Red Hat 3scale API Management 2.14

操作 Red Hat 3scale API Management

如何自动部署、扩展您的环境并对问题进行故障排除

Last Updated: 2024-07-02

Red Hat 3scale API Management 2.14 操作 Red Hat 3scale API Management

如何自动部署、扩展您的环境并对问题进行故障排除

法律通告

摘要

本指南介绍了使用 Red Hat 3scale API Management 2.14 的部署操作。

目录

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。

第 1 章 3SCALE API 管理常规配置选项

作为 Red Hat 3scale API Management 管理员，在安装或帐户中带有常规配置选项，您可以基于它们进行设置调整。

1.1. 配置登录会话的有效长度

作为 Red Hat 3scale API Management 管理员，您可以为 Admin Portal 和 Developer Portal 配置有效的登录会话长度，从而使最大超时和不活跃的限制。

要实施有效的登录会话长度，您必须将 **USER_SESSION_TTL** 设置为 秒。例如，1,800 秒为 30 分钟。如果值为 **null** （即未设置或设置为空字符串），则会话默认长度为 2 周。

先决条件

- 具有管理员特权的 3scale 帐户。

流程

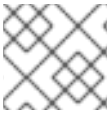
1. 更新 **system-app** secret 中的 **USER_SESSION_TTL** 值（以秒为单位）：

```
$ oc patch secret system-app -p '{"stringData": {"USER_SESSION_TTL": "'1800'"}}'
```

2. Rollout **system-app**:

```
$ oc rollout latest dc/system-app
```

第 2 章 3SCALE API 管理操作和扩展



注意

本文档不适用于笔记本电脑或类似最终用户设备的本地安装。

本节论述了红帽 3scale API 管理 2.14 安装的操作和扩展任务。

先决条件

- 在 受支持的 OpenShift 版本 上安装并最初配置 3scale On-premises 实例。

要执行 3scale 操作和扩展任务，请执行以下小节中所述的步骤：

- 重新部署 APIcast
- 在内部扩展 3scale API 管理
- 操作故障排除

2.1. 重新部署 APICAST

您可以通过 3scale 管理门户测试和提升系统更改。

先决条件

- 3scale 内部部署的实例。
- 您已选择了 APIcast 部署方法。

默认情况下，OpenShift 上的 APIcast 部署（嵌入式部署和其他 OpenShift 集群中）已配置为允许您通过 3scale 管理门户发布对您的暂存和生产网关的更改。

在 OpenShift 中重新部署 APIcast：

流程

1. 进行系统更改。
2. 在管理门户中，部署临时和测试。
3. 在管理门户中，将部署提升到生产环境。

默认情况下，APIcast 每 5 分钟检索并发布升级的更新一次。

如果您在 Docker 容器化环境或原生安装中使用 APIcast，请配置暂存和生产网关，并指示网关检索已发布更改的频率。在配置了 APIcast 网关后，您可以通过 3scale 管理门户重新部署 APIcast。

要在 Docker 容器化环境或原生安装上重新部署 APIcast：

流程

1. 配置 APIcast 网关并将其连接到 3scale 内部。

2. 进行系统更改。
3. 在管理门户中，部署临时和测试。
4. 在管理门户中，将部署提升到生产环境。

APIcast 以配置的频率检索并发布升级的更新。

2.2. 扩展内部 3SCALE API 管理

随着 APIcast 部署的增长，您可能需要增加可用的存储量。如何扩展存储取决于您用于持久性存储的文件系统类型。

如果使用网络文件系统(NFS)，使用以下命令扩展持久性卷(PV)：

```
$ oc edit pv <pv_name>
```

如果使用任何其他存储方法，则必须使用以下部分中列出的方法之一手动扩展持久性卷。

2.2.1. 方法 1：备份和恢复持久性卷

流程

1. 备份现有持久性卷中的数据。
2. 创建并附加一个目标持久性卷，按照您的新大小要求扩展。
3. 创建一个预绑定的持久性卷声明，使用 **volumeName** 字段指定新的 PVC (PersistentVolumeClaim) 和持久性卷名称的大小。
4. 将备份中的数据恢复到新创建的 PV。
5. 使用新 PV 的名称修改部署配置：

   ```
   $ oc edit dc/system-app
   ```

6. 验证您的新 PV 已配置且正常工作。
7. 删除您之前的 PVC 以释放其声明的资源。

2.2.2. 方法 2：备份和恢复 3scale API 管理

流程

1. 备份现有持久性卷中的数据。
2. 关闭 3scale pod。
3. 创建并附加一个目标持久性卷，按照您的新大小要求扩展。
4. 将备份中的数据恢复到新创建的 PV。
5. 创建预绑定持久性卷声明。指定：

a. 新 PVC 的大小

b. 使用 **volumeName** 字段的持久性卷名称。

6. 部署您的 *amp.yml*。
7. 验证您的新 PV 已配置且正常工作。
8. 删除您之前的 PVC 以释放其声明的资源。

2.2.3. 配置 3scale API 管理内部部署

为 3scale 扩展的关键部署配置有：

- APIcast 生产
- 后端监听程序
- 后端 worker

2.2.3.1. 通过 OCP 扩展

使用 APIManager CR 通过 OpenShift Container Platform (OCP)，您可以扩展或缩减部署配置。

要扩展特定的部署配置，请使用：

- 使用以下 APIManager CR 扩展 APIcast 生产部署配置：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      replicas: X
```

- 使用以下 APIManager CR 扩展部署配置的后端监听程序、后端 worker 和后端 cron 组件：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      replicas: X
    workerSpec:
      replicas: Y
    cronSpec:
      replicas: Z
```

- 将适当的环境变量设置为每个 pod 所需的进程数量。
  - 用于 **backend-listener** pod 的 **PUMA_WORKERS** ：

```
$ oc set env dc/backend-listener --overwrite PUMA_WORKERS=
<number_of_processes>
```

- 用于 **system-app** pod 的 **UNICORN_WORKERS** ：

```
$ oc set env dc/system-app --overwrite UNICORN_WORKERS=
<number_of_processes>
```

2.2.3.2. 垂直和横向硬件扩展

您可以通过添加资源来提高 OpenShift 上 3scale 部署的性能。您可以将更多计算节点作为 pod 添加到 OpenShift 集群，作为横向扩展，或者以垂直扩展形式将更多资源分配给现有计算节点。

横向扩展

您可以将更多计算节点作为容器集添加到 OpenShift 中。如果额外的计算节点与集群中的现有节点匹配，则不必重新配置任何环境变量。

垂直扩展

您可以为现有计算节点分配更多资源。如果分配更多资源，您必须在 pod 中添加额外的进程来提高性能。

注意

在 3scale 部署中避免使用具有不同规格和配置的计算节点。

2.2.3.3. 扩展路由器

随着流量的增加，确保您的红帽 OCP 路由器可以正确处理请求。如果您的路由器正在限制请求的吞吐量，您必须扩展路由器节点。

2.3. 操作故障排除

本节介绍如何配置 3scale 审计日志记录以在 OpenShift 上显示，以及如何访问 OpenShift 上的 3scale 日志和作业队列。

2.3.1. 在 OpenShift 中配置 3scale API 管理审计日志记录

这可让所有日志位于一个位置，供 Elasticsearch、Fluentd 和 Kibana(EFK)日志工具查询。这些工具提高了对 3scale 配置所做的更改的可见性，以及进行了这些更改的时间。例如，这包括对计费、应用计划、应用程序编程接口(API)配置等的变化。

先决条件

- 3scale 2.14 部署。

流程

配置审计日志记录到 **stdout**，以将所有应用日志转发到标准 OpenShift 容器集日志。

注意事项：

- 默认情况下，当 3scale 内部部署 3scale 时，将审计日志记录输出到 **stdout** 被禁用，您需要配置此功能才能完全正常工作。

- 对于托管的 3scale，不提供将审计日志记录输出到 **stdout** 的功能。

2.3.2. 启用审计日志记录

3scale 使用 **features.yml** 配置文件启用一些全局功能。要将审计日志记录启用到 **stdout**，您必须从 **ConfigMap** 挂载此文件，以替换默认的 文件。依赖于 **features.yml** 的 OpenShift pod 是 **system-app** 和 **system-sidekiq**。

先决条件

- 您必须具有 3scale 项目的管理员访问权限。

流程

1. 输入以下命令将审计日志记录启用到 **stdout** ：

```
$ oc patch configmap system -p '{"data": {"features.yml": "features: &default\n  logging:\n
audits_to_stdout: true\n\nproduction:\n  <<: *default\n"}}'
```

2. 导出以下环境变量：

```
$ export PATCH_SYSTEM_VOLUMES='{"spec":{"template":{"spec":{"volumes":[{"emptyDir":
{"medium":"Memory"},"name":"system-tmp"},{"configMap":{"items":
[{"key":"zync.yml","path":"zync.yml"},
{"key":"rolling_updates.yml","path":"rolling_updates.yml"},
{"key":"service_discovery.yml","path":"service_discovery.yml"},
{"key":"features.yml","path":"features.yml"}],"name":"system"},"name":"system-config"}]}}}}'
```

3. 输入以下命令将更新的部署配置应用到相关的 OpenShift pod:

```
$ oc patch dc system-app -p $PATCH_SYSTEM_VOLUMES

$ oc patch dc system-sidekiq -p $PATCH_SYSTEM_VOLUMES
```

2.3.3. 为 Red Hat OpenShift 配置日志记录

当您启用了审计日志记录以将 3scale 应用程序日志转发到 OpenShift 后，您可以使用日志记录工具监控 3scale 应用。

有关在 Red Hat OpenShift 中配置日志记录的详情，请参考以下内容：

- 了解 Red Hat OpenShift 的日志记录子系统

2.3.4. 访问日志

每个组件的部署配置都包含用于访问和异常的日志。如果您在部署时遇到问题，请检查这些日志以了解详细信息。

按照以下步骤访问 3scale 中的日志：

流程

1. 查找您要日志的 pod 的 ID：

```
$ oc get pods
```

2. 输入 **oc logs** 和所选 pod 的 ID：

```
$ oc logs <pod>
```

系统 pod 有两个容器，各自具有单独的日志。要访问容器的日志，请使用 **system-provider** 和 **system-developer** pod 指定 **--container** 参数：

```
$ oc logs <pod> --container=system-provider

$ oc logs <pod> --container=system-developer
```

2.3.5. 检查作业队列

作业队列包含从 **system-sidekiq** pod 发送的信息日志。使用这些日志来检查集群是否正在处理数据。您可以使用 OpenShift CLI 查询日志：

```
$ oc get jobs
```

```
$ oc logs <job>
```

2.3.6. 防止单调增长

为防止单例增长，3scale 默认调度 3scale 调度，自动清除以下表：

- **user_sessions**
  清理会每周触发一次，并删除超过两周的记录。
- **audits**
  清理会每天触发一次，并删除超过三个月的记录。
- **log_entries**
  每天清理一次，并删除超过 6 个月的记录。
- **event_store_events**
  清理会在一周内触发一次，并删除超过一周的记录。

除以上列出的表外，下表需要数据库管理员手动清除：

- **alerts**

表 2.1. SQL 清除命令

数据库类型	SQL 命令
MySQL	`DELETE FROM alerts WHERE timestamp < NOW() - INTERVAL 14 DAY;`

数据库类型	SQL 命令
PostgreSQL	`DELETE FROM alerts WHERE timestamp < NOW() - INTERVAL '14 day';`
Oracle	`DELETE FROM alerts WHERE timestamp <= TRUNC(SYSDATE) - 14;`

其他资源

- OCP 文档
- 自动缩放 pod
- 添加计算节点
- 优化路由

第 3 章 监控 3SCALE API 管理

Prometheus 是专为存储历史数据和监控大型可扩展系统而构建的容器原生软件。它在延长时间内收集数据，而不仅仅是针对当前运行的会话。Prometheus 中的警报规则由 Alertmanager 管理。

您可以使用 Prometheus 和 Alertmanager 来监控和存储 Red Hat 3scale API 管理数据，以便您可以使用图形工具（如 Grafana ）来视觉化并在数据上运行查询。

重要

- Prometheus 是一个开源系统监控工具包，Grafana 是一个开源仪表板工具包。红帽对 Prometheus 和 Grafana 的支持仅限于红帽产品文档中提供的配置建议。
- 3scale 操作器创建监控资源，但不会阻止修改这些资源。
- 您必须在同一命名空间中安装 3scale operator 和 Prometheus Operator，或使用集群范围的 Operator。

3scale 操作器允许您使用现有的 Prometheus 和 Grafana 操作器安装来监控 3scale 使用情况和资源。

先决条件

- 已安装 3scale operator。
- Prometheus Operator 已安装在集群中。Prometheus operator 是一个用于创建和管理 Prometheus 实例的操作器。它提供 3scale 监控所需的 **Prometheus** 自定义资源定义(CRD)。
  以下 Prometheus operator 和镜像版本使用 3scale 测试：
  - Prometheus operator **v0.37.0**
  - Prometheus 镜像: **quay.io/prometheus/prometheus:v2.16.0**
- Grafana Operator 在集群中安装。Grafana Operator 是一个用于创建和管理 Grafana 实例的 operator。它提供 3scale 监控所需的 **GrafanaDashboard** CRD。
  以下 Grafana operator 和镜像版本使用 3scale 测试：
  - Grafana operator **v3.9.0**
  - Grafana 镜像： **registry.hub.docker.com/grafana/grafana:7.1.1**

重要

如果您的集群在互联网上公开，请确保保护 Prometheus 和 Grafana 服务。

本节论述了如何启用对 3scale 实例的监控，以便您可以查看 Grafana 仪表板。

- 为 3scale API 管理启用监控
- 配置 Prometheus 来监控 3scale API 管理
- 配置 Grafana 来监控 3scale API 管理
- 查看 3scale API 管理的指标
- 3scale API 管理系统指标公开给 Prometheus

3.1. 为 3SCALE API 管理启用监控

要监控 3scale，您必须通过设置 APIManager 自定义资源(CR)来启用监控。

流程

1. 通过将 3scale 部署 YAML 的 **spec.monitoring.enabled** 参数设置为 **true**，将 3scale 配置为启用监控。例如：

   a. 创建名为 **3scale-monitoring.yml** 的 APIManager CR 来启用监控：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager1
spec:
  wildcardDomain: example.com
  monitoring:
    enabled: true
    enablePrometheusRules: false (1)
```

   (1) 您可以选择禁用 **PrometheusRules**，它被默认启用。

   b. 登录您的 OpenShift 集群。您必须以具有 *edit* 集群角色的用户身份在 3scale 的 OpenShift 项目中（如 **cluster-admin**）登录。

```
$ oc login
```

   c. 切换到 3scale 项目。

```
$ oc project <project_name>
```

   d. 部署自定义资源：

```
$ oc apply -f 3scale-monitoring.yml
```

其他资源

- 使用操作器在 OpenShift 上 3scale API 管理的部署配置选项
- 3scale PrometheusRules

3.2. 配置 PROMETHEUS 来监控 3SCALE API 管理

您必须使用 Prometheus 自定义资源(CR)部署和配置 **Prometheus**，以启用 3scale 的监控。

注意

确保正确设置了权限，如 Prometheus 文档 所述。

流程

1. 根据您要监控集群中的所有资源还是仅 3scale 资源，按如下所示部署 Prometheus CR：

   - 要监控集群中的所有资源，请将 **spec.podMonitorSelector** 属性设置为 **{}**，将 **spec.ruleSelector** 属性设置为 **{}**，并将 **spec.serviceMonitorSelector** 属性设置为 **{}**。例如，应用以下 CR：

     ```
     apiVersion: monitoring.coreos.com/v1
     kind: Prometheus
     metadata:
       name: prometheus
     spec:
       podMonitorSelector: {}
       ruleSelector: {}
       serviceMonitorSelector: {}
     ```

   - 如果您在同一 OpenShift 项目中部署了 3scale 和 Prometheus operator，并假设 **APP_LABEL** 的值被设置为默认的 **3scale-api-management**，请按照以下步骤监控 3scale 资源：

     a. 将 **spec.podMonitorSelector** 属性设置为：

     ```
     podMonitorSelector:
      matchExpressions:
      - key: app
          operator: In
          values:
          - 3scale-api-management
     ```

     b. 将 **spec.ruleSelector** 属性设置为：

     ```
       matchExpressions:
       - key: app
         operator: In
         values:
         - 3scale-api-management
     ```

     例如，应用以下 CR：

     ```
     apiVersion: monitoring.coreos.com/v1
     kind: Prometheus
     metadata:
       name: example
     spec:
      podMonitorSelector:
       matchExpressions:
       - key: app
          operator: In
          values:
          - 3scale-api-management
      ruleSelector:
        matchExpressions:
        - key: app
          operator: In
          values:
          - 3scale-api-management
     ```

- 如果您在不同的 OpenShift 项目中部署了 3scale 和 Prometheus operator，请按照以下步骤监控 3scale 资源：

  a. 为 OpenShift 项目添加标签，其中 3scale 使用 **MYLABELKEY=MYLABELVALUE**部署

  b. 使用 **podMonitorNamespaceSelector** 过滤器选择 3scale pod。例如，应用以下 CR：

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: example
spec:
 podMonitorSelector: {}
 ruleSelector: {}
 podMonitorNamespaceSelector:
   matchExpressions:
   - key: MYLABELKEY
     operator: In
     values:
     - MYLABELVALUE
```

2. 为确保仪表板和警报按预期工作，您必须执行以下操作之一来包含 Kubernetes 指标 （即 kube-state-metrics ）：

   - 将 Prometheus 实例与集群默认 Prometheus 实例相结合。
   - 配置您自己的提取任务，以从 kubelet、etcd 和其它数据获取指标。

其他资源

- Prometheus 文档

3.3. 配置 GRAFANA 来监控 3SCALE API 管理

您必须配置 Grafana 来启用 3scale 的监控。

流程

1. 通过覆盖 **app=3scale-api-management** 标签，确保将 **Grafana 服务配置为监控 GrafanaDashboards** 资源。例如，应用以下自定义资源(CR)：

```
apiVersion: integreatly.org/v1alpha1
kind: Grafana
metadata:
  name: grafana
spec:
  dashboardLabelSelector:
  - matchExpressions:
    - key: app
      operator: In
      values:
      - 3scale-api-management
```

由 3scale Operator 创建的 Grafana 仪表板被标记为如下：

```
    app: 3scale-api-management
    monitoring-key: middleware
```

2. 如果 Grafana Operator 安装在 3scale 以外的命名空间中，请将其配置为使用 **--namespaces** 或 **--scan-all** operator 标记来监控命名空间外的资源。如需有关 operator 标记的更多信息，请参阅 Grafana 文档。

3. 创建一个类型为 **prometheus** 的 **GrafanaDataSource** CR，以便在 Grafana 中提供 Prometheus 数据。例如：

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDataSource
metadata:
  name: prometheus
spec:
  name: middleware
  datasources:
    - name: Prometheus
      type: prometheus
      access: proxy
      url: http://prometheus-operated:9090
      isDefault: true
      version: 1
      editable: true
      jsonData:
        timeInterval: "5s"
```

   其中 **http://prometheus-operated:9090** 是 Prometheus 路由。

4. 确保正确设置了权限，如 Operator 标记 Grafana 文档中所述。

其他资源

- Grafana 文档

3.4. 查看 3SCALE API 管理的指标

配置 3scale、Prometheus 和 Grafana 后，您可以查看本节中描述的指标。

流程

1. 登录到 Grafana 控制台。
2. 检查您可以查看以下指标：
   - 安装 3scale 的 pod 和命名空间级别的 Kubernetes 资源
   - APIcast Staging
   - APIcast Production
   - 后端 worker
   - 后端监听程序

- System
- Zync

3.5. 3SCALE API 管理系统指标公开给 PROMETHEUS

您可以将以下端口配置为使用 3scale 系统 pod 和 Prometheus 端点来公开指标。

表 3.1. 3scale 系统端口

system-app	端口
system-developer	9394
system-master	9395
system-provider	9396

system-sidekiq	端口
system-sidekiq	9394

端点只能通过内部访问：

```
http://${service}:${port}/metrics
```

例如：

```
http://system-developer:9394/metrics
```

其他资源

- 向 Prometheus 公开 3scale API 管理 APIcast Metrics
- Prometheus 安全性
- Grafana 权限
- Grafana 安全性

第 4 章 使用 WEBHOOK 的 3SCALE API 管理自动化

Webhook 是一个有助于自动化的功能，也用于根据 3scale 中发生的事件集成其他系统。当在 3scale 系统中发生指定事件时，您的应用程序会收到 Webhook 消息通知。例如，通过配置 Webhook，您可以使用新帐户签名中的数据来填充 Developer Portal。

4.1. WEBHOOK 概述

Webhook 是一个由 **Webhook** 配置窗口中可用事件选择的事件触发的自定义 HTTP 回调。当发生这些事件时，3scale 系统会向 webhook 部分中指定的 URL 地址发出 HTTP 或 HTTPS 请求。使用 webhook，您可以配置监听器来调用某些所需的行为，如事件跟踪。

Webhook 的格式始终相同。它通过以下结构的 XML 文档发布到端点：

