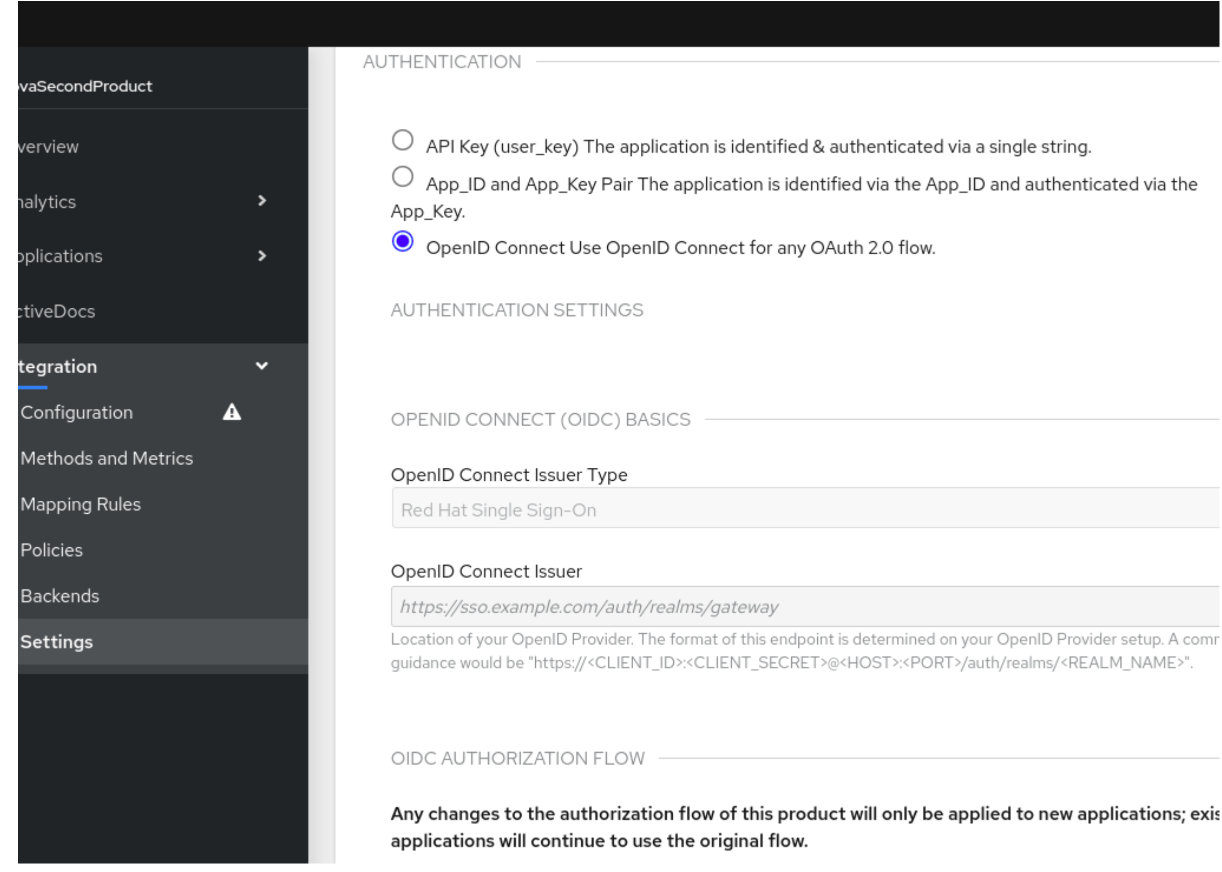
流程

1. 在 3scale Admin Portal 中，在顶层选择器中，点 Products，再选择您要为其启用 OpenID Connect 身份验证的 3scale API 产品。

2. 导航到 [Your_product_name] > Integration > Settings。

3. 在 Authentication 下，选择 OpenID Connect Use OpenID Connect for any OAuth 2.0 flow。

图 13.3. 显示 OPENID CONNECT(OIDC)BASICS 部分。

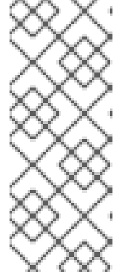


4. 在 OpenID Connect Issuer Type 字段中，确保该设置是 Red Hat Single Sign-On。

5. 在 OpenID Connect Issuer 字段中，输入配置的 OpenID Connect 身份提供程序的 URL。这个 URL 的格式如下：

https://<client_id>:<client_secret>@<rhsso_host>:<rhsso_port>/auth/realms/<realm_name>

6. 将占位符替换为前面记录的 Red Hat Single Sign-On 客户端凭证、Red Hat Single Sign-On 服务器的主机和端口，以及包含 Red Hat Single Sign-On 客户端的域名称。



注意

要在两个不同的产品间使用相同的凭证，您必须为每个产品或同一签发者配置唯一的 OIDC 签发者，但使用不同的域。您还可以将应用程序分成 2 个单独的产品，以使用具有相同域的同一 OIDC 供应商进行身份验证。有关指导，请参阅第 5 和 6 步。

7. 在 OIDC AUTHORIZATION FLOW 下，选择以下一个或多个：

- 授权代码流

  在 Red Hat Single Sign-On 中，这是标准流。

- 隐式流

- 服务帐户流

  也称为 客户端凭证流。

- 直接访问授予流

  也称为 Resource Owner Password Credentials。

图 13.4. 显示您选择的授权流：

OIDC AUTHORIZATION FLOW

Any changes to the authorization flow of this product will only be applied to new applications; existing applications will continue to use the original flow.

- [x] Authorization Code Flow
- [ ] Implicit Flow
- [ ] Service Accounts Flow
- [ ] Direct Access Grant Flow

JSON WEB TOKEN (JWT) CLAIM WITH CLIENTID

ClientID Token Claim Type

plain

Process the ClientID Token Claim value as a string or as a liquid template. When set to 'Liquid' you can define more complex rules. e.g. If 'some_claim' is an array you can select the first value this like {{ some_claim | first }}.

ClientID Token Claim

azp

The Token Claim that contains the clientID. Defaults to 'azp'.

这将配置 API 用户如何从 OpenID Connect 身份提供程序获取 JSON Web 令牌 (JWT)。当 3scale 将 Red Hat Single Sign-On 集成为 OpenID Connect 身份提供程序时，Zync 会创建启用了 授权代码 流的 Red Hat Single Sign-On 客户端。建议将此流程作为最适合大多数情况的最安全和最合适的流程。务必选择 OpenID Connect 身份提供程序支持的 OAuth 2.0 流。

8. *向下滚动并点 Update Product 以保存配置。*

9. *在左侧导航面板中点 Integration > Configuration。*

10. *向下滚动点击 Promote v.X 到 APIcast Staging。*

后续步骤

测试 Red Hat Single Sign-On 作为身份提供程序的集成。当一切工作时，返回到 Integration > Configuration 页面，再向下滚动以提升 APIcast staging 版本成为生产版本。

13.5. 将 3SCALE API 管理与第三方 OPENID CONNECT 身份提供程序集成

作为 API 供应商，您可以配置 3scale 和第三方 OpenID Connect 身份提供程序之间的 HTTP 集成。也就是说，您可以配置红帽单点登录以外的 OpenID Connect 身份提供程序。3scale 使用此集成来验证来自 API 用户的请求，并使用最新的 3scale 应用详情更新第三方身份提供程序。

将 3scale 与第三方 OpenID Connect 身份提供程序集成的大多数工作都涉及以下任务：

- 满足与 3scale Zync 相关的先决条件。
- 配置 OpenID Connect 身份提供程序，以授权来自 3scale 应用的请求。

先决条件

- 3scale Zync 已安装。
- 您选择的第三方 OpenID Connect 身份提供程序：
  - 遵循 3scale 提供的 OpenAPI 规格。
  - 允许用 <client_id> 和 <client_secret> 作为请求中的参数来注册客户端。3scale 始终是 3scale 和第三方 OpenID Connect 身份提供程序之间的客户端身份管理源。
  - 被配置为授权来自 3scale 应用程序的请求。
- 如果您的配置无法达到以前的先决条件，您必须实施基于 Zync abstract 适配器的自定义适配器。Zync 使用这个适配器与 OpenID Connect 身份提供程序交互。要创建此适配器，您可以修改 rest_adapter.rb， 它是 3scale Zync REST API 示例 的一部分。