```
<?xml version="1.0" encoding="UTF-8"?>
<event>
  <type>application</type>
  <action>updated</action>
  <object>
    THE APPLICATION OBJECT AS WOULD BE RETURNED BY A GET ON THE ACCOUNT
MANAGEMENT
    API
  </object>
</event>
```

每个元素提供信息：

- **<type>**
  为您提供事件主题，如 *应用程序*、*帐户* 等等。

- **<action>**
  使用 *updated*, *created*, *deleted* 等值指定已进行了什么。

- **<object>**
  将 XML 对象本身视为与帐户管理 API 返回的格式相同。您可以使用我们的交互式 ActiveDocs 进行检查。

如果您需要提供 3scale 发布 webhook 的保证，公开 HTTPS Webhook URL，并在 3scale 中添加自定义参数到 webhook 声明中。例如： **https://your-webhook-endpoint?someSecretParameterName=someSecretParameterValue**。决定参数名称和值。然后，在 webhook 端点内检查是否存在这个参数值。

4.2. 配置 WEBHOOK

流程

1. 从 **Dashboard** 菜单中选择 **Account Settings**，然后进入到 **Integrate > Webhooks**。
2. 指定 webhook 的行为。有两个选项：
   - **Webhooks enabled** ：选择此复选框以启用或禁用 Webhook。
   - **Actions in the admin portal also trigger webhooks**：选择此复选框以在事件发生时触发 webhook。

请考虑以下几点：

- 当调用配置了触发事件的内部 3scale API 时，请使用访问令牌；而不是供应商密钥。
- 如果没有选择此复选框，则只有 Developer Portal 中的操作会触发 Webhook。

3. 指定触发所选事件通知的 URL 地址。
4. 选择将触发回调到指定 URL 地址的事件。

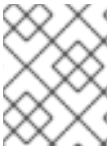
配置了设置后，点 **Update webhooks settings** 保存您的更改。

4.3. WEBHOOK 故障排除

如果您在侦听端点时遇到问题，可以恢复失效的发送。如果端点返回了 **200** 代码，3scale 会认为发生了一个 webhook。否则，它将重试 5 次，并有间隔 60 秒。在因中断或定期进行任何恢复后，您应该运行检查，并在适用清理队列时运行。您可以在 ActiveDocs 中找到有关以下方法的更多信息：

- Webhook 列表发送失败。
- Webhook 删除失败的发送。

第 5 章 3SCALE API 管理 TOOLBOX



注意

toolbox CLI 组件不再是活跃改进的主要重点。虽然它仍可用，但我们建议用户预测将来的有限改进。对配置和自动化需求的当前重点在于 3scale 应用能力操作员。

3scale toolbox 是一个 Ruby 客户端，可让您从命令行管理 3scale 产品。

3scale 文档中有有关安装 3scale toolbox、支持的 toolbox 命令、服务、计划、SSL 和 TLS 故障排除问题的信息。如需了解更多详细信息，请参阅以下部分之一：

- 安装 toolbox
- 支持的 toolbox 命令
- 导入服务
- 复制服务
- 仅复制服务设置
- OpenAPI 身份验证
- 导入 OpenAPI 定义
- 从 OpenAPI 定义导入 3scale API 管理后端
- 管理远程访问凭证
- 创建应用程序计划
- 创建指标
- 创建方法
- 创建服务
- 创建 ActiveDocs
- 列出代理配置
- 复制策略 registry
- 列出应用
- 导出产品
- 导入产品
- 导出和导入产品策略链
- 复制 API 后端
- SSL 和 TLS 故障排除

5.1. 安装 TOOLBOX

官方支持的安装 3scale toolbox 的方法是使用 3scale toolbox 容器镜像。

5.1.1. 安装 toolbox 容器镜像

本节介绍如何安装 toolbox 容器镜像。

先决条件

- 请参阅 Red Hat Ecosystem Catalog 中的 3scale API Management toolbox 镜像 。
- 您必须有一个红帽 registry 服务帐户。
- 本主题中的示例假定您已安装了 Podman。

流程

1. 登录到红帽生态系统目录：

```
$ podman login registry.redhat.io
Username: ${REGISTRY-SERVICE-ACCOUNT-USERNAME}
Password: ${REGISTRY-SERVICE-ACCOUNT-PASSWORD}
Login Succeeded!
```

2. 拉取 toolbox 容器镜像：

```
$ podman pull registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14
```

3. 验证安装：

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale help
```

其他资源

- 在 Red Hat Ecosystem Catalog 中获取镜像的说明
- 在 Kubernetes 上安装 3scale API Management toolbox 的说明
  注： 您必须在 OpenShift 上使用正确的镜像名称和 **oc** 命令而不是 **kubectl**。

5.2. 支持的 TOOLBOX 命令

使用 3scale toolbox 使用命令行工具(CLI)管理您的 API。

注意

update 命令已被删除，并由 **copy** 命令替代。

支持以下命令：

```
COMMANDS
  account          account super command
```

```
  activedocs          activedocs super command
  application         application super command
  application-plan    application-plan super command
  backend             backend super command
  copy                copy super command
  help                print help
  import              import super command
  method              method super command
  metric              metric super command
  policy-registry     policy-registry super command
  product             product super command
  proxy-config        proxy-config super command
  remote              remotes super command
  service             services super command

OPTIONS
  -c --config-file=<value>      3scale toolbox configuration file
                                (default: $HOME/.3scalerc.yaml)
  -h --help                     show help for this command
  -k --insecure                 Proceed and operate even for server
                                connections otherwise considered insecure
  -v --version                  Prints the version of this command
     --verbose                  Verbose mode
```

5.3. 导入服务

通过按照以下指定顺序指定以下字段，从 CSV 文件中导入服务。在 CSV 文件中包含这些标头：

```
service_name,endpoint_name,endpoint_http_method,endpoint_path,auth_mode,endpoint_system_name,type
```

您需要以下信息：

- 3scale admin 帐户：**{3SCALE_ADMIN}**
- 运行 3scale 实例的域：**{DOMAIN_NAME}**
  - 如果您使用的是托管 APICast，这是 3scale.net
- 您的帐户的访问密钥：{ **ACCESS_KEY}**
- 服务的 CSV 文件，例如：**example/import_example.csv**

运行以下命令导入服务：

Example

```
$ podman run -v $PWD/examples/import_example.csv:/tmp/import_example.csv
registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale import csv --
destination=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
file=/tmp/import_example.csv
```

本例使用 Podman 卷在容器中挂载资源文件。它假定该文件位于当前的 **$PWD** 文件夹中。

5.4. 复制服务

基于同一帐户或其他帐户中的现有服务创建新服务。复制服务时，也会复制相关的 ActiveDocs。

您需要以下信息：

- 要复制的服务 id： **{SERVICE_ID}**
- 3scale admin 帐户： **{3SCALE_ADMIN}**
- 运行 3scale 实例的域： **{DOMAIN_NAME}**
  - 如果您使用的是托管 APICast，这是 3scale.net
- 您的帐户的访问密钥：{ **ACCESS_KEY}**
- 如果要复制到其他帐户，目标帐户的访问密钥： **{DEST_KEY}**
- 新服务的名称： **{NEW_NAME}**

Example

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale copy service
{SERVICE_ID} --source=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
destination=https://{DEST_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
target_system_name={NEW_NAME}
```

注意

如果要复制的服务具有自定义策略，请确保它们对应的自定义策略定义已存在于要复制该服务的目标中。要了解更多有关复制自定义策略定义的信息，请查看复制策略 registry

5.5. 仅复制服务设置

您可以批量复制服务及代理设置、指标、方法、应用计划、应用计划限制，以及将规则从服务映射到其他现有服务。

您需要以下信息：

- 要复制的服务 id： **{SERVICE_ID}**
- 目标的服务 ID： **{DEST_ID}**
- 3scale admin 帐户： **{3SCALE_ADMIN}**
- 运行 3scale 实例的域： **{DOMAIN_NAME}**
  - 如果您使用的是托管 APICast，这是 3scale.net
- 您的帐户的访问密钥：{ **ACCESS_KEY}**
- 目标帐户的访问密钥： **{DEST_KEY}**

另外，您可以使用可选标志：

- 用于在复制前删除现有目标服务映射规则的 **-f** 标志。

- 用于仅将规则复制到目标服务的 **-r** 标志。

注意

update 命令已被删除，并由 **copy** 命令替代。

以下示例命令从一个服务批量复制到另一个现有服务：

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale copy [opts] service --
source=https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} --
destination=https://{DEST_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} {SERVICE_ID}
{DEST_ID}
```

5.6. OPENAPI 身份验证

通过使用 3scale toolbox 实现 OpenAPI 身份验证，您可以确保只有授权的用户有权访问您的 API，保护敏感数据，并有效管理 API 用量。这种方法将强化您的 API 基础架构，并促进开发人员和消费者之间的信任。

注意

只支持一个顶级安全要求；不支持操作级别的安全要求。

支持的安全方案：带有任何流类型的 **apiKey** 和 **oauth2**。

对于 apiKey 安全方案类型：

- 凭证位置从安全方案对象的 OpenAPI in 字段中读取。
- auth user 键是从安全方案对象的 OpenAPI name 字段读取的。

带有 apiKey 安全要求的 OpenAPI 3.0.2 部分示例：

```
openapi: "3.0.2"
security:
  - petstore_api_key: []
components:
  securitySchemes:
    petstore_api_key:
      type: apiKey
      name: api_key
      in: header
      ...
```

对于 **oauth2** 安全方案类型：

- 凭据位置硬编码为 **标头**。
- OpenID Connect Issuer Type 默认为 **rest**。您可以使用 **--oidc-issuer-type=<value&gt**; 命令选项覆盖它。
- OpenID Connect Issuer 没有从 OpenAPI 读取。由于 3scale 要求签发者 URL 必须包含客户端 secret，因此必须使用这个 **--oidc-issuer-endpoint=<value>** 命令选项设置这个问题。

- OIDC AUTHORIZATION FLOW 从安全方案对象的 flows 字段读取。

带有 oauth2 安全要求的 OpenAPI 3.0.2 部分示例：

```
openapi: "3.0.2"
security:
  - petstore_oauth:
    - write:pets
    - read:pets
components:
  securitySchemes:
    petstore_oauth:
      type: oauth2
      flows:
        clientCredentials:
          tokenUrl: http://example.org/api/oauth/dialog
          scopes:
            write:pets: modify pets in your account
            read:pets: read your pets
            ...
```

当 OpenAPI 没有指定任何安全要求时：

- 产品被视为 *Open API*。
- 添加 **default_credentials** 3scale 策略。**注意：** 这也称为 **anonymous_policy**。
- 您需要命令 **--default-credentials-userkey**。**注：** 如果没有提供，命令会失败。

其他资源

- 安全方案对象

5.7. 导入 OPENAPI 定义

要创建新服务或更新现有服务，您可以从本地文件或 URL 导入 OpenAPI 定义。导入的默认服务名称由 OpenAPI 定义中的 **info.title** 指定。但是，您可以使用 **--target_system_name=<NEW NAME>** 覆盖此服务名称。这将更新服务名称（如果已存在），或者创建新的服务名称（如果不存在）。

import openapi 命令的格式如下：

```
$ 3scale import openapi [opts] -d <destination> <specification>
```

OpenAPI **<specification>** 可以是以下之一：

- **/path/to/your/definition/file.[json|yaml|yml]**
- **http[s]://domain/resource/path.[json|yaml|yml]**

Example

```
$ podman run --rm -v ${ABSOLUTE_RESOURCE_PATH}:/tmp/toolbox:Z registry.redhat.io/3scale-amp2/toolbox-rhel8 3scale import openapi -d https://${AUTH}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME} /tmp/toolbox/my-test-api.json
```

命令选项

import openapi 命令选项包括：

-d --destination=<value>

3scale 目标实例格式： **http[s]://<authentication>@3scale_domain**。

-t --target_system_name=<value>

3scale 目标系统名称。

--backend-api-secret-token=<value>

API 网关向后端 API 发送的自定义机密令牌。

--backend-api-host-header=<value>

API 网关向后端 API 发送的自定义主机标头。

如需了解更多选项，请参阅 **3scale import openapi --help** 命令。

OpenAPI 导入规则

支持的安全方案是 **apiKey** 和 **oauth2**，其中包含任何 OAuth 流类型。

OpenAPI 规格必须是以下之一：

- 可用路径中的文件名.
- toolbox 可以从中下载内容的 URL。支持的方案是 **http** 和 **https**。
- 从 **stdin** 标准输入流读取.这通过设置 **-** 值来控制。

导入 OpenAPI 定义时应用以下附加规则：

- 定义验证为 OpenAPI 2.0 或 OpenAPI 3.0。
- 导入 OpenAPI 定义中的所有映射规则。您可以在 **API** > **Integration** 中查看它们。
- 3scale 产品中的所有映射规则都被替换。
- 仅修改 OpenAPI 定义中包含的方法。
- 仅存在于 OpenAPI 定义中的所有方法都附加到 **Hits** 指标。
- 要替换方法，方法名称必须与 OpenAPI 定义 **operation.operationId** 中定义的方法相同，使用精确的模式匹配。

注意

toolbox 将添加一个 **default_credentials** 策略，如果还没有在策略链中，它也称为 **anonymous_policy**。**default_credentials** 策略将配置在可选参数 **--default-credentials-userkey** 中提供的 *userkey*。

OpenAPI 3.0 提供了使用其安全方案和安全要求功能为您的 API 指定安全性的方法。如需更多信息，请参阅官方 Swagger 身份验证和授权 文档。

OpenAPI 3.0 限制

导入 OpenAPI 3.0 定义时会应用以下限制：

- 只有 **servers** 列表中的第一个 **server.url** 元素解析为专用 URL。**server.url** 元素的 **path** 组件将用作 OpenAPI 的 **basePath** 属性。
- toolbox 不会解析操作对象中路径项和服务器中的服务器。
- 不支持安全方案对象中的多个流。

5.8. 从 OPENAPI 定义导入 3SCALE API 管理后端

您可以使用 **import** 命令导入 OpenAPI 定义并创建 3scale 后端 API。命令行选项 **--backend** 可启用此功能。3scale 使用 OpenAPI 定义来创建和存储后端及其专用基础 URL，以及映射规则和方法。

先决条件

- 具有 3scale 2.14 On-Premises 实例的管理员特权的用户帐户。
- 定义 API 的 OAS 文档。

流程

- 使用以下格式运行 **import** 命令来创建后端：

```
$ 3scale import openapi -d <remote> --backend <OAS>
```

- 将 **<remote>** 替换为用于创建后端的 3scale 实例的 URL。使用此格式：**http[s]://<authentication>@3scale_domain**
- 将 **<OAS>** 替换为 **/path/to/your/oasdoc.yaml**。

表 5.1. 其他 OpenAPI 定义选项

选项	描述
-o --output=<value>	输出格式。可以是 JSON 或 YAML。
--override-private-base-url=<value>	3scale 从 OpenAPI 定义的 **servers[0].url** 字段读取后端的专用端点。要覆盖该字段中的设置，请指定这个选项，将 **<value>** 替换为您选择的私有基本 URL。当 OpenAPI 定义没有在 **servers[0].url** 字段中指定值，且您在 **import** 命令中没有指定这个选项时，执行会失败。
--prefix-matching	在从 OpenAPI 操作派生的映射规则时，使用前缀匹配而不是严格匹配。
--skip-openapi-validation	跳过 OpenAPI 模式验证。
-t --target_system_name=<value>	目标系统名称是您租户中的唯一键。系统名称可以从 OpenAPI 定义中推断出来，但您可以使用这个参数覆盖您自己的名称。

5.9. 管理远程访问凭证

为了便于使用远程 3scale 实例，您可以使用 3scale toolbox 定义远程 URL 地址和验证详情，以便在配置文件中访问这些远程实例。然后，您可以在任意 toolbox 命令中使用一个短名称来引用这些远程数据。

配置文件的默认位置为 **$HOME/.3scalerc.yaml**。但是，您可以使用 **THREESCALE_CLI_CONFIG** 环境变量或 **--config-file <config_file>** toolbox 选项指定另一个位置。

在添加远程访问凭证时，您可以指定 **access_token** 或 **provider_key** ：

- **http[s]://<access_token>@<3scale-instance-domain>**
- **http[s]://<provider_key>@<3scale-instance-domain>**

5.9.1. 添加远程访问凭证

以下示例命令在 **<url>** 添加一个远程 3scale 实例，带有短的 **<name>** ：

```
$ 3scale remote add [--config-file <config_file>] <name> <url>
```

Example

```
$ podman run --name toolbox-container registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14
3scale remote add instance_a https://123456789@example_a.net

$ podman commit toolbox-container toolbox
```

本例创建远程实例并提交容器以创建新镜像。然后，您可以使用包含的远程信息运行新镜像。例如，以下命令使用新镜像显示新添加的远程：

```
$ podman run toolbox 3scale remote list
instance_a https://example_a.net 123456789
```

然后，其他 toolbox 命令可以使用新创建的镜像来访问添加的远程系统。这个示例使用名为 **toolbox** 的镜像而不是 **registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14** 的镜像。

警告

在容器中存储 toolbox 的 secret 存在潜在的安全风险，例如，将容器与 secret 一起分发到其他用户或使用容器进行自动化时。在 OpenShift 中使用 Podman 或机密中的安全卷。

其他资源

有关使用 Podman 的更多详细信息，请参阅：

- 在 Red Hat Enterprise Linux 8 中构建、运行和管理 Linux 容器

5.9.2. 列出远程访问凭证

以下示例命令演示了如何列出远程访问凭证：

```
$ 3scale remote list [--config-file <config_file>]
```

此命令以以下格式显示添加的远程 3scale 实例列表：**<name> <URL> <authentication-key>**:

Example

```
$ podman run <toolbox_image_with_remotes_added> 3scale remote list
instance_a https://example_a.net 123456789
instance_b https://example_b.net 987654321
```

5.9.3. 删除远程访问凭证

以下示例命令演示了如何删除远程访问凭证：

```
$ 3scale remote remove [--config-file <config_file>] <name>
```

这个命令使用简短 **<name>** 删除远程 3scale 实例：

Example

```
$ podman run <toolbox_image_with_remote_added> 3scale remote remove instance_a
```

5.9.4. 重命名远程访问凭证

以下示例命令演示了如何重命名远程访问凭证：

```
$ 3scale remote rename [--config-file <config_file>] <old_name> <new_name>
```

这个命令使用简短的 **<old_name>** 将远程 3scale 实例重命名为 **<new_name>** ：

Example

```
$ podman run <toolbox_image_with_remote_added> 3scale remote rename instance_a instance_b
```

5.10. 创建应用程序计划

使用 3scale toolbox 在 Developer Portal 中创建、更新、列出、删除、显示或导出/导入应用程序计划。

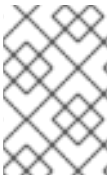
5.10.1. 创建新应用程序计划

使用以下步骤创建新应用程序计划：

- 您必须提供应用计划名称。
- 要覆盖 **system-name**，请使用可选参数。
- 如果存在具有相同名称的应用计划，您将看到错误消息。
- 使用 **--default** 标志，将应用计划设置为 **--default**。
- 使用 **--publish** 标志创建一个发布的应用计划。

- 默认情况下，它将被 **隐藏**。

- 使用 **--disabled** 标志创建 **禁用** 的应用程序计划。
  - 默认情况下将 **启用** 它。



注意

- **服务** 位置参数是服务引用，可以是服务 **ID** 或 service **system_name**。
  - toolbox 使用其中任一个。

以下命令创建新应用程序计划：

```
$ 3scale application-plan create [opts] <remote> <service> <plan-name>
```

在创建应用程序计划时使用以下选项：

```
Options
      --approval-required=<value>    The application requires approval:
                                     true or false
      --cost-per-month=<value>       Cost per month
      --default                      Make the default application plan
      --disabled                     Disable all methods and metrics in
                                     the application plan
  -o --output=<value>                Output format on stdout:
                                     one of json|yaml
  -p --published                     Publish the application plan
      --setup-fee=<value>            Set-up fee
  -t --system-name=<value>           Set application plan system name
      --trial-period-days=<value>    The trial period in days

Options for application-plan
  -c --config-file=<value>           3scale toolbox configuration file:
                                     defaults to $HOME/.3scalerc.yaml
  -h --help                          Print help for this command
  -k --insecure                      Proceed and operate even for server
                                     connections otherwise considered
                                     insecure
  -v --version                       Print the version of this command
      --verbose                      Verbose mode
```

5.10.2. 创建或更新应用程序计划

如果使用以下步骤创建新应用程序计划（如果不存在），或更新现有应用程序计划：

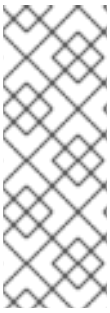
- 使用 **--default** 标志更新 **默认** 应用计划。

- 使用 **--publish** 标志，更新 发**布的** 应用计划。

- 使用 **--hide** 标志更新 **隐藏** 的应用计划。

- 使用 **--disabled** 标志更新 **禁用** 的应用程序计划。

- 使用 **--enabled** 标志，更新 **启用** 的应用计划。



注意

- **服务** 位置参数是服务引用，可以是服务 **ID** 或 service **system_name**。
  - toolbox 使用其中任一个。
- **plan** 位置参数是计划引用，可以是计划 **id** 或 plan **system_name**。
  - toolbox 使用其中任一个。

以下命令更新应用程序计划：

```
$ 3scale application-plan create [opts] <remote> <service> <plan>
```

在更新应用程序计划时使用以下选项：

```
Options
      --approval-required=<value>    The application requires approval:
                                 true or false
      --cost-per-month=<value>       Cost per month
      --default                    Make the default application plan
      --disabled                   Disable all methods and metrics in
                                 the application plan
      --enabled                    Enable the application plan
      --hide                     Hide the application plan
   -n --name=<value>               Set the plan name
   -o --output=<value>             Output format on stdout:
                                 one of json|yaml
   -p --publish                   Publish the application plan
      --setup-fee=<value>          Set-up fee
      --trial-period-days=<value>    The trial period in days

Options for application-plan
   -c --config-file=<value>        3scale toolbox configuration file:
                                 defaults to $HOME/.3scalerc.yaml
   -h --help                      Print help for this command
   -k --insecure                   Proceed and operate even for server
                                 connections otherwise considered
                                 insecure
   -v --version                   Print the version of this command
      --verbose                   Verbose mode
```

5.10.3. 列出应用计划

以下命令列出应用程序计划：

```
$ 3scale application-plan list [opts] <remote> <service>
```

在列出应用程序计划时使用以下选项：

```
Options
   -o --output=<value>          Output format on stdout:
                              one of json|yaml
```

```
Options for application-plan
   -c --config-file=<value>      3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                 Proceed and operate even for server
                              connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.10.4. 显示应用程序计划

以下命令显示应用程序计划：

```
$ 3scale application-plan show [opts] <remote> <service> <plan>
```

在显示应用程序计划时使用以下选项：

```
Options
   -o --output=<value>           Output format on stdout:
                              one of json|yaml

Options for application-plan
   -c --config-file=<value>      3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                 Proceed and operate even for server
                              connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.10.5. 删除应用程序计划

以下命令删除应用程序计划：

```
$ 3scale application-plan delete [opts] <remote> <service> <plan>
```

在删除应用程序计划时使用以下选项：

```
Options for application-plan
   -c --config-file=<value>      3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                 Proceed and operate even for server
                              connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.10.6. 导出/导入应用程序计划

您可以将单个应用程序计划导出或导入到 **yaml** 内容。

注意以下几点：

- 包括了应用程序计划中定义的限制。
- 包括了应用程序计划中定义的定价规则。
- 包括根据限制和价格规则引用的指标/方法。
- 应用程序计划中定义的功能包括：
- 服务可以被 **id** 或 **system_name** 引用。
- 应用程序计划可以被 **id** 或 **system_name** 来引用。

5.10.6.1. 将应用计划导出到文件

以下命令导出应用程序计划：

```
$ 3scale application-plan export [opts] <remote> <service_system_name> <plan_system_name>
```

Example

```
$ podman run -u root -v $PWD:/tmp registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale
application-plan export --file=/tmp/plan.yaml remote_name service_name plan_name
```

本例使用 Podman 卷将输出结果中的导出文件挂载到当前的 **$PWD** 文件夹。

注意

特定于 export 命令：

- 对远程服务和应用计划进行只读操作.
- 命令输出可以是 **stdout** 或文件。
  - 如果没有由 **-f** 选项指定，则默认情况下，**yaml** 内容将写入 **stdout**。

在导出应用程序计划时使用以下选项：

```
Options
   -f --file=<value>           Write to file instead of stdout

Options for application-plan
   -c --config-file=<value>      3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                  Proceed and operate even for server
                              connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.10.6.2. 从文件导入应用计划

以下命令导入应用程序计划：

```
$ 3scale application-plan import [opts] <remote> <service_system_name>
```

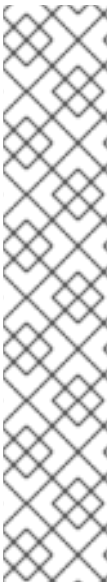
Example

```
$ podman run -v $PWD/plan.yaml:/tmp/plan.yaml registry.redhat.io/3scale-amp2/toolbox-
rhel8:3scale2.14 3scale application-plan import --file=/tmp/plan.yaml remote_name service_name
```

本例使用 Podman 卷从当前的 **$PWD** 文件夹挂载容器中导入的文件。

5.10.6.3. 从 URL 导入应用计划

```
$ 3scale application-plan import -f http[s]://domain/resource/path.yaml remote_name service_name
```



注意

特定于导入命令：

- 命令输入内容可以 **stdin**、文件或 URL 格式。
  - 如果没有由 **-f** 选项指定，默认情况下，**yaml** 内容将从 **stdin** 中读取。
- 如果无法在远程服务中找到应用计划，它将被创建。
- 可选的 param **-p**，**--plan**，用于覆盖远程目标应用程序计划 **id** 或 **system_name**。
  - 如果没有通过 **-p** 选项指定，默认情况下，应用程序计划将由来自 **yaml** 内容的计划属性 **system_name from** 引用。
- 将创建在远程服务中没有的 yaml 内容中的任何指标或方法。

导入应用程序计划时使用以下选项：

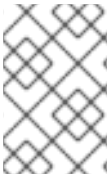
```
Options
   -f --file=<value>              Read from file or URL instead of
                              stdin
   -p --plan=<value>              Override application plan reference

Options for application-plan
   -c --config-file=<value>       3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                  Print help for this command
   -k --insecure              Proceed and operate even for server
                              connections otherwise considered
                              insecure
   -v --version               Print the version of this command
     --verbose                Verbose mode
```

5.11. 创建指标

使用 3scale toolbox 在 Developer Portal 中创建、更新、列出和删除指标。

使用以下步骤创建指标：

- 您必须提供指标名称。
- 要覆盖 **system-name**，请使用可选参数。
- 如果存在相同名称的指标，您将看到错误消息。
- 使用 **--disabled** 标志创建**禁用的**指标。
  - 默认情况下将 **启用** 它。



注意

- **服务** 位置参数是服务引用，可以是服务 **ID** 或 service **system_name**。
  - toolbox 使用其中任一个。

以下命令创建指标：

```
$ 3scale metric create [opts] <remote> <service> <metric-name>
```

在创建指标时请使用以下选项：

```
Options
      --description=<value>      Set a metric description
      --disabled                 Disable this metric in all application
                                 plans
   -o --output=<value>           Output format on stdout:
                                 one of json|yaml
   -t --system-name=<value>      Set the application plan system name
      --unit=<value>             Metric unit: default hit

Options for metric
   -c --config-file=<value>      3scale toolbox configuration file:
                                 defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                 Proceed and operate even for server
                                 connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.11.1. 创建或更新指标

如果新的指标不存在，请使用以下步骤创建新指标，或更新现有指标：

- 如果存在相同名称的指标，您将看到错误消息。
- 使用 **--disabled** 标志更新**禁用的**指标。
- 使用 **--enabled** 标志更新为 **启用** 的指标。

注意

- **服务** 位置参数是服务引用，可以是服务 **ID** 或 service **system_name**。
  - toolbox 使用其中任一个。
- **指标** 位置参数是指标引用，可以是指标 **ID** 或指标 **system_name**。
  - toolbox 使用其中任一个。

以下命令更新了指标：

```
$ 3scale metric apply [opts] <remote> <service> <metric>
```

在更新指标时请使用以下选项：

```
Options
      --description=<value>      Set a metric description
      --disabled                 Disable this metric in all application
                                 plans
      --enabled                  Enable this metric in all application
                                 plans
   -n --name=<value>             This will set the metric name
      --unit=<value>             Metric unit: default hit
   -o --output=<value>           Output format on stdout:
                                 one of json|yaml

Options for metric
   -c --config-file=<value>      3scale toolbox configuration file:
                                 defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                 Proceed and operate even for server
                                 connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.11.2. 列出指标

以下命令列出了指标：

```
$ 3scale metric list [opts] <remote> <service>
```

在列出指标时请使用以下选项：

```
Options
   -o --output=<value>           Output format on stdout:
                                 one of json|yaml

Options for metric
   -c --config-file=<value>      3scale toolbox configuration file:
                                 defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                 Proceed and operate even for server
```

```
                        connections otherwise considered insecure
  -v --version              Print the version of this command
    --verbose               Verbose mode
```

5.11.3. 删除指标

以下命令删除指标：

```
$ 3scale metric delete [opts] <remote> <service> <metric>
```

在删除指标时请使用以下选项：

```
Options for metric
  -c --config-file=<value>      3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
  -h --help                 Print help for this command
  -k --insecure              Proceed and operate even for server
                        connections otherwise considered insecure
  -v --version              Print the version of this command
    --verbose               Verbose mode
```

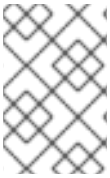
5.12. 创建方法

使用 3scale toolbox 在 Developer Portal 中创建、应用、列出和删除方法。

5.12.1. 创建方法

使用以下步骤创建方法：

- 您必须提供方法名称。
- 要覆盖 **system-name**，请使用可选参数。
- 如果存在具有相同名称的方法，您将看到错误消息。
- 通过 **--disabled** 标志创建 **禁用** 的方法。
  - 默认情况下将 **启用** 它。



注意

- **服务** 位置参数是服务引用，可以是服务 **ID** 或 service **system_name**。
  - toolbox 使用其中任一个。

以下命令创建一个方法：

```
$ 3scale method create [opts] <remote> <service> <method-name>
```

在创建方法时使用以下选项：

```
Options
      --description=<value>      Set a method description
```

```
    --disabled              Disable this method in all
                          application plans
  -o --output=<value>       Output format on stdout:
                          one of json|yaml
  -t --system-name=<value>  Set the method system name

Options for method
  -c --config-file=<value>  3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
  -h --help                 Print help for this command
  -k --insecure             Proceed and operate even for server
                          connections otherwise considered insecure
  -v --version              Print the version of this command
    --verbose               Verbose mode
```

5.12.2. 创建或更新方法

如果方法不存在，请使用以下步骤创建新方法，或更新现有的方法：

- 如果存在具有相同名称的方法，命令将返回错误消息。
- 使用 **--disabled 标志** 更新为 **禁用** 的方法。
- 使用 **--enabled 标志** 更新为 **启用** 的方法。

注意

- **服务** 位置参数是服务引用，可以是服务 **ID** 或 service **system_name**。
  - toolbox 使用其中任一个。
- **method** 位置参数是方法引用，可以是方法 **id** 或方法 **system_name**。
  - toolbox 使用其中任一个。

以下命令更新方法：

```
$ 3scale method apply [opts] <remote> <service> <method>
```

在更新方法时使用以下选项：

```
Options
    --description=<value>   Set a method description
    --disabled              Disable this method in all
                          application plans
    --enabled               Enable this method in all
                          application plans
  -n --name=<value>         Set the method name
  -o --output=<value>       Output format on stdout:
                          one of json|yaml

Options for method
  -c --config-file=<value>  3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
  -h --help                 Print help for this command
```

```
  -k --insecure               Proceed and operate even for server
                            connections otherwise considered insecure
  -v --version              Print the version of this command
    --verbose               Verbose mode
```

5.12.3. 列出方法

以下命令列出方法：

```
$ 3scale method list [opts] <remote> <service>
```

在列出方法时使用以下选项：

```
Options
  -o --output=<value>         Output format on stdout:
                            one of json|yaml

Options for method
  -c --config-file=<value>    3scale toolbox configuration file:
                            defaults to $HOME/.3scalerc.yaml
  -h --help                 Print help for this command
  -k --insecure               Proceed and operate even for server
                            connections otherwise considered insecure
  -v --version              Print the version of this command
    --verbose               Verbose mode
```

5.12.4. 删除方法

以下命令删除方法：

```
$ 3scale method delete [opts] <remote> <service> <metric>
```

在删除方法时使用以下选项：

```
Options for method
  -c --config-file=<value>    3scale toolbox configuration file:
                            defaults to $HOME/.3scalerc.yaml
  -h --help                 Print help for this command
  -k --insecure               Proceed and operate even for server
                            connections otherwise considered insecure
  -v --version              Print the version of this command
    --verbose               Verbose mode
```

5.13. 创建服务

使用 3scale toolbox 在 Developer Portal 中创建、应用、列出、显示或删除服务。

5.13.1. 创建新服务

以下命令创建新服务：

```
$ 3scale service create [options] <remote> <service-name>
```

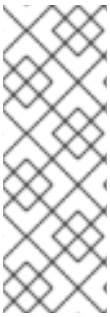
-

在创建服务时使用以下选项：

```
Options
   -a --authentication-mode=<value>     Specify authentication mode of
                                 the service:
                                  - '1' for API key
                                  - '2' for App Id/App Key
                                  - 'oauth' for OAuth mode
                                  - 'oidc' for OpenID Connect
   -d --deployment-mode=<value>         Specify the deployment mode of
                                 the service
      --description=<value>             Specify the description of the
                                 service
   -o --output=<value>                  Output format on stdout:
                                 one of json|yaml
   -s --system-name=<value>             Specify the system-name of the
                                 service
      --support-email=<value>           Specify the support email of the
                                 service

Options for service
   -c --config-file=<value>             3scale toolbox configuration file:
                                 defaults to $HOME/.3scalerc.yaml
   -h --help                            Print help for this command
   -k --insecure                         Proceed and operate even for
                                 server connections otherwise
                                 considered insecure
   -v --version                         Print the version of this command
      --verbose                         Verbose mode
```

5.13.2. 创建或更新服务

如果新服务不存在，请使用以下内容创建新服务，或更新现有服务：



注意

- **service-id_or_system-name** 位置参数是服务引用。
  - 它可以是服务 **ID**，也可以是 service **system_name**。
  - toolbox 将自动找出这一点。
- 此命令是 **幂等的**。

以下命令更新服务：

```
$ 3scale service apply <remote> <service-id_or_system-name>
```

在更新服务时使用以下选项：

```
Options
   -a --authentication-mode=<value>     Specify authentication mode of
                                 the service:
```

```
                                - '1' for API key
                                - '2' for App Id/App Key
                                - 'oauth' for OAuth mode
                                - 'oidc' for OpenID Connect
  -d --deployment-mode=<value>        Specify the deployment mode of
                              the service
     --description=<value>            Specify the description of the
                              service
  -n --name=<value>                   Specify the name of the metric
     --support-email=<value>          Specify the support email of the
                              service
  -o --output=<value>                 Output format on stdout:
                              one of json|yaml

Options for services
  -c --config-file=<value>            3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
  -h --help                           Print help for this command
  -k --insecure                       Proceed and operate even for
                              server connections otherwise
                              considered insecure
  -v --version                        Print the version of this command
     --verbose                        Verbose mode
```

5.13.3. 列出服务

以下命令列出服务：

```
$ 3scale service list <remote>
```

在列出服务时使用以下选项：

```
Options
  -o --output=<value>          Output format on stdout:
                          one of json|yaml

Options for services
  -c --config-file=<value>     3scale toolbox configuration file:
                          defaults to $HOME/.3scalerc.yaml
  -h --help                    Print help for this command
  -k --insecure                Proceed and operate even for server
                          connections otherwise considered insecure
  -v --version                 Print the version of this command
     --verbose                 Verbose mode
```

5.13.4. 显示服务

以下命令显示服务：

```
$ 3scale service show <remote> <service-id_or_system-name>
```

在显示服务时使用以下选项：

```
Options
   -o --output=<value>          Output format on stdout:
                           one of json|yaml

Options for services
   -c --config-file=<value>     3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
   -h --help                    Print help for this command
   -k --insecure                Proceed and operate even for server
                           connections otherwise considered insecure
   -v --version                 Print the version of this command
     --verbose                  Verbose mode
```

5.13.5. 删除服务

以下命令删除服务：

```
$ 3scale service delete <remote> <service-id_or_system-name>
```

在删除服务时使用以下选项：

```
Options for services
   -c --config-file=<value>     3scale toolbox configuration file:
                           defaults to $HOME/.3scalerc.yaml
   -h --help                    Print help for this command
   -k --insecure                Proceed and operate even for server
                           connections otherwise considered insecure
   -v --version                 Print the version of this command
     --verbose                  Verbose mode
```

5.14. 创建 ACTIVEDOCS

使用 3scale toolbox 在 Developer Portal 中创建、更新、列出或删除 ActiveDocs。

5.14.1. 创建新的 ActiveDocs

要从 API 定义创建一个新的 ActiveDocs，符合 OpenAPI 规格：

1. 将 API 定义添加到 3scale，选择性地为其指定名称：

   ```
   $ 3scale activedocs create <remote> <activedocs-name> <specification>
   ```

   ActiveDocs 的 OpenAPI 规格是必需的，且必须是以下值之一：

   - 可用路径中的文件名.
   - toolbox 可以从中下载内容的 URL。支持的方案是 **http** 和 **https**。
   - 从 **stdin** 标准输入流读取.这通过设置 **-** 值来控制。
     在创建 ActiveDocs 时使用以下选项：

     ```
     Options
        -d --description=<value>        Specify the description of
     ```

```
                              the ActiveDocs
    -i --service-id=<value>         Specify the Service ID
                              associated to the ActiveDocs
    -o --output=<value>             Output format on stdout: one
                              of json|yaml
    -p --published                  Specify to publish the
                              ActiveDocs on the Developer
                              Portal. Otherwise it is hidden.
    -s --system-name=<value>        Specify the system-name of
                              the ActiveDocs
       --skip-swagger-validations   Specify to skip validation
                              of the Swagger specification
Options for ActiveDocs
    -c --config-file=<value>        toolbox configuration file.
                              Defaults to $HOME/.3scalerc.yaml
    -h --help                       Print help for this command
    -k --insecure                   Proceed and operate even for
                              server connections otherwise
                              considered insecure
    -v --version                    Print the version of this command
       --verbose                    Verbose mode
```

2. 在 Developer Portal 中发布该定义。

5.14.2. 创建或更新 ActiveDocs

使用以下命令创建新的 ActiveDocs（如果不存在），或者使用新的 API 定义更新现有 ActiveDocs：

```
$ 3scale activedocs apply <remote> <activedocs_id_or_system_name>
```

在更新 ActiveDocs 时使用以下选项：

```
Options
  -d --description=<value>            Specify the description of the
                              ActiveDocs
       --hide                   Specify to hide the ActiveDocs
                              on the Developer Portal
  -i --service-id=<value>             Specify the Service ID associated
                              to the ActiveDocs
  -o --output=<value>                 Output format on stdout:
                              one of json|yaml
    --openapi-spec=<value>            Specify the Swagger specification.
                              Can be a file, a URL or '-' to read
                              from stdin. This is a mandatory
                              option when applying the ActiveDoc
                              for the first time.
  -p --publish                  Specify to publish the ActiveDocs
                              on the Developer Portal. Otherwise
                              it is hidden
  -s --name=<value>                   Specify the name of the ActiveDocs
    --skip-swagger-validations=<value>  Specify whether to skip validation
                              of the Swagger specification: true
                              or false. Defaults to true.

Options for ActiveDocs
```

```
  -c --config-file=<value>          3scale toolbox configuration file:
                                 defaults to $HOME/.3scalerc.yaml
  -h --help                        Print help for this command
  -k --insecure                     Proceed and operate even for server
                                 connections otherwise considered
                                 insecure
  -v --version                     Print the version of this command
     --verbose                     Verbose mode
```

注意

在 3scale 2.8 中 **activedocs apply --skip-swagger-validations** 的行为有所变化。您可能需要使用 **activedocs apply** 更新现有脚本。在以前的版本中，如果您没有在每个 **activedocs apply** 命令中指定这个选项，则不会跳过验证。现在，**--skip-swagger-validations** 默认为 **true**。

5.14.3. 列出 ActiveDocs

使用以下命令在 Developer Portal 中获取有关所有 ActiveDocs 的信息，包括：

- id
- 名称
- 系统名称
- 描述
- 发布（这意味着可以在开发人员门户中显示）
- 创建日期
- 最新更新日期

以下命令列出所有定义的 ActiveDocs：

```
$ 3scale activedocs list <remote>
```

在列出 ActiveDocs 时请使用以下选项：

```
Options
  -o --output=<value>          Output format on stdout:
                            one of json|yaml
  -s --service-ref=<value>     Filter the ActiveDocs by service
                            reference
Options for ActiveDocs
  -c --config-file=<value>     3scale toolbox configuration file:
                            defaults to $HOME/.3scalerc.yaml
  -h --help                    Print help for this command
  -k --insecure                Proceed and operate even for server
                            connections otherwise considered insecure
  -v --version                 Print the version of this command
     --verbose                 Verbose mode
```

5.14.4. 删除 ActiveDocs

以下命令删除 ActiveDocs：

```
$ 3scale activedocs delete <remote> <activedocs-id_or-system-name>
```

在删除 ActiveDocs 时使用以下选项：

```
Options for ActiveDocs
   -c --config-file=<value>      3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                  Proceed and operate even for server
                              connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.15. 列出代理配置

使用 3scale toolbox 列出、显示并在 Developer Portal 中提升所有定义的代理配置。

以下命令列出代理配置：

```
$ 3scale proxy-config list <remote> <service> <environment>
```

在列出代理配置时请使用以下选项：

```
Options
   -o --output=<value>           Output format on stdout:
                              one of json|yaml

Options for proxy-config
   -c --config-file=<value>      3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                  Proceed and operate even for server
                              connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.15.1. 显示代理配置

以下命令显示代理配置：

```
$ 3scale proxy-config show <remote> <service> <environment>
```

在显示代理配置时请使用以下选项：

```
Options
      --config-version=<value>   Specify the proxy configuration version.
                              If not specified, defaults to latest
   -o --output=<value>           Output format on stdout:
```

```
                        one of json|yaml

Options for proxy-config
   -c --config-file=<value>      3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
   -h --help                Print help for this command
   -k --insecure              Proceed and operate even for server
                        connections otherwise considered
                        insecure
   -v --version             Print the version of this command
      --verbose             Verbose mode
```

5.15.2. 提升代理配置

以下命令将最新的暂存代理配置提升到生产环境：

```
$ 3scale proxy-config promote <remote> <service>
```

在将最新的暂存代理配置提升到生产环境时，请使用以下选项：

```
Options for proxy-config
   -c --config-file=<value>      3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
   -h --help                Print help for this command
   -k --insecure              Proceed and operate even for server
                        connections otherwise considered insecure
   -v --version             Print the version of this command
      --verbose             Verbose mode
```

5.15.3. 导出代理配置

例如，如果您有一个自我管理的 APIcast 网关没有连接到 3scale 实例，请使用 **proxy-config export** 命令。在这种情况下，手动注入 3scale 配置，或使用 APICast 部署和配置选项注入 3scale 配置。在这两种情况下，您必须提供 3scale 配置。

以下命令导出可注入 APIcast 网关的配置：

```
$ 3scale proxy-config export <remote>
```

您可以在导出用作 3scale 配置文件 的供应商帐户的代理配置时指定以下选项：

```
Options for proxy-config
--environment=<value>      Gateway environment. Must be 'sandbox' or
                        'production' (default: sandbox)
-o --output=<value>        Output format. One of: json|yaml
```

5.15.4. 部署代理配置

以下 **deploy** 命令将您的 APIcast 配置提升到 3scale 中的暂存环境，或提升到生产环境（如果使用 Service Mesh）。

```
$ 3scale proxy deploy <remote> <service>
```

当使用 **deploy** 命令将 APIcast 配置提升到 stage 环境时，您可以指定以下选项：

```
-o --output=<value>        Output format. One of: json|yaml
```

5.15.5. 更新代理配置

以下 **update** 命令更新您的 APIcast 配置。

```
$ 3scale proxy update <remote> <service>
```

您可以使用 **update** 命令更新 APIcast 配置时指定以下选项：

```
-o --output=<value>           Output format. One of: json|yaml
-p --param=<value>             APIcast configuration parameters. Format:
                          [--param key=value]. Multiple options allowed.
```

5.15.6. 显示代理配置

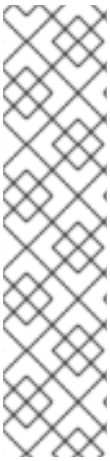
以下 **show** 命令获取您的 HTPasswded APIcast 配置。

```
$ 3scale proxy show <remote> <service>
```

在使用 **show** 命令时，您可以指定以下选项，以获取取消部署 APIcast 配置：

```
$ -o --output=<value>          Output format. One of: json|yaml
```

5.15.7. 部署代理配置（已弃用）



注意

在 3scale 2.12 中，对 **proxy-config deploy** 命令的支持已被弃用。

使用以下命令：

- **代理部署**
- **代理更新**
- **代理显示**

如需更多信息，请参阅部署代理配置。

以下 **deploy** 命令将您的 APIcast 配置提升到 3scale 中的暂存环境，或提升到生产环境（如果使用 Service Mesh）。

```
$ 3scale proxy-config deploy <remote> <service>
```