  您可以根据最适合您的要求的方法在 zync-que pod 中包含 rest_adapter.rb 模块。例如，您可以通过卷挂载 configMap，也可以为 Zync 构建新镜像。

流程

1. 在 3scale Admin Portal 中，在顶层选择器中，点 Products，再选择您要为其启用 OpenID Connect 身份验证的 3scale API 产品。

2. *导航到 [Your_product_name] > Integration > Settings。*

3. *在 Authentication 下，选择 OpenID Connect Use OpenID Connect for any OAuth 2.0 flow。*

   这将显示 OPENID CONNECT(OIDC)BASICS 部分。

4. *在 OpenID Connect Issuer Type 字段中，确保设置是 REST API。*

5. *在 OpenID Connect Issuer 字段中输入 OpenID Connect 身份提供程序的 URL。这个 URL 的格式如下：*

   ```
   https://<client_id>:<client_secret>@<oidc_host>:<oidc_port>/<endpoint>
   ```

   例如，在 Zync rest_adapter.rb 示例中，URL 端点被硬编码为 {endpoint}/clients。您的端点可以是 {endpoint}/register 或其他对象。

6. *在 OIDC AUTHORIZATION FLOW 下，选择以下一个或多个：*

   - *授权代码流*
   - *隐式流*
   - *服务帐户流*
   - *直接访问授予流*

   这将配置 API 使用者应用如何从 OpenID Connect 身份提供程序接收 JSON Web 令牌 (JWT)。建议将 授权代码流 作为最安全且适合大多数情况。务必选择 OpenID Connect 身份提供程序支持的 OAuth 2.0 流。

7. 向下滚动并点 Update Product 以保存配置。

8. 在左侧导航面板中点 Integration > Configuration。

9. 向下滚动点击 Promote v.X 到 APIcast Staging。

后续步骤

测试与第三方身份提供程序的集成。当一切工作时，返回到 Integration > Configuration 页面，再向下滚动以提升 APIcast staging 版本成为生产版本。

13.6. 测试 3SCALE API 管理与 OPENID CONNECT 身份提供程序的集成

将 3scale 与 OpenID Connect 身份提供程序集成后，测试集成以确认：

- 当 API 用户订阅了 3scale 管理的 API 时，它会收到访问凭证。
- APIcast 可以验证来自 API 用户的请求。

先决条件

- 3scale 和 OpenID Connect 身份提供程序之间的集成适用于特定的 3scale API 产品。此集成使用授权代码流。
- 一个应用程序计划可供 API 用户在 Developer 门户中订阅。此应用程序计划提供对您配置了 OpenID Connect 身份验证的 3scale 管理的 API 的访问，即 3scale 产品。
- 将请求发送到上游 API 的应用。上游 API 是 3scale 产品的后端，API 消费者应用程序可在订阅后访问。或者，您可以使用 Postman 来发送请求。

流程

1. 在 Developer Portal 中，订阅一个应用程序计划。

这会在 Developer Portal 中创建应用程序。OpenID Connect 身份提供程序应返回客户端 ID 和客户端 secret，您可以在 Developer 门户的应用程序页面中看到。

2. *记下应用程序的客户端 ID 和客户端 secret。*

3. *验证 OpenID Connect 身份提供程序现在是否有具有相同客户端 ID 和客户端 secret 的客户端。例如，当 Red Hat Single Sign-On (RH-SSO) 是 OpenID Connect 身份提供程序时，您将在配置的 RH-SSO 域中看到一个新客户端。*

4. *在 Admin Portal 中的应用程序页面中，在 REDIRECT URL 字段中输入将 API 请求发送到上游 API 的应用 URL。*

5. *验证您的 OpenID Connect 身份提供程序具有正确的重定向 URL。*

6. *发现使用此端点接收 OpenID Connect 身份提供程序的身份验证请求的 URL：*

```
.well-known/openid-configuration
```