当使用 **deploy** 命令将 APIcast 配置提升到 stage 环境时，您可以指定以下选项：

```
$ -o --output=<value>        Output format. One of: json|yaml
```

其他资源

- 远程

5.16. 复制策略 REGISTRY

在以下情况下，使用 toolbox 命令将策略 registry 从 3scale 源帐户复制到目标帐户：

- 目标帐户中正在创建缺少的自定义策略。
- 目标帐户中正在更新匹配的自定义策略。
- 此 copy 命令具有幂等性。

注意

- 缺少的自定义策略定义为源帐户中存在的自定义策略，且不存在于帐户租户中。
- 匹配自定义策略定义为源和目标帐户中存在的自定义策略。

以下命令复制策略 registry：

```
$ 3scale policy-registry copy [opts] <source_remote> <target_remote>
```

```
Option for policy-registry
   -c --config-file=<value>      3scale toolbox configuration file:
                                 defaults to $HOME/.3scalerc.yaml
   -h --help                     Print help for this command
   -k --insecure                 Proceed and operate even for server
                                 connections otherwise considered insecure
   -v --version                  Print the version of this command
      --verbose                  Verbose mode
```

5.17. 列出应用

使用 3scale toolbox 列出、创建、显示、应用或删除应用程序开发人员门户。

以下命令列出应用程序：

```
$ 3scale application list [opts] <remote>
```

在列出应用程序时使用以下选项：

```
OPTIONS
      --account=<value>          Filter by account
   -o --output=<value>           Output format on stdout:
                                 one of json|yaml
      --plan=<value>             Filter by application plan. Service
                                 option required.
      --service=<value>          Filter by service

OPTIONS FOR APPLICATION
   -c --config-file=<value>      3scale toolbox configuration file:
```

```
                        defaults to $HOME/.3scalerc.yaml
  -h --help             Print help for this command
  -k --insecure          Proceed and operate even for server
                        connections otherwise considered insecure
  -v --version          Print the version of this command
    --verbose           Verbose mode
```

5.17.1. 创建应用程序

使用 create 命令创建与给定 3scale 帐户和应用计划相关联的一个应用。

所需的位置参数如下：

- **<service>** 参考。它可以是服务 **ID**，也可以是 service **system_name**。
- **<account>** 参考。它可以是以下之一：
  - 帐户 **ID**
  - 帐户管理员用户的**用户名、电子邮件** 或 **user_id**
  - **provider_key**
- **<application plan>** 参考。它可以是计划 **ID**，也可以是计划 **system_name**。
- **<name>** 应用程序名称。

以下命令创建应用程序：

```
$ 3scale application create [opts] <remote> <account> <service> <application-plan> <name>
```

在创建应用程序时使用以下选项：

```
OPTIONS
      --application-id=<value>     App ID or Client ID (for OAuth and
                           OpenID Connect authentication modes)
                           of the application to be created.
      --application-key=<value>    App Key(s) or Client Secret (for OAuth
                           and OpenID Connect authentication
                           modes) of the application created.
      --description=<value>        Application description
  -o --output=<value>              Output format on stdout:
                           one of json|yaml
      --redirect-url=<value>       OpenID Connect redirect url
      --user-key=<value>           User Key (API Key) of the application
                           to be created.

OPTIONS FOR APPLICATION
  -c --config-file=<value>      3scale toolbox configuration file:
                        defaults to $HOME/.3scalerc.yaml
  -h --help             Print help for this command
  -k --insecure          Proceed and operate even for server
                        connections otherwise considered insecure
  -v --version          Print the version of this command
    --verbose           Verbose mode
```

5.17.2. 显示应用程序

以下命令显示应用程序：

```
$ 3scale application show [opts] <remote> <application>
```

应用程序参数允许：

- **user_key** - API 密钥
- **App_id** - 来自 app_id/app_key 对或 *OAuth* 和 *OpenID Connect* (OIDC)验证模式的*客户端 ID*
- 应用程序内部 **ID**

```
OPTIONS
   -o --output=<value>          Output format on stdout:
                              one of json|yaml

OPTIONS FOR APPLICATION
   -c --config-file=<value>     3scale toolbox configuration file:
                              defaults to $HOME/.3scalerc.yaml
   -h --help                    Print help for this command
   -k --insecure                 Proceed and operate even for server
                              connections otherwise considered insecure
   -v --version                 Print the version of this command
     --verbose                  Verbose mode
```

5.17.3. 创建或更新应用程序

如果新应用程序不存在，或更新现有应用程序，使用以下命令创建新应用程序：

```
$ 3scale application apply [opts] <remote> <application>
```

应用程序参数允许：

- **user_key** - API 密钥
- **App_id** - 来自 app_id/app_key 对或 *OAuth* 和 *OIDC* 验证模式的*客户端 ID*
- 应用程序内部 **ID**
- 如果未找到应用程序且需要创建应用程序，则需要 **account** 可选参数。它可以是以下之一：
  - 帐户 **ID**
  - 3scale 帐户管理员用户的**用户名**、**电子邮件** 或 **user_id**
  - **provider_key**
- **名称** 不能用作唯一标识符，因为应用名称在 3scale 中不唯一。
- 使用 **--resume** 标志恢复暂停的应用。
- 暂停应用 - 将状态更改为通过 **--suspend** 标志暂停。

在更新应用程序时使用以下选项：

```
OPTIONS
      --account=<value>          Application's account. Required when
                                 creating
      --application-key=<value>   App Key(s) or Client Secret (for OAuth
                                 and OpenID Connect authentication
                                 modes) of the application to be
                                 created. Only used when application
                                 does not exist.
      --description=<value>       Application description
      --name=<value>             Application name
    -o --output=<value>          Output format on stdout:
                                 one of json|yaml
      --plan=<value>             Application's plan. Required when
                                 creating.
      --redirect-url=<value>      OpenID Connect redirect url
      --resume                   Resume a suspended application
      --service=<value>          Application's service. Required when
                                 creating.
      --suspend                   Suspends an application (changes the
                                 state to suspended)
      --user-key=<value>          User Key (API Key) of the application
                                 to be created.

OPTIONS FOR APPLICATION
    -c --config-file=<value>      3scale toolbox configuration file:
                                defaults to $HOME/.3scalerc.yaml
    -h --help                    Show help for this command
    -k --insecure                 Proceed and operate even for server
                                connections otherwise considered insecure
    -v --version                 Print the version of this command
       --verbose                 Verbose mode.
```

5.17.4. 取消应用程序

以下命令删除应用程序：

```
$ 3scale application delete [opts] <remote> <application>
```

应用程序参数允许：

- **user_key** - API 密钥
- **App_id** - 来自 app_id/app_key 对或 *OAuth* 和 *OIDC* 验证模式的*客户端 ID*
- 应用程序内部 **ID**

5.18. 导出产品

您可以使用 **yaml** 格式导出 3scale 产品定义，以便您可以将该产品导入到与源 3scale 实例没有连接的 3scale 实例中。您必须先设置 3scale 产品，然后才能导出该产品。请参阅 创建新产品来测试 API 调用。

当两个 3scale 实例具有网络连接时，当您想要在 3scale 实例中使用相同的 3scale 产品时，请使用 toolbox **3scale copy** 命令。

描述

当您导出 3scale 产品时，toolbox 以 **yaml** 格式序列化产品定义，该定义遵循 Product 和 Backend 自定义资源定义(CRD)。如需更多信息，请参阅使用 3scale API 管理操作器配置和调配 3scale。除了产品的基本信息外，输出 **yaml** 包括：

- 链接到产品的后端。
- 链接后端的指标、方法和映射规则.
- 应用计划中定义的限值和定价规则.
- 限值和定价规则引用的指标和方法。

导出产品是一种只读操作。换句话说，重复导出产品是安全的。toolbox 不会更改要导出的产品。如果要修改，您可以在将其导入到另一个 3scale 实例前修改 **yaml** 输出。

导出 3scale 产品适用于以下情况：

- 源和目标 3scale 实例之间没有连接。例如，当您想要在多个 3scale 实例中使用同一产品时，可能会存在严重网络限制阻止运行 toolbox **3scale copy** 命令。
- 您需要使用 Git 或其它源控制系统，以 **yaml** 格式维护 3scale 产品定义。

3scale toolbox **export** 和 **import** 命令也可能有助于备份和恢复产品定义。

格式

使用此格式运行 **export** 命令：

```
$ 3scale product export [-f output-file] <remote> <product>
```

export 命令可以将输出发送到 **stdout** 或文件。默认值为 **stdout**。要将输出发送到一个文件，请使用 **-f** 或 **--file** 选项指定带有 **.yaml** 的文件名称。

将 **<remote>** 替换为与您要从中导出产品的 3scale 实例关联的 3scale 供应商帐户别名或 URL。有关指定此功能的更多信息，请参阅管理远程访问凭证。

将 **<product>** 替换为您要导出的产品的系统名称或 3scale ID。此产品必须与您指定的 3scale 供应商帐户关联。您可以在产品 **Overview** 页面的 3scale 管理门户中找到产品的系统名称。要获得产品的 3scale ID，请运行 toolbox **3scale services show** 命令。

Example

以下命令从与 **my-3scale-1** 供应商帐户关联的 3scale 实例导出 **petstore** 产品，并将其输出到 **petstore-product.yaml** 文件：

```
$ 3scale product export -f petstore-product.yaml my-3scale-1 petstore
```

以下是 **Default API** 产品的序列化示例：

```
apiVersion: v1
kind: List
items:
```

```
- apiVersion: capabilities.3scale.net/v1beta1
  kind: Product
  metadata:
    annotations:
      3scale_toolbox_created_at: '2021-02-17T10:59:23Z'
      3scale_toolbox_version: 0.17.1
    name: api.xysnalcj
  spec:
    name: Default API
    systemName: api
    description: ''
    mappingRules:
    - httpMethod: GET
      pattern: "/v2"
      metricMethodRef: hits
      increment: 1
      last: false
    metrics:
      hits:
        friendlyName: Hits
        unit: hit
        description: Number of API hits
    methods:
      servicemethod01:
        friendlyName: servicemethod01
        description: ''
    policies:
    - name: apicast
      version: builtin
      configuration: {}
      enabled: true
    applicationPlans:
      basic:
        name: Basic
        appsRequireApproval: false
        trialPeriod: 0
        setupFee: 0.0
        custom: false
        state: published
        costMonth: 0.0
        pricingRules:
        - from: 1
          to: 1000
          pricePerUnit: 1.0
          metricMethodRef:
            systemName: hits
        limits:
        - period: hour
          value: 1222222
          metricMethodRef:
            systemName: hits
            backend: backend_01
    backendUsages:
      backend_01:
        path: "/v1/pets"
      backend_02:
```

```
      path: "/v1/cats"
  deployment:
    apicastSelfManaged:
      authentication:
        oidc:
          issuerType: rest
          issuerEndpoint: https://hello:test@example.com/auth/realms/3scale-api-consumers
          jwtClaimWithClientID: azp
          jwtClaimWithClientIDType: plain
          authenticationFlow:
            standardFlowEnabled: false
            implicitFlowEnabled: true
            serviceAccountsEnabled: false
            directAccessGrantsEnabled: true
          credentials: query
          security:
            hostHeader: ''
            secretToken: some_secret
          gatewayResponse:
            errorStatusAuthFailed: 403
            errorHeadersAuthFailed: text/plain; charset=us-ascii
            errorAuthFailed: Authentication failed
            errorStatusAuthMissing: 403
            errorHeadersAuthMissing: text/plain; charset=us-ascii
            errorAuthMissing: Authentication parameters missing
            errorStatusNoMatch: 404
            errorHeadersNoMatch: text/plain; charset=us-ascii
            errorNoMatch: No Mapping Rule matched
            errorStatusLimitsExceeded: 429
            errorHeadersLimitsExceeded: text/plain; charset=us-ascii
            errorLimitsExceeded: Usage limit exceeded
      stagingPublicBaseURL: http://staging.example.com:80
      productionPublicBaseURL: http://example.com:80
- apiVersion: capabilities.3scale.net/v1beta1
  kind: Backend
  metadata:
    annotations:
      3scale_toolbox_created_at: '2021-02-17T10:59:34Z'
      3scale_toolbox_version: 0.17.1
    name: backend.01.pcjwxbdu
  spec:
    name: Backend 01
    systemName: backend_01
    privateBaseURL: https://b1.example.com:443
    description: new desc
    mappingRules:
    - httpMethod: GET
      pattern: "/v1/pets"
      metricMethodRef: hits
      increment: 1
      last: false
    metrics:
      hits:
        friendlyName: Hits
        unit: hit
        description: Number of API hits
```

```
    methods:
      mybackendmethod01:
        friendlyName: mybackendmethod01
        description: ''
- apiVersion: capabilities.3scale.net/v1beta1
  kind: Backend
  metadata:
    annotations:
      3scale_toolbox_created_at: '2021-02-17T10:59:34Z'
      3scale_toolbox_version: 0.17.1
    name: backend.02.tiedgjsk
  spec:
    name: Backend 02
    systemName: backend_02
    privateBaseURL: https://b2.example.com:443
    description: ''
    mappingRules:
    - httpMethod: GET
      pattern: "/v1/cats"
      metricMethodRef: hits
      increment: 1
      last: false
    metrics:
      hits:
        friendlyName: Hits
        unit: hit
        description: Number of API hits
    methods:
      backend02_method01:
        friendlyName: backend02_method01
        description: ''
```

导出和传送到 Product CR

运行 **export** 命令时，您可以管道输出来创建 产品自定义资源(CR)。哪个 3scale 实例包含这个 CR 取决于以下条件：

- 如果定义了 **threescale-provider-account** secret，3scale operator 会在由该 secret 标识的 3scale 实例中创建 product CR。
- 如果没有定义 **threescale-provider-account** secret，那么如果命名空间中安装了 3scale 实例，则新产品 CR 将位于该命名空间中，3scale 操作器会在该命名空间中创建产品 CR。
- 如果没有定义 **threescale-provider-account** secret，如果新产品 CR 所在的命名空间不包含 3scale 实例，则 3scale 操作器会将产品 CR 标记为失败状态。

假设您在包含 **3scale-provider-account** 机密的命名空间中运行以下命令：toolbox 将 **petstore** CR 传送到 3scale 实例，该实例在 **3scale-provider-account** secret 中标识：

```
$ 3scale product export my-3scale-1 petstore | oc apply -f -
```

其他资源

- 使用 3scale 操作器配置和调配 3scale
- 3scale API Management 操作器如何标识自定义资源链接的租户

5.19. 导入产品

要在多个 3scale 实例中使用相同的 3scale 产品（当源和目标 3scale 实例没有网络连接时），请从一个 3scale 实例导出 3scale 产品，并将它导入到另一个 3scale 实例。要导入产品，请运行 toolbox **3scale product import** 命令。

当两个 3scale 实例具有网络连接时，当您想要在 3scale 实例中使用相同的 3scale 产品时，请使用 toolbox **3scale copy** 命令。

描述

当您导入 3scale 产品时，toolbox 需要以 **.yaml** 格式的序列化 **Product** 和 **Backend** 自定义资源定义 (CRD)。您可以通过运行 toolbox **3scale product export** 命令或手动创建 **.yaml** 格式的产品定义来获取此 **.yaml** 内容。

如果您导出了产品，导入的定义包含导出的内容，其中包括：

- 链接到产品的后端。
- 链接后端的指标、方法和映射规则.
- 应用计划中定义的限值和定价规则.
- 限值和定价规则引用的指标和方法。

如果要修改，您可以在导入 3scale 实例前修改导入的 **.yaml** 输出。

import 命令具有幂等性。您可以多次运行它来导入同一产品，并且生成的 3scale 配置保持不变。如果导入过程中出现错误，则可以安全地重新运行该命令。如果 **import** 过程无法在 3scale 实例中找到产品，它会创建该产品。它还会创建 **.yaml** 定义中定义的任何指标、方法或后端，且无法在 3scale 实例中找到。

在以下情况下导入 3scale 产品：

- 源和目标 3scale 实例之间没有连接。例如，当您想要在多个 3scale 实例中使用同一产品时，可能会存在严重网络限制阻止运行 toolbox **3scale copy** 命令。
- 您需要使用 Git 或其它源控制系统来维护 3scale 产品定义（采用 **.yaml** 格式）。

3scale toolbox **export** 和 **import** 命令也可能有助于备份和恢复产品定义。

格式

运行 **import** 命令使用这个格式：

```
$ 3scale product import [<options>] <remote>
```

import 命令从 **stdin** 或文件获取 **.yaml** 输入。默认值为 **stdin**。

您可以指定这些选项：

- **-f** 或 **--file** 后跟文件名从您指定的 **.yaml** 文件中获取输入。此文件必须包含一个 3scale 产品定义，该定义遵循 3scale **Product** 和 **Backend** CRD。
- **-o** 或 **--output** 后跟 **json** 或 **yaml** 输出报告，其中列出了您指定的格式导入的内容。默认输出格式为 **json**。

将 **<remote>** 替换为与您要导入该产品的 3scale 实例关联的 3scale 供应商帐户别名或 URL。有关指定此功能的更多信息，请参阅管理远程访问凭证。

Example

以下命令将 **petstore-product.yaml** 中定义的产品导入到与 **my-3scale-2** 供应商帐户关联的 3scale 实例中。默认情况下，导入内容的报告采用 **.json** 格式。

```
$ 3scale product import -f petstore-product.yaml my-3scale-2
```

import 命令输出列出导入项目的报告，例如：

```
api:
  product_id: 2555417888846
  backends:
    backend_01:
      backend_id: 73310
      missing_metrics_created: 1
      missing_methods_created: 1
      missing_mapping_rules_created: 1
    backend_02:
      backend_id: 73311
      missing_metrics_created: 0
      missing_methods_created: 2
      missing_mapping_rules_created: 1
  missing_methods_created: 1
  missing_metrics_created: 1
  missing_mapping_rules_created: 2
  missing_application_plans_created: 2
  application_plans:
    basic:
      application_plan_id: 2357356246461
      missing_limits_created: 7
      missing_pricing_rules_created: 7
    unlimited:
      application_plan_id: 2357356246462
      missing_limits_created: 1
      missing_pricing_rules_created: 0
```

导出产品系列化定义的示例位于导出产品的末尾。

5.20. 导出和导入产品策略链

您可以将产品的策略链导出或导入到 *yaml* 或 *json* 内容。在命令行中，根据其 **id** 或 **system** 引用该产品。您必须先设置 3scale 产品，然后才能导出或导入产品的策略链。请参阅创建新产品来测试 API 调用。

export 命令的特性

- 命令是远程产品的只读操作。
- 默认情况下，命令将输出写入标准输出 **stdout**。**f** 标志可用于将命令的输出写入文件。
- 命令输出格式为 **json** 或 **yaml**。请注意，默认格式为 **yaml**。

导出产品策略链的帮助选项

```
NAME
```

```
    export - export product policy chain
USAGE
    3scale policies export [opts] <remote>
    <product>
DESCRIPTION
    export product policy chain
OPTIONS
    -f --file=<value>           Write to file instead of stdout
    -o --output=<value>         Output format. One of: json|yaml
```

命令格式

- 以下是将策略链导出到 *yaml* 中的文件的命令格式：

```
$ 3scale policies export -f policies.yaml -o yaml remote_name product_name
```

import 命令的特性：

- 命令将从标准输入或 **stdin** 中读取输入。设置 **-f FILE** 标志时，将从文件中读取输入。设置 **-u** URL 标志后，将从 URL 中读取输入。
- 导入的内容可以是 **yaml** 或 **json**。您不需要指定格式，因为 toolbox 会自动检测到它。
- 现有策略链被新导入的链覆盖。然后实施 **SET** 语义。
- 所有内容验证都委派给 3scale API。

导入产品策略链的帮助选项

```
NAME
    import - import product policy chain
USAGE
    3scale policies import [opts] <remote>
    <product>
DESCRIPTION
    import product policy chain
OPTIONS
    -f --file=<value>           Read from file
    -u --url=<value>            Read from url
```

命令格式

- 以下是从文件中导入策略链的命令格式：

```
$ 3scale policies import -f plan.yaml remote_name product_name
```

- 以下是从 URI 导入策略链的命令格式：

```
$ 3scale policies import -f http[s]://domain/resource/path.yaml remote_name product_name
```

5.21. 复制 API 后端

在指定的 3scale 系统上创建指定源 API 后端的副本。默认情况下，目标系统首先使用源后端系统名称搜索：

- 如果没有找到带有所选系统名称的后端，则会创建它。
- 如果找到带有所选系统名称的后端，则会替换它。仅创建缺少的指标和方法，而映射规则完全替换为新指标和方法。

您可以使用 **--target_system_name** 选项覆盖系统名称。

复制的组件

复制以下 API 后端组件：

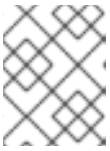
- 指标
- 方法
- 映射规则：它们会被复制和替换。

流程

- 输入以下命令复制 API 后端：

```
$ 3scale backend copy [opts] -s <source_remote> -d <target_remote> <source_backend>
```

指定的 3scale 实例可以是远程名称或 URL。



注意

您只能为每个命令复制单个 API 后端。您可以使用多个 命令复制多个后端。您可以通过指定不同的 **--target_system_name name** 来多次复制同一后端。

在复制 API 后端时使用以下选项：

```
Options
   -d --destination=<value>            3scale target instance: URL or
                                  remote name (required).
   -s --source=<value>                 3scale source instance: URL or
                                  remote name (required).
   -t --target_system_name=<value>     Target system name: defaults to
                                  source system name.
```

以下示例命令演示了如何通过为 **--target_system_name** 指定不同的值来多次复制 API 后端：

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale backend copy [-t
target_system_name] -s 3scale1 -d 3scale2 api_backend_01
```

5.22. 复制 API 产品

在目标 3scale 系统上创建指定源 API 产品的副本。默认情况下，源 API 产品系统名称首先搜索目标系统：

- 如果没有找到具有所选 **system-name** 的产品，则会创建它。

- 如果找到具有所选 **system-name** 的产品，则会更新它。仅创建缺少的指标和方法，而映射规则完全替换为新指标和方法。

您可以使用 **--target_system_name** 选项覆盖系统名称。

复制的组件

复制以下 API 产品组件：

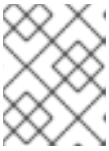
- 配置和设置
- 指标和方法
- 映射规则：它们会被复制和替换。
- 应用程序计划、定价规则和限值
- 应用程序用量规则
- 策略（policy）
- 后端
- ActiveDocs

流程

- 输入以下命令复制 API 产品：

```
$ 3scale product copy [opts] -s <source_remote> -d <target_remote> <source_product>
```

指定的 3scale 实例可以是远程名称或 URL。



注意

您只能为每个命令复制单个 API 产品。您可以使用多个 命令复制多个产品。您可以通过指定不同的 **--target_system_name name** 来多次复制同一产品。

在复制 API 产品时请使用以下选项：

```
Options
  -d --destination=<value>            3scale target instance: URL or
                                remote name (required).
  -s --source=<value>                 3scale source instance: URL or
                                remote name (required).
  -t --target_system_name=<value>     Target system name: defaults to
                                source system name.
```

以下示例命令演示了如何通过为 **--target_system_name** 指定不同的值来多次复制 API 产品：

```
$ podman run registry.redhat.io/3scale-amp2/toolbox-rhel8:3scale2.14 3scale product copy [-t
target_system_name] -s 3scale1 -d 3scale2 my_api_product_01
```

5.23. SSL 和 TLS 故障排除

本节介绍如何解决安全套接字层/交易层安全性(SSL/TLS)的问题。

如果您遇到与自签名 SSL 证书相关的问题，您可以下载和使用远程主机证书，如本节所述。例如，典型的错误包括 **SSL certificate problem: self signed certificate** 或 **self signed certificate in certificate chain**。

流程

1. 使用 **openssl** 下载远程主机证书。例如：

```
$ echo | openssl s_client -showcerts -servername self-signed.badssl.com -connect self-signed.badssl.com:443 2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > self-signed-cert.pem
```

2. 确保证书使用 **curl** 正常工作。例如：

```
$ SSL_CERT_FILE=self-signed-cert.pem curl -v https://self-signed.badssl.com
```

如果证书正常工作，您将不再获得 SSL 错误。如果证书无法正常工作，请尝试使用 **-k** 选项（或其长形式 **--insecure**）运行 **curl** 命令。这表示即使对于其他被视为不安全的服务器连接也一样希望继续。

3. 将 **SSL_CERT_FILE** 环境变量添加到 **3scale** 命令。例如：

```
$ podman run --env "SSL_CERT_FILE=/tmp/self-signed-cert.pem" -v $PWD/self-signed-cert.pem:/tmp/self-signed-cert.pem registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.14 3scale service list https://{ACCESS_KEY}@{3SCALE_ADMIN}-admin.{DOMAIN_NAME}
```

本例使用 Podman 卷在容器中挂载证书文件。它假定该文件位于当前的 **$PWD** 文件夹中。

另一种方法是使用 3scale toolbox 镜像作为基础镜像创建您自己的 toolbox 镜像，然后安装您自己的可信证书存储。

其他资源

- Red Hat Certificate System 文档
- 在 Red Hat Enterprise Linux 8 中构建、运行和管理 Linux 容器

第 6 章 在 3SCALE API 管理中映射 API 环境

API 提供程序提供对通过 3scale 管理门户管理的 API 的访问权限。然后，您可以在许多环境中部署 API 后端。API 后端环境包括以下：

- 用于开发、质量保证(QA)、登台和生产的不同环境。
- 用于管理其自身 API 后端组的团队或部门的不同环境。

红帽 3scale API 管理产品代表 API 的单一 API 或子集，但也可用于映射和管理不同的 API 后端环境。

要了解为您的 3scale 产品映射 API 环境的信息，请查看以下部分：

- 每个环境的产品
- 3scale API 管理内部实例
- 3scale API 管理混合方法
- 使用 APIcast 网关进行 3scale API 管理

6.1. 每个环境的产品

此方法为每个 API 后端环境使用单独的 3scale 产品。在每个产品中，配置生产网关和暂存网关，以便能够像使用 API 后端一样安全地测试并提升对网关配置的更改。

> Production Product => Production Product APIcast gateway => Production Product API upstream
> Staging Product => Staging Product APIcast gateway => Staging Product API upstream

为 API 后端环境配置产品，如下所示：

- 使用环境 API 后端的基础 URL 创建一个后端。
- 使用后端路径 / 将后端添加到环境的产品中。

开发环境

- 创建开发后端
  - **Name:** Dev
  - **Private Base URL:** API 后端的 URL
- 创建 Dev 产品
  - **Production Public Base URL: https://dev-api-backend.yourdomain.com**
  - **Staging Public Base URL: https://dev-api-backend.yourdomain.com**
  - 使用后端路径 / 添加 Dev 后端。

QA 环境

- 创建 QA 后端
  - **名称：** QA

- **Private Base URL:** API 后端的 URL

- 创建 QA 产品
  - **Production Public Base URL: https://qa-api-backend.yourdomain.com**
  - **Staging Public Base URL: https://qa-api-backend.yourdomain.com**
  - 使用后端路径 / 添加 QA 后端。

生产环境

- 创建生产后端
  - **Name:** Prod
  - **Private Base URL:** API 后端的 URL
- 创建生产产品
  - **Production Public Base URL: https://prod-api-backend.yourdomain.com**
  - **Staging Public Base URL: https://prod-api-backend.yourdomain.com**
  - 使用后端路径 / 添加生产环境后端。

其他资源

- 3scale API 管理 的第一步。

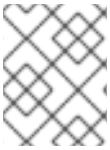
6.2. 3SCALE API 管理内部实例

对于 3scale 内部部署实例，有多种方式可以设置 3scale 来管理 API 后端环境。

- 每个 API 后端环境有一个单独的 3scale 实例
- 使用 多租户 功能的 3scale 实例

6.2.1. 每个环境隔离 3scale API 管理实例

在这种方法中，为每个 API 后端环境部署一个单独的 3scale 实例。这种架构的优势在于每个环境都将相互隔离，因此没有共享的数据库或其他资源。例如，在一个环境中进行的任何负载测试都不会影响其他环境中的资源。



注意

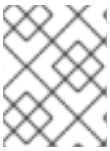
如前文所述，这种分离安装具有优势，但需要更多运营资源和维护。OpenShift 管理控制台中需要这些额外的资源，而不一定在 3scale 层上。

6.2.2. 每个环境隔离 3scale API 管理租户

在这个方法中使用了一个 3scale 实例，但多租户功能用于支持多个 API 后端。

有两个选项：

- 在单个租户内，创建环境和 3scale 产品之间的 1 到 1 到 1 个映射。
- 根据需要，在每个租户具有一个或多个产品的环境和租户之间创建一个 1 到 1 的映射。
  - 将有三个与 API 后端环境对应的租户，即 dev-tenant、qa-tenant、prod-tenant。这种方法的优点在于，它允许环境的逻辑分割，但使用共享的物理资源。



注意

最终，在分析将 API 环境映射到具有多个租户的单一安装的最佳策略时，需要考虑共享的物理资源。

6.3. 3SCALE API 管理混合方法

3scale API 管理内部实例中描述的方法 可以组合使用。例如：

- 用于生产环境的独立 3scale 实例。
- 为 dev 和 qa 中的非生产环境提供单独的 3scale 实例，具有单独的租户。

6.4. 使用 APICAST 网关进行 3SCALE API 管理

对于 3scale 内部部署实例，有两个替代方案可用来设置 3scale 来管理 API 后端环境：

- 每个 3scale 安装附带两个内置 APIcast 网关，供暂存和生产使用。
- 将 额外的 APIcast 网关部署到运行 3scale 的 OpenShift 集群。

6.4.1. APIcast 内置默认网关

当使用 APIcast 内置网关时，使用上述方法配置的 API 后端将自动 处理带有 APIcast 网关的 3scale API 管理。当 3scale Master Admin 添加租户时，将为 production 和暂存内置 APIcast 网关中的租户创建一个路由。请参阅 了解多租户子域

- **<API_NAME>-<TENANT_NAME>-apicast.staging.<WILDCARD_DOMAIN>**
- **<API_NAME>-<TENANT_NAME>-apicast.production.<WIDLCARD_DOMAIN>**

因此，映射到不同租户的每个 API 后端环境都会获得自己的路由。例如：

- Dev **<API_NAME>-dev-apicast.staging.<WILDCARD_DOMAIN>**
- QA **<API_NAME>-qa-apicast.staging.<WILDCARD_DOMAIN>**
- Prod **<API_NAME>-prod-apicast.staging.<WILDCARD_DOMAIN>**

6.4.2. 额外的 APIcast 网关

额外的 APIcast 网关是部署到与 3scale 实例所运行集群不同的 OpenShift 集群上。设置和使用其他 APIcast 网关的方法不止一种。启动 APIcast 时使用的环境变量变量 **THREESCALE_PORTAL_ENDPOINT** 的值取决于如何设置额外的 APIcast 网关。

单独的 APIcast 网关可用于每个 API 后端环境。例如：

```
DEV_APICAST -> DEV_TENANT ; DEV_APICAST started with
```

```
THREESCALE_PORTAL_ENDPOINT = admin portal for DEV_TENANT
QA_APICAST -> QA_TENANT ; QA_APICAST started with THREESCALE_PORTAL_ENDPOINT =
admin portal for QA_APICAST
PROD_APICAST -> PROD_TENANT ; PROD_APICAST started with
THREESCALE_PORTAL_ENDPOINT = admin portal for PROD_APICAST
```

THREESCALE_PORTAL_ENDPOINT 由 APIcast 用于下载配置。映射到 API 后端环境的每一租户都使用单独的 APIcast 网关。**THREESCALE_PORTAL_ENDPOINT** 设置为租户的管理门户，其中包含特定于该 API 后端环境的所有产品配置。

单个 APIcast 网关可用于多个 API 后端环境。在本例中，**THREESCALE_PORTAL_ENDPOINT** 设置为主管理门户。

其他资源

- API 供应商
- 产品

第 7 章 使用 3SCALE API 管理 TOOLBOX 自动执行 API 生命周期

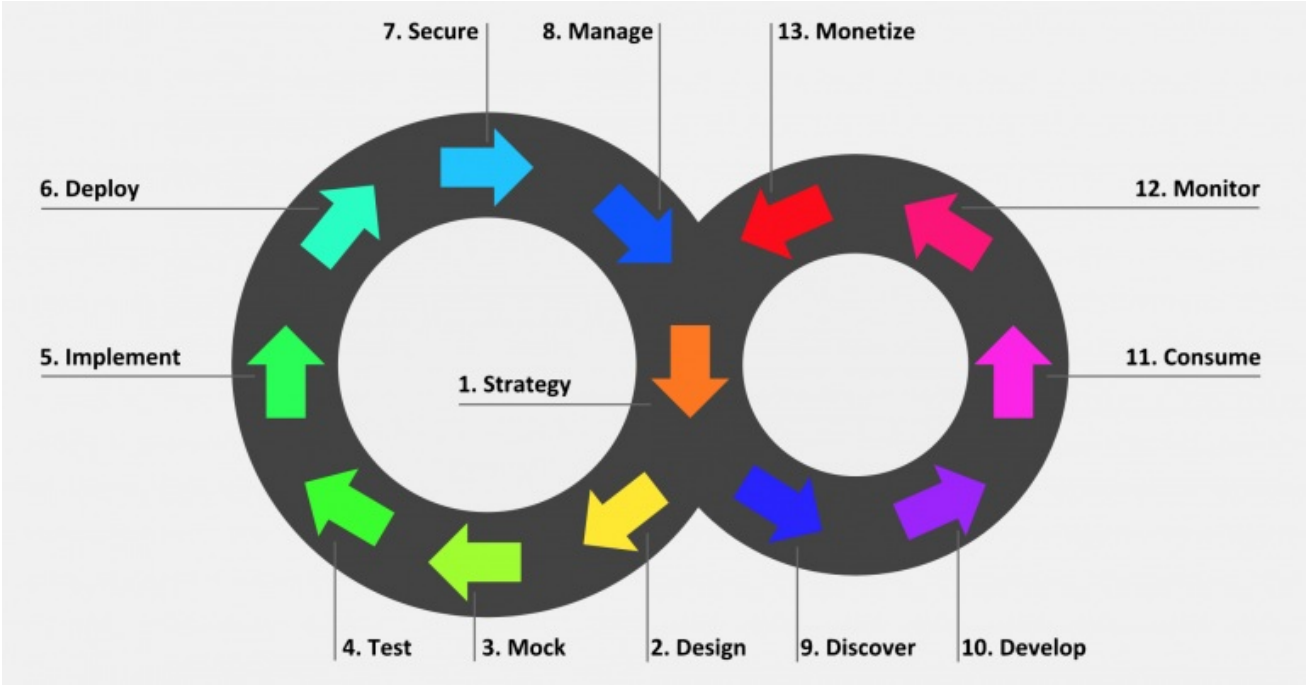
本主题介绍了红帽 3scale API 管理的 API 生命周期的概念，并演示了 API 供应商如何使用 3scale toolbox 命令使用 Jenkins 持续集成/持续部署(CI/CD)管道自动执行部署阶段。它描述了如何部署示例 Jenkins CI/CD 管道、如何使用 3scale 共享库创建自定义 Jenkins 管道以及如何从头开始创建自定义管道：

- 第 7.1 节 “API 生命周期阶段概述”
- 第 7.2 节 “部署示例 Jenkins CI/CD 管道”
- 第 7.3 节 “使用 3scale API 管理 Jenkins 共享库创建管道”
- 第 7.4 节 “使用 Jenkinsfile 创建管道”

7.1. API 生命周期阶段概述

API 生命周期描述了在创建 API 时需要的所有活动，直到它被弃用。3scale 可让 API 供应商执行完整的 API 生命周期管理。本节解释了 API 生命周期中的每个阶段，并描述了其目标和预期结果。

下图显示了左侧基于 API 供应商的阶段，以及右侧基于 API 使用者的阶段：



注意

红帽目前支持 API 供应商周期的设计、实施、部署和管理阶段以及 API 消费者周期的所有阶段。

7.1.1. API 供应商周期

API 提供程序周期阶段基于指定、开发和部署 API。下文描述了每个阶段的目标和结果：

表 7.1. API 供应商生命周期阶段

Stage	*目标*	*结果*
1.Strategy	确定 API 的公司战略，包括目标、资源、目标市场、时间线和制定计划。	制定企业战略的清晰计划以实现目标。
2.Design	提前创建 API 合同，以中断项目之间的依赖关系、收集反馈并缩短风险和上市时间（例如，使用 Apicurio Studio）。	以使用者为中心的 API 合同定义可通过 API 交换的消息。API 用户提供了反馈。
3.Mock	进一步指定使用真实示例和载荷的 API 合同，供 API 用户用来开始实施。	模拟 API 是实时的，返回真实示例。API 合同附带示例。
4.Test	进一步指定具有可用于测试所开发的 API 的业务预期的 API 合同。	创建一组验收测试。API 文档已根据业务预期完成。
5.Implement	使用红帽 Fuse 或您选择的开发语言等集成框架实施 API。确保实施与 API 合同匹配。	API 已被实施。如果需要自定义 API 管理功能，还会开发 3scale APIcast 策略。
6.Deploy	使用带有 3scale toolbox 的 CI/CD 管道自动执行 API 集成、测试、部署和管理。	CI/CD 管道以自动化方式集成、测试、部署和管理 API 到生产环境。
7.Secure	确保 API 的安全（例如，使用安全开发实践和自动安全测试）。	制定安全指南、流程和门。
8.Manage	大规模管理环境、版本控制、弃用和停用之间的 API 提升。	进程和工具已经到位，可以大规模管理 API（例如，语义版本控制以防止破坏对 API 的更改）。

7.1.2. API 使用者周期

API 消费者周期阶段基于提升、分发和调整 API 以供使用。下文描述了每个阶段的目标和结果：

表 7.2. API 消费者生命周期阶段

Stage	*目标*	*结果*
9.Discover	将 API 提升给第三方开发人员、合作伙伴和内部用户。	开发人员门户是实时的，最新的文档会持续推送到此开发人员门户（例如，使用 3scale ActiveDocs）。
10.Develop	指导并允许第三方开发人员、合作伙伴和内部用户基于 API 开发应用程序。	开发人员门户包括最佳实践、指南和建议。API 开发人员有权访问模拟和测试端点来开发他们的软件。

Stage	*目标*	*结果*
11.Consume	大规模处理不断增加的 API 用户并管理 API 使用者。	分阶段的应用计划可供消费，并持续推动最新的价格和限制。API 用户可以从 CI/CD 管道集成 API 密钥或客户端 ID/secret 生成。
12.Monitor	收集有关 API 健康、质量和开发人员参与情况（例如，时间到第一个 Hello World！）的指标。	建立了监控系统。仪表板显示 API 的 KPI（例如，正常运行时间、每分钟请求数、延迟等）。
13.Monetize	按规模驱动新的地区（此阶段是可选的）。	例如，针对大量小型 API 用户时，启用货币化，并且用户按照使用情况自动计费。

7.2. 部署示例 JENKINS CI/CD 管道

使用 3scale toolbox 的 API 生命周期自动化侧重于 API 生命周期的部署阶段，并可让您使用 CI/CD 管道来自动化 API 管理解决方案。本节介绍如何部署调用 3scale toolbox 的示例 Jenkins 管道：

- 第 7.2.1 节 “Jenkins CI/CD 管道示例”
- 第 7.2.2 节 “设置 3scale API 管理托管环境”
- 第 7.2.3 节 “设置 3scale API 管理内部环境”
- 第 7.2.4 节 “为 OpenID Connect 部署红帽单点登录”
- 第 7.2.5 节 “安装 3scale API 管理 toolbox 并启用访问”
- 第 7.2.6 节 “部署 API 后端”
- 第 7.2.7 节 “部署自我管理的 APIcast 实例”
- 第 7.2.8 节 “安装和部署示例管道”
- 第 7.2.9 节 “3scale API 管理 toolbox 的 API 生命周期自动化限制”

7.2.1. Jenkins CI/CD 管道示例

以下示例在 Red Hat Integration 存储库中提供，作为如何为 API 生命周期自动化创建和部署 Jenkins 管道的示例：

表 7.3. Jenkins 共享库管道示例

管道示例	*目标环境*	*安全性*
SaaS - API 密钥	3scale 托管	API 密钥
混合 - 开放	3scale 托管和 3scale 内部部署，APIcast 自我管理	None

管道示例	目标环境	安全性
Hybrid - OpenID Connect	3scale 托管和 3scale 内部部署，APIcast 自我管理	OpenID Connect(OIDC)
多环境	3scale 在开发、测试和生产上托管，APIcast 自我管理	API 密钥
语义版本	3scale 在开发、测试和生产上托管，APIcast 自我管理	API 密钥、无、OIDC

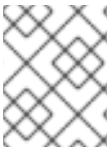
这些示例使用 3scale Jenkins 共享库调用 3scale toolbox 来演示关键 API 管理功能。执行本主题中的设置步骤后，您可以使用 红帽集成存储库中为各个示例用例 提供的 OpenShift 模板来安装管道。

重要

示例管道和应用程序仅作为示例提供。红帽完全支持底层 API、CLI 和其他示例管道利用的接口。您对管道所做的任何修改都不受红帽直接支持。

7.2.2. 设置 3scale API 管理托管环境

所有 Jenkins CI/CD 管道示例都需要设置 3scale Hosted 环境。



注意

SaaS - API 键、**多环境**和 **Semantic 版本管道** 仅使用 3scale Hosted。**混合 - 开放** 和混合 **- OIDC** 管道也使用 3scale 内部部署。另请参阅 设置 3scale 内部环境。

先决条件

- 您必须有一个 Linux 工作站。
- 您必须有一个 3scale 托管环境。
- 您必须有一个 OpenShift 3.11 集群。OpenShift 4 目前不受支持。
  - 有关支持配置的更多信息，请参阅 Red Hat 3scale API 管理支持的配置 页面。
- 如 OpenShift 文档 中所述，确保 OpenShift 路由器上已启用了通配符路由。

流程

1. 登录您的 3scale 托管管理门户控制台。
2. 生成对帐户管理 API 具有写入访问权限的新访问令牌。
3. 保存生成的访问令牌以供以后使用。例如：

```
$ export SAAS_ACCESS_TOKEN=123...456
```

4. 保存 3scale 租户的名称，以备以后使用。这是管理门户 URL 中 **-admin.3scale.net** 前面的字符串。例如：

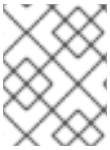
```
$ export SAAS_TENANT=my_username
```

5. 导航到管理门户中的 **Audience** > **Accounts** > **Listing**。
6. 单击 **Developer**。
7. 保存 **开发人员帐户 ID**。这是 **/buyers/accounts/** 后 URL 的最后一部分。例如：

```
$ export SAAS_DEVELOPER_ACCOUNT_ID=123...456
```

7.2.3. 设置 3scale API 管理内部环境

混合 - 开放和混合 - OIDC 示例 Jenkins CI/CD 管道只需要设置 3scale 内部环境。



注意

如果要使用这些 **混合** 示例管道，您必须设置 3scale 内部部署环境和 3scale 托管环境。另请参阅 设置 3scale API 管理托管环境。

先决条件

- 您必须有一个 Linux 工作站。
- 您必须有一个 3scale 内部环境。有关使用 OpenShift 中的模板安装 3scale 内部的详情，请参阅 3scale API 管理安装文档。
- 您必须有一个 OpenShift 4.x 集群。
  - 有关支持配置的更多信息，请参阅 Red Hat 3scale API 管理支持的配置 页面。
- 如 OpenShift 文档 中所述，确保 OpenShift 路由器上已启用了通配符路由。

流程

1. 登录您的 3scale 内部管理门户控制台。
2. 生成对帐户管理 API 具有写入访问权限的新访问令牌。
3. 保存生成的访问令牌以供以后使用。例如：

```
$ export SAAS_ACCESS_TOKEN=123...456
```

4. 保存 3scale 租户名称以供以后使用：

```
$ export ONPREM_ADMIN_PORTAL_HOSTNAME="$(oc get route system-provider-admin -o jsonpath='{.spec.host}')"
```

5. 定义通配符路由：

```
$ export OPENSHIFT_ROUTER_SUFFIX=app.openshift.test # Replace me!
```

```
$ export APICAST_ONPREM_STAGING_WILDCARD_DOMAIN=onprem-staging.$OPENSHIFT_ROUTER_SUFFIX

$ export APICAST_ONPREM_PRODUCTION_WILDCARD_DOMAIN=onprem-production.$OPENSHIFT_ROUTER_SUFFIX
```

注意

您必须将 **OPENSHIFT_ROUTER_SUFFIX** 的值设置为 OpenShift 路由器的后缀（如 **app.openshift.test**）。

6. 将通配符路由添加到现有的 3scale 内部实例中：