例如：

```
https://<rhsso_host>:<rhsso_port>/auth/realms/<realm_name>/.well-known/openid-configuration
```

7. *对于基础 URL，使用 OpenID Connect Issuer 字段中配置的 API 供应商的值。*

8. *编写使用上游 API 的应用程序的 API 使用者：*

   a. *使用您的 OpenConnect 身份提供程序启动授权流。此请求必须包含 3scale 应用程序的客户端 ID 和客户端 secret。在某些情况下，还需要最终用户身份。*

   b. *接收身份提供程序的响应，其中包含授权代码。*

9. *API 使用者的应用程序执行以下操作：*

a. 交换 JWT 的授权代码。

b. 身份验证时从 RH-SSO 接收 JWT。

c. 发送包含 JWT 的 API 请求到上游 API 后端。

如果 APIcast 接受请求中的 JWT，则您的应用程序将从 API 后端收到响应。

另外，为放置 API 消费者应用，使用 Postman 测试令牌流是否已正确实施。

13.7. 3SCALE API 管理与 OPENID CONNECT 身份提供程序集成的示例

本例演示了当您将 Red Hat 3scale API Management 与 Red Hat Single Sign-On 集成为 OpenID Connect 身份提供程序时的流程。这个示例有以下特征：

- 在管理门户中，API 供应商定义了一个 3scale API 产品，并将该产品配置为使用 Red Hat Single Sign-On 作为 OpenID Connect 身份提供程序。
- 这个产品的 OpenID Connect 配置包括：
  - 公共基本 URL：https://api.example.com
  - 私有基本 URL：https://internal-api.example.com
  - OpenID Connect Issuer: https://zync:41dbb98b-e4e9-4a89-84a3-91d1d19c4207@idp.example.com/auth/realms/myrealm
  - 选择 授权代码流 （即标准流）。
- 在 3scale Developer Portal 中，有以下特征：此应用程序是 API 消费者订阅访问 Developer 门户中特定应用程序计划提供的 3scale API 产品的结果。

- 客户端 ID ：myclientid
- 客户端 Secret ：myclientsecret
- 重定向 URL:https://myapp.example.com

- 在 Red Hat Single Sign-On 中，在 myrealm 域中，有具有以下特征的客户端：
  - 客户端 ID ：myclientid
  - 客户端 Secret ：myclientsecret
  - 重定向 URL:https://myapp.example.com
- myrealm realm 具有这个用户：
  - 用户名：myuser
  - 密码：mypassword
- myrealm 中的 3scale Zync 客户端具有正确的 服务帐户 角色。

流程如下：

1. 最终用户(API consumer)在以下端点上向身份验证服务器发送授权请求（本例中为 Red Hat Single Sign-On）：

```
https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/auth
```

在请求中，应用程序提供这些参数：

- *客户端 ID：myclientid*
- *重定向 URL: https://myapp.example.com*

2. *应用将 end-user 重定向到 Red Hat Single Sign-On 登录窗口。*

3. *最终用户使用这些凭证登录到 Red Hat Single Sign-On：*

    - *用户名：myuser*
    - *密码：mypassword*

4. *根据配置，以及是否首次在此特定应用验证最终用户时，可能会显示同意窗口。*

5. *Red Hat Single Sign-On 向最终用户发出授权代码。*

6. *API 消费者应用使用以下端点来发送请求，以交换 JWT 的授权代码：*

    ```
    https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/token
    ```

    请求包含授权代码和这些参数：

    - *客户端 ID：myclientid*
    - *客户端 secret：myclientsecret*
    - *重定向 URL: https://myapp.example.com.*

7. ***Red Hat Single Sign-On 返回带有 access_token 字段的 JSON Web Token (JWT)，如 eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A。***

8. ***API 消费者应用程序向 https://api.example.com 发送一个 API 请求，其中包含以下标头：***

    > *Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A.*

9. ***该应用应该会收到来自 https://internal-api.example.com 的成功响应。***