```
$ oc create route edge apicast-wildcard-staging --service=apicast-staging --hostname="wildcard.$APICAST_ONPREM_STAGING_WILDCARD_DOMAIN" --insecure-policy=Allow --wildcard-policy=Subdomain

$ oc create route edge apicast-wildcard-production --service=apicast-production --hostname="wildcard.$APICAST_ONPREM_PRODUCTION_WILDCARD_DOMAIN" --insecure-policy=Allow --wildcard-policy=Subdomain
```

7. 导航到管理门户中的 **Audience** > **Accounts** > **Listing**。
8. 单击 **Developer**。
9. 保存 **开发人员帐户 ID**。这是 **/buyers/accounts/** 后 URL 的最后一部分：

```
$ export ONPREM_DEVELOPER_ACCOUNT_ID=5
```

7.2.4. 为 OpenID Connect 部署红帽单点登录

如果您使用 **Hybrid - OpenID Connect (OIDC)** 或 **Semantic 版本** 示例管道，请执行本节中的步骤以使用 3scale 部署红帽单点登录。这是 OIDC 身份验证所需要的，在两个样本中使用。

流程

1. 按照 Red Hat Single sign-on 文档中所述，部署红帽单点登录 7.3。
以下示例命令提供了一个简短概述：

```
$ oc replace -n openshift --force -f https://raw.githubusercontent.com/jboss-container-images/redhat-sso-7-openshift-image/sso73-dev/templates/sso73-image-stream.json

$ oc replace -n openshift --force -f https://raw.githubusercontent.com/jboss-container-images/redhat-sso-7-openshift-image/sso73-dev/templates/sso73-x509-postgresql-persistent.json

$ oc -n openshift import-image redhat-sso73-openshift:1.0

$ oc policy add-role-to-user view system:serviceaccount:$(oc project -q):default

$ oc new-app --template=sso73-x509-postgresql-persistent --name=sso -p DB_USERNAME=sso -p SSO_ADMIN_USERNAME=admin -p DB_DATABASE=sso
```

2. 保存 Red Hat 单点登录安装的主机名，以便稍后使用：

```
$ export SSO_HOSTNAME="$(oc get route sso -o jsonpath='{.spec.host}')"
```

3. 按照 3scale API 管理开发人员门户文档中所述，为 3scale 配置红帽单点登录 文档。

4. 保存 realm 名称、客户端 ID 和客户端 secret 以供以后使用：

```
$ export REALM=3scale

$ export CLIENT_ID=3scale-admin

$ export CLIENT_SECRET=123...456
```

7.2.5. 安装 3scale API 管理 toolbox 并启用访问

本节论述了如何安装 toolbox、创建远程 3scale 实例，以及置备用于访问管理门户的 secret。

流程

1. 如 3scale API 管理 toolbox 中所述，在本地安装 3scale toolbox 。

2. 运行适当的 toolbox 命令创建 3scale 远程实例：

3scale 托管

```
$ 3scale remote add 3scale-saas "https://$SAAS_ACCESS_TOKEN@$SAAS_TENANT-admin.3scale.net/"
```

3scale On-premises

```
$ 3scale remote add 3scale-onprem "https://$ONPREM_ACCESS_TOKEN@$ONPREM_ADMIN_PORTAL_HOSTNAME/"
```

3. 运行以下命令来置备包含 3scale 管理门户和访问令牌的 secret：

```
$ oc create secret generic 3scale-toolbox -n "$TOOLBOX_NAMESPACE" --from-file="$HOME/.3scalerc.yaml"
```

7.2.6. 部署 API 后端

本节介绍如何部署示例管道提供的示例 API 后端。在创建和部署自己的管道时，您可以根据需要替换您自己的 API 后端。

流程

1. 部署示例 Beer Catalog API 后端，以便与以下示例搭配使用：

- **SaaS - API 密钥**
- **混合 - 开放**
- **混合 - OIDC**

```
$ oc new-app -n "$TOOLBOX_NAMESPACE" -i openshift/redhat-openjdk18-
openshift:1.4 https://github.com/microcks/api-lifecycle.git --context-dir=/beer-catalog-
demo/api-implementation --name=beer-catalog

$ oc expose -n "$TOOLBOX_NAMESPACE" svc/beer-catalog
```

2. 保存 Beer Catalog API 主机名以供以后使用：

```
$ export BEER_CATALOG_HOSTNAME="$(oc get route -n "$TOOLBOX_NAMESPACE"
beer-catalog -o jsonpath='{.spec.host}')"
```

3. 部署示例 Red Hat Event API 后端，以便与以下示例搭配使用：

- **多环境**
- **语义版本**

```
$ oc new-app -n "$TOOLBOX_NAMESPACE" -i openshift/nodejs:10
'https://github.com/nmasse-itix/rhte-api.git#085b015' --name=event-api

$ oc expose -n "$TOOLBOX_NAMESPACE" svc/event-api
```

4. 保存 Event API 主机名供以后使用：

```
$ export EVENT_API_HOSTNAME="$(oc get route -n "$TOOLBOX_NAMESPACE" event-
api -o jsonpath='{.spec.host}')"
```

7.2.7. 部署自我管理的 APIcast 实例

本节用于 3scale 托管环境中的 APIcast 自我管理实例。它应用到除 **SaaS - API 密钥** 以外的所有示例管道。

流程

1. 定义通配符路由：

```
$ export APICAST_SELF_MANAGED_STAGING_WILDCARD_DOMAIN=saas-
staging.$OPENSHIFT_ROUTER_SUFFIX

$ export APICAST_SELF_MANAGED_PRODUCTION_WILDCARD_DOMAIN=saas-
production.$OPENSHIFT_ROUTER_SUFFIX
```

2. 在项目中部署 APIcast 自我管理实例：

```
$ oc create secret generic 3scale-tenant --from-
literal=password=https://$SAAS_ACCESS_TOKEN@$SAAS_TENANT-admin.3scale.net

$ oc create -f https://raw.githubusercontent.com/3scale/apicast/v3.5.0/openshift/apicast-
template.yml

$ oc new-app --template=3scale-gateway --name=apicast-staging -p
CONFIGURATION_URL_SECRET=3scale-tenant -p CONFIGURATION_CACHE=0 -p
RESPONSE_CODES=true -p LOG_LEVEL=info -p CONFIGURATION_LOADER=lazy -p
```

```
APICAST_NAME=apicast-staging -p DEPLOYMENT_ENVIRONMENT=sandbox -p
IMAGE_NAME=registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.14

$ oc new-app --template=3scale-gateway --name=apicast-production -p
CONFIGURATION_URL_SECRET=3scale-tenant -p CONFIGURATION_CACHE=60 -p
RESPONSE_CODES=true -p LOG_LEVEL=info -p CONFIGURATION_LOADER=boot -p
APICAST_NAME=apicast-production -p DEPLOYMENT_ENVIRONMENT=production -p
IMAGE_NAME=registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.14

$ oc scale dc/apicast-staging --replicas=1

$ oc scale dc/apicast-production --replicas=1

$ oc create route edge apicast-staging --service=apicast-staging --
hostname="wildcard.$APICAST_SELF_MANAGED_STAGING_WILDCARD_DOMAIN" --
insecure-policy=Allow --wildcard-policy=Subdomain

$ oc create route edge apicast-production --service=apicast-production --
hostname="wildcard.$APICAST_SELF_MANAGED_PRODUCTION_WILDCARD_DOMAIN"
--insecure-policy=Allow --wildcard-policy=Subdomain
```

7.2.8. 安装和部署示例管道

设置所需的环境后，您可以使用 红帽集成存储库中为各个示例用例提供的 OpenShift 模板来安装和部署示例 管道。例如，本节仅显示 **SaaS - API 密钥** 示例。

流程

1. 使用提供的 OpenShift 模板安装 Jenkins 管道：

```
$ oc process -f saas-usecase-apikey/setup.yaml \
        -p DEVELOPER_ACCOUNT_ID="$SAAS_DEVELOPER_ACCOUNT_ID" \
        -p PRIVATE_BASE_URL="http://$BEER_CATALOG_HOSTNAME" \
        -p NAMESPACE="$TOOLBOX_NAMESPACE" |oc create -f -
```

2. 按如下方式部署示例：

```
$ oc start-build saas-usecase-apikey
```

其他资源

- Red Hat Integration 仓库中的用例示例

7.2.9. 3scale API 管理 toolbox 的 API 生命周期自动化限制

这个版本有以下限制：

OpenShift 支持

OpenShift 3.11 仅支持示例管道。OpenShift 4 目前不受支持。有关支持配置的更多信息，请参阅 Red Hat 3scale API 管理支持的配置 页面。

更新应用程序

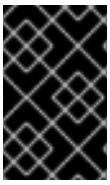
- 您可以对应用程序使用 **3scale application apply** toolbox 命令创建和更新应用程序。创建命令支持帐户、计划、服务和应用程序键。
- 更新命令不支持对帐户、计划或服务的更改。如果传递了更改，则会触发管道，不会显示任何错误，但不会更新这些字段。

复制服务

当使用 **3scale copy service** toolbox 命令复制带有自定义策略的服务时，您必须首先单独复制自定义策略。

7.3. 使用 3SCALE API 管理 JENKINS 共享库创建管道

本节提供创建使用 3scale toolbox 的自定义 Jenkins 管道的最佳实践。它解释了如何在 Groovy 中编写 Jenkins 管道，该管道使用 3scale Jenkins 共享库根据示例应用程序调用 toolbox。如需了解更多详细信息，请参阅 Jenkins 共享库。



重要

红帽支持红帽集成存储库中提供的 Jenkins 管道示例。

对这些管道所做的任何修改都不受红帽直接支持。不支持您为环境创建的自定义管道。

先决条件

- 部署示例 Jenkins CI/CD 管道 .
- 您的 API 必须具有 OpenAPI 规格文件。例如，您可以使用 Apicurio Studio 生成此操作。

流程

1. 将以下内容添加到 Jenkins 管道的开头，以引用管道的 3scale 共享库：

```
#!groovy

library identifier: '3scale-toolbox-jenkins@master',
  retriever: modernSCM([$class: 'GitSCMSource',
    remote: 'https://github.com/rh-integration/3scale-toolbox-jenkins.git'])
```

2. 声明一个全局变量来存放 三个**scaleService** 对象，以便您可以在管道的不同阶段使用它。

```
def service = null
```

3. 使用所有相关信息创建 **ThreescaleService** ：

```
stage("Prepare") {
  service = toolbox.prepareThreescaleService(
    openapi: [ filename: "swagger.json" ],
    environment: [ baseSystemName: "my_service" ],
    toolbox: [ openshiftProject: "toolbox",
               destination: "3scale-tenant",
               secretName: "3scale-toolbox" ],
    service: [:],
    applications: [
      [ name: "my-test-app", description: "This is used for tests", plan: "test", account: "
```

```
<CHANGE_ME>" ]
        ],
    applicationPlans: [
      [ systemName: "test", name: "Test", defaultPlan: true, published: true ],
      [ systemName: "silver", name: "Silver" ],
      [ artefactFile: "https://raw.githubusercontent.com/my_username/API-Lifecycle-
Mockup/master/testcase-01/plan.yaml"],
    ]
   )

 echo "toolbox version = " + service.toolbox.getToolboxVersion()
}
```

- **OpenAPI.filename** 是包含 OpenAPI 规范的文件的路径。
- **environment.baseSystemName** 用于根据 OpenAPI 规格 **info.version** 中的 **environment.environmentName** 和 API 主要版本计算最终 **system_name**。
- **toolbox.openshiftProject** 是创建 Kubernetes 作业的 OpenShift 项目。
- **toolbox.secretName** 是包含 3scale toolbox 配置文件的 Kubernetes secret 的名称，如 安装 3scale API Management toolbox 并启用访问 所示。
- **toolbox.destination** 是 3scale toolbox 远程实例的名称。
- **applicationPlans** 是使用一个 **.yaml** 文件或提供应用程序计划属性详情创建的应用程序计划列表。

4. 添加一个管道阶段来在 3scale 中置备服务：

```
stage("Import OpenAPI") {
 service.importOpenAPI()
 echo "Service with system_name ${service.environment.targetSystemName} created !"
}
```

5. 添加一个阶段来创建应用程序计划：

```
stage("Create an Application Plan") {
 service.applyApplicationPlans()
}
```

6. 添加全局变量和阶段来创建测试应用程序：

```
stage("Create an Application") {
 service.applyApplication()
}
```

7. 添加阶段来运行集成测试。使用 APIcast Hosted 实例时，您必须获取代理定义来提取暂存公共 URL：

```
stage("Run integration tests") {
 def proxy = service.readProxy("sandbox")
 sh """set -e +x
 curl -f -w "ListBeers: %{http_code}\n" -o /dev/null -s ${proxy.sandbox_endpoint}/api/beer -H
'api-key: ${service.applications[0].userkey}'
```

```
  curl -f -w "GetBeer: %{http_code}\n" -o /dev/null -s
${proxy.sandbox_endpoint}/api/beer/Weissbier -H 'api-key: ${service.applications[0].userkey}'
  curl -f -w "FindBeersByStatus: %{http_code}\n" -o /dev/null -s
${proxy.sandbox_endpoint}/api/beer/findByStatus/   available -H 'api-key:
${service.applications[0].userkey}'
  """
}
```

8. 添加一个阶段来将 API 提升到生产环境：

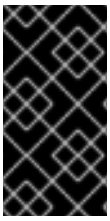
```
stage("Promote to production") {
  service.promoteToProduction()
}
```

其他资源

- 使用 Jenkinsfile 创建管道
- 3scale API 管理 toolbox

7.4. 使用 JENKINSFILE 创建管道

本节提供在使用 3scale toolbox 的 Groovy 中从头开始编写自定义 **Jenkinsfile** 的最佳实践。



重要

红帽支持红帽集成存储库中提供的 Jenkins 管道示例。

对这些管道所做的任何修改都不受红帽直接支持。不支持您为环境创建的自定义管道。本节仅供参考。

先决条件

- 部署示例 Jenkins CI/CD 管道.
- 您的 API 必须具有 OpenAPI 规格文件。例如，您可以使用 Apicurio Studio 生成此操作。

流程

1. 编写一个工具函数来调用 3scale toolbox。下面创建一个运行 3scale toolbox 的 Kubernetes 作业：

```
#!groovy

def runToolbox(args) {
  def kubernetesJob = [
    "apiVersion": "batch/v1",
    "kind": "Job",
    "metadata": [
      "name": "toolbox"
    ],
    "spec": [
      "backoffLimit": 0,
      "activeDeadlineSeconds": 300,
```

```
      "template": [
        "spec": [
          "restartPolicy": "Never",
          "containers": [
            [
              "name": "job",
              "image": "registry.redhat.io/3scale-amp2/toolbox-rhel7:3scale2.14",
              "imagePullPolicy": "Always",
              "args": [ "3scale", "version" ],
              "env": [
                [ "name": "HOME", "value": "/config" ]
              ],
              "volumeMounts": [
                [ "mountPath": "/config", "name": "toolbox-config" ],
                [ "mountPath": "/artifacts", "name": "artifacts" ]
              ]
            ]
          ],
          "volumes": [
            [ "name": "toolbox-config", "secret": [ "secretName": "3scale-toolbox" ] ],
            [ "name": "artifacts", "configMap": [ "name": "openapi" ] ]
          ]
        ]
      ]
    ]

    kubernetesJob.spec.template.spec.containers[0].args = args

    sh "rm -f -- job.yaml"
    writeYaml file: "job.yaml", data: kubernetesJob

    sh """set -e
    oc delete job toolbox --ignore-not-found
    sleep 2
    oc create -f job.yaml
    sleep 20 # Adjust the sleep duration to your server velocity
    """

    def logs = sh(script: "set -e; oc logs -f job/toolbox", returnStdout: true)
    echo logs
    return logs
}
```

Kubernetes 对象模板

此功能使用 Kubernetes 对象模板来运行 3scale toolbox，您可以根据您的需要进行调整。它设置 3scale toolbox CLI 参数，并将生成的 Kubernetes 作业定义写入 YAML 文件，清理之前运行 toolbox，创建 Kubernetes 作业并等待：

- 您可以将等待持续时间调整到服务器速度，以匹配 Pod 在 **Created** 和 **Running** 状态之间过渡所需的时间。您可以使用轮询循环来优化此步骤。
- OpenAPI 规格文件从名为 **openapi** 的 **ConfigMap** 获取。

- 3scale 管理门户主机名和访问令牌是从名为 **3scale-toolbox** 的 secret 获取的，如 安装 3scale API 管理 toolbox 并启用访问 所示。
- **ConfigMap** 将由管道在第 3 步中创建。但是，该 secret 已在管道外部置备，并受基于角色的访问控制(RBAC)的约束，以提高安全性。

2. 在 Jenkins 管道阶段定义要用于 3scale toolbox 的全局环境变量。例如：

3scale 托管

```
def targetSystemName = "saas-apikey-usecase"
def targetInstance = "3scale-saas"
def privateBaseURL = "http://echo-api.3scale.net"
def testUserKey = "abcdef1234567890"
def developerAccountId = "john"
```

3scale On-premises

使用自我管理的 APIcast 或 3scale 内部安装时，您必须再声明两个变量：

```
def publicStagingBaseURL = "http://my-staging-api.example.test"
def publicProductionBaseURL = "http://my-production-api.example.test"
```

这些变量如下所述：

- **targetSystemName** ：要创建的服务名称。
- **targetInstance** ：这与 安装 3scale API 管理 toolbox 时创建的 3scale 远程实例的名称匹配，并启用访问。
- **privateBaseURL** ：API 后端的端点主机。
- **testUserKey** ：用于运行集成测试的用户 API 密钥。可以像所示一样硬编码，或者从 HMAC 功能生成。
- **developerAccountId** ：将在其中创建测试应用程序的目标帐户的 ID。
- **publicStagingBaseURL** ：要创建的服务的公共暂存基本 URL。
- **publicProductionBaseURL** ：要创建的服务的公共生产基本 URL。

3. 添加 pipeline 阶段来获取 OpenAPI 规格文件，并将它作为 **ConfigMap** 在 OpenShift 中置备，如下所示：

```
node() {
 stage("Fetch OpenAPI") {
   sh """set -e
   curl -sfk -o swagger.json https://raw.githubusercontent.com/microcks/api-
lifecycle/master/beer-catalog-demo/api-contracts/beer-catalog-api-swagger.json
   oc delete configmap openapi --ignore-not-found
   oc create configmap openapi --from-file="swagger.json"
   """
 }
```

4. 添加一个使用 3scale toolbox 将 API 导入到 3scale 的管道阶段：

3scale 托管

3scale 托管

```
stage("Import OpenAPI") {
  runToolbox([ "3scale", "import", "openapi", "-d", targetInstance, "/artifacts/swagger.json", "--override-private-base-url=${privateBaseURL}", "-t", targetSystemName ])
}
```

3scale On-premises

使用自我管理的 APIcast 或 3scale 内部安装时，还必须指定公共暂存和生产基本 URL 的选项：

```
stage("Import OpenAPI") {
  runToolbox([ "3scale", "import", "openapi", "-d", targetInstance, "/artifacts/swagger.json", "--override-private-base-url=${privateBaseURL}", "-t", targetSystemName, "--production-public-base-url=${publicProductionBaseURL}", "--staging-public-base-url=${publicStagingBaseURL}" ])
}
```

5. 添加使用 toolbox 创建 3scale 应用程序计划和应用程序的管道阶段：

```
stage("Create an Application Plan") {
  runToolbox([ "3scale", "application-plan", "apply", targetInstance, targetSystemName, "test", "-n", "Test Plan", "--default" ])
}

stage("Create an Application") {
  runToolbox([ "3scale", "application", "apply", targetInstance, testUserKey, "--account=${developerAccountId}", "--name=Test Application", "--description=Created by Jenkins", "--plan=test", "--service=${targetSystemName}" ])
}
```

```
stage("Run integration tests") {
  def proxyDefinition = runToolbox([ "3scale", "proxy", "show", targetInstance, targetSystemName, "sandbox" ])
  def proxy = readJSON text: proxyDefinition
  proxy = proxy.content.proxy

  sh """set -e
  echo "Public Staging Base URL is ${proxy.sandbox_endpoint}"
  echo "userkey is ${testUserKey}"
  curl -vfk ${proxy.sandbox_endpoint}/beer -H 'api-key: ${testUserKey}'
  curl -vfk ${proxy.sandbox_endpoint}/beer/Weissbier -H 'api-key: ${testUserKey}'
  curl -vfk ${proxy.sandbox_endpoint}/beer/findByStatus/available -H 'api-key: ${testUserKey}'
  """
}
```

6. 添加使用 toolbox 将 API 提升到您的生产环境的阶段。

```
stage("Promote to production") {
  runToolbox([ "3scale", "proxy", "promote", targetInstance,  targetSystemName ])
}
```

其他资源

- 使用 Jenkinsfile 创建管道
- 3scale API 管理 toolbox

第 8 章 使用 3SCALE API MANAGEMENT 操作器配置和调配 3SCALE

作为 Red Hat 3scale API Management 管理员，您可以使用 3scale 操作器配置 3scale 服务并调配 3scale 资源。您可以在 OpenShift Container Platform(OCP)用户界面中使用 Operator。使用 Operator 是在管理门户中或使用 3scale 内部 API 配置和调配 3scale 的替代选择。

当您使用 3scale 操作器配置服务或置备资源时，更新该服务或资源的唯一方法是更新其自定义资源 (CR)。

注意

在管理门户中，服务和资源是可见的，但不能在那里进行更新。同样，不要使用内部 3scale API 进行更新来更新服务和资源。使用 CR 以外的方法进行更新将导致 Operator 恢复更改，使配置保持不变。

本章详细介绍了操作器应用程序功能的工作原理以及如何使用操作器部署自定义资源：

- 您的第一个 3scale API 管理产品和后端
- 提升产品的 APIcast 配置
- 与功能相关的后端自定义资源
- 与功能相关的产品自定义资源
- 租户自定义资源
- 开发人员帐户自定义资源

另外，还有有关使用 3scale 操作器时的能力限制的信息。

8.1. 一般先决条件

要使用 3scale 操作器配置和置备 3scale，需要以下元素：

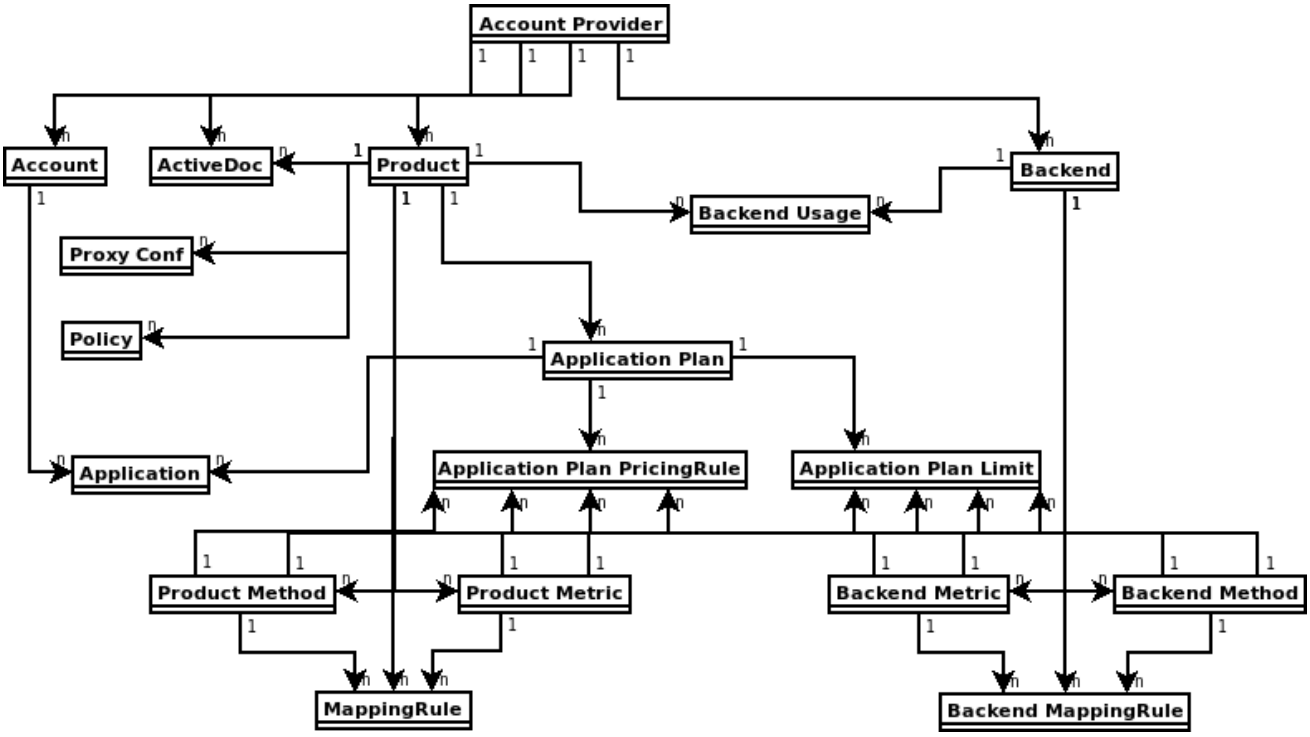
- 具有 3scale API Management 2.14 On-Premises 实例的管理员特权的用户帐户
- 安装了 3scale API 管理操作器。
- OpenShift Container Platform 4，具有在 OpenShift 集群中具有管理员特权的用户帐户。
  - 有关支持配置的更多信息，请参阅 Red Hat 3scale API 管理支持的配置 页面。

8.2. 通过 3SCALE API 管理 OPERATOR 的应用程序功能

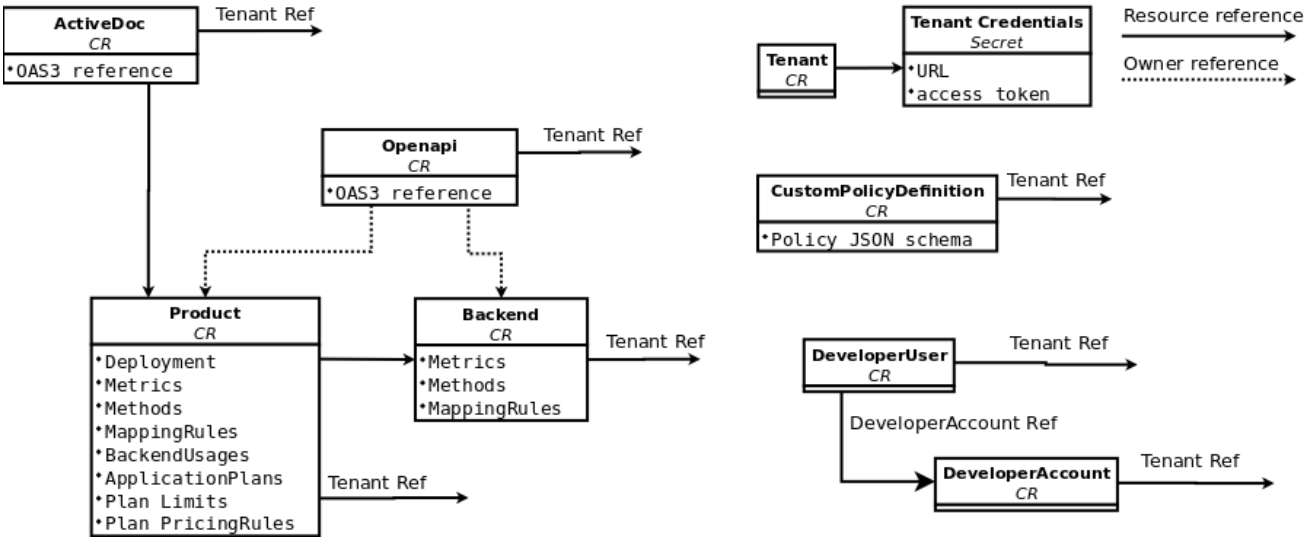
3scale 操作器包含这些特性：

- 允许与红帽 3scale API 管理解决方案交互。
- 使用 OpenShift 中的自定义资源，以声明方式管理 3scale 应用。

下图显示了 3scale 实体和关系，它们有资格以声明性方式使用 OpenShift 自定义资源进行管理。产品包含一个或多个后端。在产品层面上，您可以配置应用程序、应用程序计划以及映射规则。在后端，您可以为每个后端设置指标、方法和映射规则。



3scale 操作器提供自定义资源定义及其关系，如下图中所示。



8.3. 部署第一个 3SCALE API 管理产品和后端

在新创建的租户中使用 OpenShift Container Platform (OCP)，您将以最低配置部署第一个 3scale 产品和后端。

先决条件

常规先决条件 中列出的安装要求与以下事项相同：

- 3scale 帐户可以在正常工作的 OpenShift 命名空间或远程安装中是本地的。
- 此帐户中所需的参数是 3scale 管理门户 URL 地址和访问令牌。

流程

1. 使用 3scale 管理门户中的凭据，为 3scale 提供程序帐户创建一个机密。例如：**adminURL=https://3scale-admin.example.com** 和 **token=123456**。

```
$ oc create secret generic threescale-provider-account --from-
literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
```

2. 使用上游 API URL 配置 3scale 后端：

   a. 使用以下内容创建 YAML 文件：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend1
spec:
  name: "Operated Backend 1"
  systemName: "backend1"
  privateBaseURL: "https://api.example.com"
```

   - 创建该文件后，操作员将确认步骤是否成功。
   - 有关 Backend 自定义资源(CR)和可能值字段的详情，请参阅 后端自定义资源定义(CRD)参考。

   b. 创建自定义资源：

```
$ oc create -f backend1.yaml
```

3. 配置 3scale 产品：

   a. 使用应用到之前创建的后端的所有默认设置创建产品：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  systemName: "operatedproduct1"
  backendUsages:
    backend1:
      path: /
```

   - 创建该文件后，操作员将确认步骤是否成功。
   - 有关 Product CR 和可能值字段的详情，请参阅 产品 CRD 参考。

   b. 创建自定义资源：

```
$ oc create -f product1.yaml
```

   另外，您可以更新现有产品 CRD 以链接到后端：

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

4. 创建的自定义资源将需要几秒钟时间来填充 3scale 实例。要确认何时同步资源，您可以选择以下方法之一：

   - 验证对象的 *status* 字段。
   - 使用 **oc wait** 命令：

```
$ oc wait --for=condition=Synced --timeout=-1s backend/backend1

$ oc wait --for=condition=Synced --timeout=-1s product/product1
```

8.4. 提升产品的 APICAST 配置

使用 3scale operator，您可以将产品的 APIcast 配置提升到 staging 或 production。**ProxyConfigPromote** 自定义资源 (CR)将最新的 APIcast 配置提升到 stage 环境。另外，您还可以配置 **ProxyConfigPromote** CR 以提升到生产环境。

注意

ProxyConfigPromote 对象仅在创建时生效。创建后，它们上的任何更新都不会协调。

先决条件

安装的要求与常规先决条件中列出的相同， 包括：

- 已创建了一个 产品 CR。

流程

1. 使用以下内容创建一个 YAML 文件：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
```

要将 APIcast 配置提升到生产环境，请将可选字段 **spec.production** 设置为 **true** ：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
  production: true
```

要在成功提升后删除 **ProxyConfigPromote** 对象，将可选字段 **spec.deleteCR** 设置为 **true** ：

```
apiVersion: capabilities.3scale.net/v1beta1
```

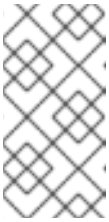
```
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
  deleteCR: true
```

2. 要检查文件的状态条件，请输入以下命令：

```
oc get proxyconfigpromote proxyconfigpromote-sample -o yaml
```

a. 输出应该显示状态为 **Ready** ：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ProxyConfigPromote
metadata:
  name: proxyconfigpromote-sample
spec:
  productCRName: product1-sample
status:
  conditions:
  - lastTransitionTime: "2022-10-28T11:35:19Z"
    status: "True"
    type: Ready
```



注意

如果您没有在代理配置中进行更改，您会收到 ProxyConfigPromote CR 的 **Failed** 输出状态，其中包含以下信息：**无法提升生产环境，因为没有检测到产品更改，删除 proxyConfigPromote CR，或首先向 stage env 引入更改以继续。**按照这些说明完成此流程。

3. 创建自定义资源：

```
oc create -f proxyconfigpromote-sample.yaml
```

- 对于给定示例，输出为：

```
proxyconfigpromote.capabilities.3scale.net/proxyconfigpromote-sample created
```

其他资源

- ProxyConfigPromote CRD 参考

8.5. 3SCALE API MANAGEMENT 操作器如何标识自定义资源链接的租户

您可以部署 3scale 自定义资源(CR)来管理各种 3scale 对象。一个 3scale CR 链接到一个租户。

如果 3scale Operator 安装在与 3scale 相同的命名空间中，则默认的行为是 3scale CR 链接到该 3scale 实例的默认租户。要将 3scale CR 链接到不同的租户，您可以执行以下操作之一：

- 在包含 3scale CR 的命名空间中创建 **threescale-provider-account** secret。部署 3scale CR 时，Operator 会读取此 secret 来标识 CR 链接的租户。要使 Operator 使用此 secret，需要满足以下条件之一：
  - 3scale CR 将 **spec.providerAccountRef** 字段指定为 null。
  - 3scale CR 省略 **spec.providerAccountRef** 字段。
    threescale-provider-account 机密标识 CR 链接的租户。secret 必须包含对 3scale 实例的引用，格式为 URL 和凭据，以便以令牌的形式访问该 3scale 实例中的租户。例如：

    ```
    $ oc create secret generic threescale-provider-account --from-literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
    ```

    threescale-provider-account 机密可以标识任何 3scale 实例中的任何租户，只要 HTTP 连接可用。换句话说，3scale CR 和 3scale 实例，其中包含 CR 链接到的租户，也可以位于不同的 OpenShift 集群中。

- 在 3scale CR 中，指定 **spec.providerAccountRef**，并将它设置为对标识该租户的 OpenShift **Secret** 的本地引用的名称。在以下 3scale **DeveloperAccount** CR 示例中，**mytenant** 是 secret：

  ```
  apiVersion: capabilities.3scale.net/v1beta1
  kind: DeveloperAccount
  metadata:
    name: developeraccount-simple-sample
  spec:
    orgName: Ecorp
    providerAccountRef:
      name: mytenant
  ```

  在 secret 中：
  - **adminURL** 指定 3scale 实例的 URL，它可以位于任何命名空间中。
  - **token** 指定用于访问 3scale 实例中的一个租户的凭据。此租户可以是默认租户，也可以是该实例中的任何其他租户。
    通常，当您部署租户 CR 时，您要创建此 secret。例如：

    ```
    apiVersion: v1
    kind: Secret
    metadata:
      name: mytenant
    type: Opaque
    stringData:
      adminURL: https://my3scale-admin.example.com:443
      token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    ```

如果 3scale 操作器无法识别 CR 链接的租户，Operator 会生成错误消息。

8.6. 部署 3SCALE API MANAGEMENT OPENAPI 自定义资源

OpenAPI 自定义资源(CR)是导入 OpenAPI 规格(OAS)文档的一种方法，您可以在 Developer Portal 中用于 ActiveDocs。OAS 是一个标准，不会将您绑定到使用与 API 使用一个特定的编程语言。人工和计算机可以更轻松地了解 API 产品的功能，而无需源代码访问、文档或网络流量检查。

先决条件

- 具有 3scale 2.14 On-Premises 实例的管理员特权的用户帐户。
- 定义 API 的 OAS 文档。
- 了解 **OpenAPI** CR 如何与租户链接。

8.6.1. 部署 3scale OpenAPI 自定义资源，从 secret 中导入 OAS 文档

部署 **OpenAPI** 自定义资源 (CR)，以便您可以创建 3scale 后端和产品。

注意

Operator 只读取 secret 中的内容。Operator 不读取 secret 中的字段名称。

先决条件

了解 3scale 操作器如何标识自定义资源链接的租户。

流程

1. 定义包含 OAS 文档的 secret。例如，您可以使用以下内容创建 **myoasdoc1.yaml** ：

```
openapi: "3.0.2"
info:
  title: "some title"
  description: "some description"
  version: "1.0.0"
paths:
  /pet:
    get:
      operationId: "getPet"
      responses:
        405:
          description: "invalid input"
```

2. 创建 secret.例如：

```
$ oc create secret generic myoasdoc1 --from-file myoasdoc1.yaml

secret/myoasdoc1 created
```

3. 定义 **OpenAPI** CR。确保指定包含 OAS 文档的 secret 的引用。例如，您可以创建 **myopenapicr1.yaml** 文件：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: OpenAPI
metadata:
  name: myopenapicr1
spec:
  openapiRef:
    secretRef:
      name: myoasdoc1
```

4. 创建您刚刚定义的资源。例如：

```
$ oc create -f myopenapicr1.yaml
```

对于给定示例，输出为：

```
$ openapi.capabilities.3scale.net/myopenapicr1 created
```

8.6.2. 3scale API Management OpenAPI 自定义资源定义的功能

了解 **OpenAPI** 自定义资源定义(CRD)部署功能将有助于您配置 3scale 产品、后端并为开发者门户创建 ActiveDocs。

- OAS 文档可以从以下项读取：
  - Kubernetes secret。
  - http 和 https 格式的 URL。
- 在 OAS 文档中，**info.title** 设置不能超过 215 个字符。Operator 使用此设置来创建 OpenShift 对象名称，其长度限制。
- 只有服务器列表中的第一个 **server[0].url** 元素将解析为私有 URL。OpenAPI 规格(OAS)使用其 **server[0].url** 元素的 **basePath** 组件。
- **OpenAPI** CRD 支持单个顶层安全要求，但它不支持操作级别的安全性。
- **OpenAPI** CRD 支持 **apiKey** 安全方案。

其他资源

- 与功能相关的产品自定义资源
- **OpenAPI** CRD 参考
- 对象名称和 ID

8.6.3. 定义 OpenAPI 自定义资源时导入规则

导入规则指定当为 3scale 部署设置 OpenAPI 文档时，OpenAPI 规格(OAS)如何与 3scale 一起工作。

产品名称

默认产品系统名称通过 OpenAPI 文档中的 **info.title** 字段获取。要覆盖 OpenAPI 文档中的产品名称，在 **OpenAPI** 自定义资源(CR)中指定 **spec.productSystemName** 字段。

私有基本 URL

私有基础 URL 从 **OpenAPI** CR **servers[0].url** 字段读取。您可以使用 **OpenAPI** CR 中的 **spec.privateBaseURL** 字段来覆盖它。

3scale 方法

导入 OpenAPI 文档中定义的每个操作都会转换为产品级别的一个 3scale 方法。方法名称从操作对象的 **operationId** 字段读取。

3scale 映射规则

导入 OpenAPI 文档中定义的每个操作都会转换为产品级别的 3scale 映射规则。以前，现有映射规则由使用 **OpenAPI** CR 导入的用户替代。

在 OpenAPI 文档中，**paths** 对象提供了用于操作动词和模式属性的映射规则。3scale 方法相应地与 **operationId** 关联。

delta 值硬编码为 **1**。

默认情况下配置*严格匹配*策略。可以使用 **OpenAPI** CRD 的 **spec.PrefixMatching** 字段切换到*前缀匹配*。

身份验证

只支持一个顶级安全要求。不支持操作级别的安全要求。

支持的安全方案是 **apiKey**。

apiKey 安全方案类型：

- *凭证位置* 将从安全方案对象的 **in** 字段中的 OpenAPI 文档读取。
- *Auth user* 键将从安全方案对象的 OpenAPI 文档的 **name** 字段中读取。

以下是 OAS 3.0.2 部分带有 **apiKey** 安全要求的部分示例：

```
openapi: "3.0.2"
security:
  - petstore_api_key: []
components:
  securitySchemes:
    petstore_api_key:
      type: apiKey
      name: api_key
      in: header
```

当 OpenAPI 文档没有指定任何安全要求时，会应用以下内容：

- 产品身份验证将为 **apiKey** 配置。
- *凭证位置* 将默认为 3scale 值 **As query parameters (GET) or body parameters (POST/PUT/DELETE)**。
- *Auth user* 密钥默认为 3scale 值 **user_key**。

3scale *Authentication Security* 可以使用 **OpenAPI** CRD 的 **spec.privateAPIHostHeader** 和 **spec.privateAPISecretToken** 字段来设置。

ActiveDocs

不会创建 3scale ActiveDoc。

3scale 产品策略链

3scale 策略链是默认 3scale 创建的。

3scale 部署模式

默认情况下，配置的 3scale 部署模式将为 APIcast 3scale 管理。但是，当 **spec.productionPublicBaseURL** 或 **spec.stagingPublicBaseURL** 或两个字段都位于 **OpenAPI** CR 中时，产品的部署模式是 APIcast 自我管理的。

带有自定义公共基础 URL 的 **OpenAPI** CR 示例：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: OpenAPI
metadata:
  name: openapi1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-
Specification/master/examples/v3.0/petstore.yaml"
  productionPublicBaseURL: "https://production.my-gateway.example.com"
  stagingPublicBaseURL: "https://staging.my-gateway.example.com"
```

8.6.4. 部署 3scale API Management OpenAPI 自定义资源，从 URL 中导入 OAS 文档

您可以部署从您指定的 URL 中导入 OAS 文档的 **OpenAPI** 自定义资源。然后，您可以在 Developer Portal 中使用此 OAS 文档作为您的 API 的 ActiveDocs 的基础。

先决条件

- 如果您要创建一个 **OpenAPI** 自定义资源，它没有链接到同一命名空间中的 3scale 实例中默认租户，则该命名空间将包含 **OpenAPI** CR 的 secret 包含一个用于标识 **OpenAPI** CR 链接的租户。secret 的名称是以下之一：
  - **threescale-provider-account**
  - 用户定义

  此机密包含 3scale 实例的 URL 和令牌，其中包含用于访问 3scale 实例中某一租户的凭据。

流程

1. 在 OpenShift 帐户中，进入到 **Operators > Installed operators**。
2. 点 3scale operator。
3. 选择 *YAML* 选项卡。
4. 创建 **OpenAPI** 自定义资源(CR)。例如：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: OpenAPI
metadata:
  name: openapi1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-
Specification/master/examples/v3.0/petstore.yaml"
  providerAccountRef:
    name: mytenant
```

5. 点击 **Save**。3scale 操作器需要几秒钟时间来创建 **OpenAPI** CR。

验证

1. 在 OpenShift 中，在 **3scale Product Overview** 页面中，确认 *Synced* 条件被标记为 **True**。
2. 进入 3scale 帐户。
3. 确认存在 OAS 文档。对于上例，您会看到一个名为 **openapi1** 的新 OAS 文档。

8.6.5. 其他资源

- OpenAPI 规格
- OpenAPI CRD 参考
- 部署可选租户自定义资源

8.7. 部署 3SCALE API 管理 ACTIVEDOC 自定义资源

Red Hat 3scale API Management ActiveDocs 基于 API 定义文档，用于定义符合 OpenAPI 规格的 RESTful Web 服务。**ActiveDoc** 自定义资源(CR)是导入 OpenAPI 规格(OAS)文档的一种方法，您可以在 Developer Portal 中用于 ActiveDocs。OAS 是一个标准，不会将您绑定到使用与 API 使用一个特定的编程语言。人工和计算机可以更轻松地了解 API 产品的功能，而无需源代码访问、文档或网络流量检查。

先决条件

- 具有 3scale 2.14 On-Premises 实例的管理员特权的用户帐户。
- 定义 API 的 OAS 文档。
- 了解 **ActiveDoc** CR 如何链接到租户。

8.7.1. 部署 3scale API 管理 ActiveDoc 自定义资源，从 secret 中导入 OAS 文档

部署 **ActiveDoc** 自定义资源 (CR)，以便您可以创建 3scale 后端和产品。

注意

Operator 只读取 secret 中的内容。Operator 不读取 secret 中的字段名称。例如，数据采用 **key: value** 对结构，其中 **value** 代表文件的内容，**key** 是文件名。Operator 在 ActiveDoc CRD 的上下文中会忽略文件名。Operator 只读取文件的内容。

先决条件

- 您可以了解 3scale API 管理操作器如何标识自定义资源链接到 的租户。
- 定义包含 OAS (OpenAPI 规格) 文档的 secret。例如，您可以使用以下内容创建 **myoasdoc1.yaml** ：

```
openapi: "3.0.2"
info:
  title: "some title"
  description: "some description"
```

```
  version: "1.0.0"
paths:
  /pet:
    get:
      operationId: "getPet"
      responses:
        405:
          description: "invalid input"
```

流程

1. 创建 secret.例如：

```
$ oc create secret generic myoasdoc1 --from-file myoasdoc1.yaml

secret/myoasdoc1 created
```

2. 定义您的 **ActiveDoc** CR。确保指定包含 OAS 文档的 secret 的引用。例如，您可以创建 **myactivedoccr1.yaml** 文件：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ActiveDoc
metadata:
  name: myactivedoccr1
spec:
  name: "Operated ActiveDoc From secret"
  activeDocOpenAPIRef:
    secretRef:
      name: myoasdoc1
```

3. 创建您刚刚定义的资源。例如：

```
$ oc create -f myactivedoccr1.yaml
```

对于给定示例，输出为：

```
$ activedoc.capabilities.3scale.net/myactivedoccr1 created
```

验证

1. 登录到您的 Red Hat OpenShift Container Platform (OCP) 管理员帐户。
2. 进入 **Operators > Installed Operators**。
3. 点 *Red Hat Integration - 3scale* 。
4. 点 *Active Doc* 选项卡。
5. 确认存在 OAS 文档。对于上例，您会看到一个名为 **myactivedoccr1** 的新 OAS 文档。

8.7.2. 3scale API 管理 ActiveDoc 自定义资源定义的功能

ActiveDoc 自定义资源定义 (CRD) 与开发人员 **OpenAPI** 文档格式的产品文档相关。了解 **ActiveDoc** CRD 部署功能可帮助您为开发人员门户创建 ActiveDocs。

- **ActiveDoc** CR，可从以下任一位置读取 OpenAPI 文档：
  - 机密.
  - **http** 或 **https** 格式的 URL
- 另外，您可以使用 **productSystemName** 字段将 **ActiveDoc** CR 与 3scale 产品相关联。该值必须是 3scale 产品的 CR 的 **system_name**。
- 您可以使用 **published** 字段发布或隐藏 3scale 中的 **ActiveDoc** 文档。默认情况下，这设置会被**隐藏**。
- 您可以使用 **skipSwaggerValidations** 字段来跳过 OpenAPI 3.0 验证。默认情况下， **ActiveDoc** CR 会被验证。

其他资源

- 与功能相关的产品自定义资源
- **ActiveDoc** CRD 参考

8.7.3. 部署 3scale API 管理 ActiveDoc 自定义资源，从 URL 中导入 OAS 文档

您可以部署一个 **ActiveDoc** 自定义资源 (CR)，从您指定的 URL 中导入 OAS (OpenAPI 规格) 文档。然后，您可以在 Developer Portal 中使用此 OAS 文档作为您的 API 的 ActiveDocs 的基础。

先决条件

- 您可以了解 3scale API 管理操作器如何标识自定义资源链接到 的租户。

流程

1. 在 OpenShift 帐户中，进入到 **Operators > Installed operators**。
2. 点 3scale operator。
3. 点 *Active Doc* 选项卡。
4. 创建 **ActiveDoc** CR。例如：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: ActiveDoc
metadata:
  name: myactivedoccr1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-
Specification/master/examples/v3.0/petstore.yaml"
  providerAccountRef:
    name: mytenant
```

5. 可选。对于自我管理的 APIcast，在 **ActiveDoc** CR 中，将 **productionPublicBaseURL** 和 **stagingPublicBaseURL** 字段设置为部署的 URL。例如：

```
apiVersion: capabilities.3scale.net/v1beta1
```

```
kind: ActiveDoc
metadata:
  name: myactivedoccr1
spec:
  openapiRef:
    url: "https://raw.githubusercontent.com/OAI/OpenAPI-
Specification/master/examples/v3.0/petstore.yaml"
  productionPublicBaseURL: "https://production.my-gateway.example.com"
  stagingPublicBaseURL: "https://staging.my-gateway.example.com"
```

6. 点击 **Save**。3scale operator 需要几秒钟时间来创建 **ActiveDoc** CR。

验证

1. 登录到您的 Red Hat OpenShift Container Platform (OCP) 管理员帐户。
2. 进入 **Operators > Installed Operators**。
3. 点 *Red Hat Integration 3scale* 。
4. 点 *Active Doc* 选项卡。
5. 确认存在 OAS 文档。对于上例，您会看到一个名为 **myactivedoccr1** 的新 OAS 文档。

8.7.4. 其他资源

- OpenAPI 规格
- **ActiveDoc** CRD 参考
- 部署可选租户自定义资源

8.8. 与功能相关的后端自定义资源

在新创建的租户中使用 Openshift Container Platform，您将配置后端、对应的指标、方法和映射规则。您还将了解 backend 自定义资源的状态，以及如何将后端链接到租户帐户。

先决条件

常规先决条件 中列出的安装要求与以下事项相同：

- 3scale 帐户中所需的最小参数是管理门户 URL 地址和访问令牌。

8.8.1. 部署与功能相关的后端自定义资源

在新创建的租户中使用 OpenShift Container Platform (OCP)，您将配置新的后端。

流程

1. 在 OpenShift 帐户中，导航到 **Installed operator**。
2. 单击 3scale 操作器。
3. 在 **3scale Backend 下**，点 *Create Instance*。

4. 选择 YAML 视图。

5. 创建指向特定 3scale Admin URL 地址的 3scale 后端：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: <your_backend_OpenShift_name>
spec:
  name: "<your_backend_name>"
  privateBaseURL: "<your_admin_portal_URL>"
```

例如：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
```

6. 要保存您的更改，请点击 **Create**。

7. 等待几秒钟，以在 OpenShift 和 3scale 帐户中创建后端。然后，您可以执行以下操作：

   a. 在 **3scale Backend Overview** 页面中检查 *Synced* 条件标记为 **True**，确认 OpenShift 中已创建了后端。

   b. 进入 3scale 帐户，您会看到已创建了后端。在上例中，您将看到一个名为 **My Backend Name** 的新后端。

8.8.2. 定义后端指标

将 Openshift Container Platform (OCP)与新创建的 3scale 租户一起使用，在后端自定义资源(CR)中定义所需的后端指标。

考虑以下观察：

- **metrics** 映射键名称将用作 **system_name**。在以下示例中： **metric01**、**metric02** 和 **hits**。
- 在所有指标和方法中，指标 **metrics** 键名称都必须是唯一的。
- **unit** 和 **friendlyName** 是必填字段。
- 如果没有添加 **hits** 指标，Operator 将创建这个指标。

流程

- 在新的 3scale 后端中添加后端指标，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
```

```
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  metrics:
    metric01:
      friendlyName: Metric01
      unit: "1"
    metric02:
      friendlyName: Metric02
      unit: "1"
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit
```

8.8.3. 定义后端方法

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，在后端自定义资源(CR)中定义所需的后端方法。

考虑以下观察：

- 映射键名称的**methods** 将用作 **system_name**。在以下示例中： **Method01** 和 **Method02**。
- 在所有指标和方法中，**methods** 映射键名称必须是唯一的。
- **friendlyName** 是一个必填字段。

流程

- 在新的 3scale 后端中添加后端方法，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  methods:
    method01:
      friendlyName: Method01
    method02:
      friendlyName: Method02
```

8.8.4. 定义后端映射规则

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，在后端自定义资源(CR)中定义所需的后端映射规则。

考虑以下观察：

- **httpMethod**, **pattern**, **increment** 和 **metricMethodRef** 是必填字段。

- **metricMethodRef** 包含对现有指标或方法映射键名称 **system_name** 的引用。在以下示例中，**hits**。

流程

- 将后端映射规则添加到新的 3scale 后端，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  mappingRules:
    - httpMethod: GET
      pattern: "/pets"
      increment: 1
      metricMethodRef: hits
    - httpMethod: GET
      pattern: "/pets/id"
      increment: 1
      metricMethodRef: hits
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
```

8.8.5. backend 自定义资源的状态

status 字段显示对最终用户有用的资源信息。它不应该手动更新，它会与资源的每个更改同步。

这些是 *status* 字段的属性：

- **backendId**
  3scale 后端的内部标识符。

- **conditions**
  表示 **status.Conditions** Kubernetes 通用模式。它有这些类型或状态：

  - **无效**

    不支持 **BackendSpec** 中的配置组合。这不是临时错误，但表示在进行进度前必须修复的状态。

  - **同步**

    后端已成功同步。

  - **Failed**

    同步过程中出现错误。

- **observedGeneration**
  这是一个帮助字段，用于确认状态信息已更新为最新的资源规格。

同步资源示例：

```
status:
  backendId: 59978
  conditions:
  - lastTransitionTime: "2020-06-22T10:50:33Z"
    status: "False"
    type: Failed
  - lastTransitionTime: "2020-06-22T10:50:33Z"
    status: "False"
    type: Invalid
  - lastTransitionTime: "2020-06-22T10:50:33Z"
    status: "True"
    type: Synced
  observedGeneration: 2
```

8.8.6. 链接到租户帐户的后端自定义资源

当 3scale 操作器找到新的 3scale 资源时，*LookupProviderAccount* 进程从识别拥有该资源的租户开始。

进程检查租户凭据来源。如果未找到，则引发错误。

以下步骤描述了进程如何验证租户凭证源：

1. 检查 *providerAccountRef resource* 属性中的凭据。这是一个 secret 本地引用，例如 **mytenant**:

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Backend
metadata:
  name: backend-1
spec:
  name: "My Backend Name"
  privateBaseURL: "https://api.example.com"
  providerAccountRef:
    name: mytenant
```

mytenant secret 必须具有填充租户凭据的 *adminURL* 和 *token* 字段。例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: mytenant
type: Opaque
stringData:
  adminURL: https://my3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

2. 检查默认的 **threescale-provider-account** 机密。例如： **adminURL=https://3scale-admin.example.com** 和 **token=123456**:

```
$ oc create secret generic threescale-provider-account --from-
literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
```

3. 在 3scale 部署的相同命名空间中检查默认供应商账户：如果 3scale 安装位于与自定义资源相同的命名空间中，Operator 将自动为默认的 3scale 租户（provider 帐户）收集所需的凭证。

8.8.7. 删除后端自定义资源

您可以通过删除管理它的**后端**自定义资源 (CR) 来删除后端实体。当您删除 **后端** CR 时，3scale Operator 会更新部署的 **Product** CR，该 CR 引用已删除的后端。更新在这些属性中：

- **backendUsages**
- **applicationPlans**

这些属性不再引用已删除的后端。

重要

删除**后端** CR 定义的唯一方法是按照此处所述的步骤进行操作。不要使用 Admin Portal 或 3scale API 来删除部署为 CR 的后端。

先决条件

- 3scale 管理员权限或具有删除命名空间中具有删除权限的 OpenShift 角色，其中包含您要删除的 **后端** CR。要识别谁可以删除特定 **的后端** CR，请运行 **oc policy who-can delete** 命令。例如，如果 CR 中的名称是 **mybackend**，则运行这个命令：

```
$ oc policy who-can delete product.capabilities.3scale.net/mybackend
```

- 要删除至有效租户的**后端** CR 的链接。

流程

- 运行 **oc delete** 命令删除 **后端** CR。例如，如果您部署了 **mybackend.yaml** 文件中定义的**后端**，则可运行以下命令：

```
$ oc delete -f mybackend.yaml
```

或者，您可以运行 **oc delete** 命令并指定其定义中指定的后端名称。例如：

```
$ oc delete backend.capabilities.3scale.net/mybackend
```

8.9. 与功能相关的产品自定义资源

在新创建的租户中使用 OpenShift Container Platform (OCP)，您将配置产品及其相应的指标、方法、应用程序计划和映射规则，以及定义产品后端使用情况，并将您的产品链接到您的租户帐户。

先决条件

常规先决条件 中列出的安装要求与以下事项相同：

- 3scale 帐户中的最低必要参数是产品名称。

8.9.1. 部署与功能相关的产品自定义资源

在新创建的租户中使用 OpenShift Container Platform (OCP)，您将配置新产品。

8.9.1.1. 部署基本产品自定义资源

流程

1. 在 OpenShift 帐户中，导航到 **Installed operator**。
2. 单击 3scale 操作器。
3. 在 **3scale 产品** 下，单击 *Create Instance*。
4. 选择 YAML 视图。
5. 创建 3scale 产品：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: <your_product_OpenShift_name>
spec:
  name: "<your_product_name>"
```

例如：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
```

6. 要保存您的更改，请点击 **Create**。
7. 等待几秒钟，让产品在 OpenShift 和 3scale 帐户中创建。然后，您可以执行以下操作：
   a. 在 **3scale 产品概述** 页面中检查 *Synced* 条件标记为 **True**，以确认该产品已在 OpenShift 中创建。
   b. 转至 3scale 帐户，您会看到该产品已创建好。在上例中，您将看到一个名为 **OperatedProduct 1** 的新产品。

另外，您可以为您创建的每个产品指定 APIcast 部署模式。有两种替代方案：

- APIcast 托管
- APIcast 自我管理

8.9.1.2. 使用 APIcast 托管部署产品

使用托管 APIcast 配置您的产品：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
```

```
  name: "OperatedProduct 1"
  deployment:
    apicastHosted: {}
```

8.9.1.3. 使用 APIcast 自助管理部署产品

使用 APIcast 自我管理配置您的产品。在这种情况下，指定一个 **stagePublicBaseURL** 和 **productionPublicBaseURL**:

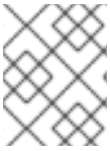
```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  deployment:
    apicastSelfManaged:
      stagingPublicBaseURL: "https://staging.api.example.com"
      productionPublicBaseURL: "https://production.api.example.com"
```

8.9.2. 定义产品应用程序计划

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，使用 **applicationPlans** 对象在产品自定义资源(CR)中定义所需的应用程序计划。

请考虑以下观察：

- **applicationPlans** map 键名称将用作 **system_name**。在以下示例中： **plan01** 和 **plan02**。



注意

setupFee 和 **costMonth** 是通用的 3scale 概念。在 3scale 用户界面中创建应用程序计划时，您必须输入这些详情。参阅使用您的定价规则配置应用程序计划。

流程

- 在新的 3scale 产品中添加应用程序计划，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  applicationPlans:
    plan01:
      name: "My Plan 01"
      setupFee: "14.56"
    plan02:
      name: "My Plan 02"
      trialPeriod: 3
      costMonth: 3
```

8.9.3. 为产品应用程序计划定义限制

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，使用 **applicationPlans.limits** 列表为您的产品应用程序计划定义所需的限制。

考虑以下观察：

- **period**、**value** 和 **metricMethodRef** 是必填字段。
- **metricMethodRef** 引用可以是产品，也可以是后端引用。使用可选的 **backend** 字段来引用 backend 指标的所有者。

流程

- 为 3scale 产品的应用程序计划定义限值，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
  applicationPlans:
    plan01:
      name: "My Plan 01"
      limits:
        - period: month
          value: 300
          metricMethodRef:
            systemName: hits
            backend: backendA
        - period: week
          value: 100
          metricMethodRef:
            systemName: hits
```

8.9.4. 定义产品应用程序计划的定价规则

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，使用 **applicationPlans.pricingRules** 列表为您的产品应用程序计划定义所需的定价规则。

考虑以下观察：

- **from**, **to**, **pricePerUnit** 和 **metricMethodRef** 是必填字段。
- **from** 和 **to** 将被验证。对于任何规则，**from** 的值小于 **to**，对同一个规则有相互重叠的范围不被允许。
- **metricMethodRef** 引用可以是产品，也可以是后端引用。使用可选的 **backend** 字段来引用 backend 指标的所有者。

流程

- 为 3scale 产品的应用计划定义定价规则，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
  applicationPlans:
    plan01:
      name: "My Plan 01"
      pricingRules:
        - from: 1
          to: 100
          pricePerUnit: "15.45"
          metricMethodRef:
            systemName: hits
        - from: 1
          to: 300
          pricePerUnit: "15.45"
          metricMethodRef:
            systemName: hits
            backend: backendA
```

8.9.5. 使用 OpenID Connect 定义产品身份验证

您可以为 3scale 产品部署 **产品自定义资源** (CR)，该产品使用 OpenID Connect (OIDC)对任何 OAuth 2.0 流进行身份验证。3scale 与 OpenID Connect 等第三方身份提供程序(IdP)集成，以验证 API 请求。有关 OpenID Connect 的更多信息，请参阅 OpenID Connect 集成。与第三方 IdP 集成后，您将有两种类型的数据与产品 CR 包括：

- **issuerType**：在与第三方 IdP 集成时，使用红帽单点登录 (RH-SSO) 以及 **rest** 值时，会使用 **keycloak** 值。
- **issuerEndpoint** ：其中包含所需凭证的 URL。

先决条件

- 您必须配置 RH-SSO。请参阅 配置红帽单点登录。
- 您必须 配置 HTTP 与第三方身份提供程序的集成。

注意

issuerEndpoint 中提供的 **CLIENT_ID** 和 **CLIENT_CREDENTIALS** 凭据必须具有足够的权限来管理域中的其他客户端。

流程

1. 确定端点 **issuerEndpoint**，它定义 OpenID 供应商的位置，并在产品 CR 中使用此格式：

```
https://<client_id>:<client_secret>@<host>:<port_number>/auth/realms/<realm_name>`
```

2. 定义或更新 3scale **产品** CR，为任何 OAuth 2.0 流指定 OpenID Connect(OIDC)身份验证。例如：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  deployment:
    <any>:
      authentication:
        oidc:
          issuerType: "keycloak"
          issuerEndpoint:
"https://myclientid:myclientsecret@mykeycloack.example.com/auth/realms/myrealm"
          authenticationFlow:
            standardFlowEnabled: false
            implicitFlowEnabled: true
            serviceAccountsEnabled: true
            directAccessGrantsEnabled: true
          jwtClaimWithClientID: "azp"
          jwtClaimWithClientIDType: "plain"
```

3. 创建您刚刚定义的资源。例如：

```
$ oc create -f product1.yaml
```

对于给定示例，输出为：

```
$ product.capabilities.3scale.net/product1 created
```

其他资源

- 产品 CRD 参考

8.9.6. 定义产品指标

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，使用 **metrics** 对象在产品自定义资源(CR)中定义所需的指标。

考虑以下观察：

- **metrics** 映射键名称将用作 **system_name**。在以下示例中： **metric01** 和 **hits**。
- 在所有指标和方法中，指标 **metrics** 键名称都必须是唯一的。
- **unit** 和 **friendlyName** 是必填字段。
- 如果没有添加 **hits** 指标，它将由操作器创建。

流程

- 在新的 3scale 后端中添加产品指标，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    metric01:
      friendlyName: Metric01
      unit: "1"
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
```

8.9.7. 定义产品方法

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，使用 method 对象在产品自定义资源中定义所需的 **方法**。

考虑以下观察：

- 映射键名称的**methods** 将用作 **system_name**。在以下示例中： **Method01** 和 **Method02**。
- 在所有指标和方法中，**methods** 映射键名称必须是唯一的。
- **friendlyName** 是一个必填字段。

流程

- 在新的 3scale 产品中添加方法，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  methods:
    method01:
      friendlyName: Method01
    method02:
      friendlyName: Method02
```

8.9.8. 定义产品映射规则

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，使用 **mappingRules** 对象在产品自定义资源(CR)中定义所需的映射规则。

考虑以下观察：

- **httpMethod**, **pattern**, **increment** 和 **metricMethodRef** 是必填字段。
- **metricMethodRef** 包含对现有指标或方法映射键名称 **system_name** 的引用。在以下示例中，**hits**。

流程

- 将产品映射规则添加到新的 3scale 后端，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  metrics:
    hits:
      description: Number of API hits
      friendlyName: Hits
      unit: "hit"
  methods:
    method01:
      friendlyName: Method01
  mappingRules:
    - httpMethod: GET
      pattern: "/pets"
      increment: 1
      metricMethodRef: hits
    - httpMethod: GET
      pattern: "/cars"
      increment: 1
      metricMethodRef: method01
```

8.9.9. 定义产品后端使用情况

将 OpenShift Container Platform (OCP)与新创建的 3scale 租户一起使用，通过应用 **backendUsages** 对象来定义所需的后端以声明方式添加到产品中。

考虑以下观察：

- **path** 是必填字段。
- **backendUsages** 映射密钥名称引用后端的 **system_name**。在以下示例中：**backendA** 和 **backendB**。

流程

- 在产品中添加后端以声明性地定义其使用情况，如下例所示：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
```

```
backendUsages:
  backendA:
    path: /A
  backendB:
    path: /B
```

8.9.10. 在 3scale API 管理产品自定义资源中配置网关响应

作为 3scale 管理员，您可以配置 产品自定义资源，以指定对对该 **API** 产品 公开 API 的请求的网关响应。部署 CR 后，3scale 确保网关返回您指定的响应和错误消息。

在 **Product** CR 中， **gatewayResponse** 对象包含您希望网关返回的响应。

流程

1. 在新的或已部署的 **Product** CR 中，在 **gatewayResponse** 对象中配置一个或多个响应。以下示例显示了使用名为 **userKey** 的身份验证模式的 Apicast 托管部署的响应配置：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  deployment:
    apicastHosted:
      authentication:
        userkey:
          gatewayResponse:
            errorStatusAuthFailed: 500
            errorHeadersAuthFailed: "text/plain; charset=mycharset"
            errorAuthFailed: "My custom reponse body"
            errorStatusAuthMissing: 500
            errorHeadersAuthMissing: "text/plain; charset=mycharset"
            errorAuthMissing: "My custom reponse body"
            errorStatusNoMatch: 501
            errorHeadersNoMatch: "text/plain; charset=mycharset"
            errorNoMatch: "My custom reponse body"
            errorStatusLimitsExceeded: 502
            errorHeadersLimitsExceeded: "text/plain; charset=mycharset"
            errorLimitsExceeded: "My custom reponse body"
```

2. 部署包含网关响应的产品 CR。例如，如果您更新了 **product1.yaml** 文件，请运行以下命令：

```
$ oc create -f product1.yaml
```

对于给定示例，输出为：

```
product.capabilities.3scale.net/product1 created
```

8.9.11. 在 3scale API 管理产品自定义资源中配置策略链

作为 3scale 管理员，您可以配置 **Product** 自定义资源 (CR)以指定您要应用到该 API 产品的策略链。部署 CR 后，3scale 将配置的策略应用到产品的上游公开 API。

流程

1. 在新的或已部署的 产品 CR 中，在 **policies** 对象中配置一个或多个策略。例如：

- 使用来自值的配置

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  policies:
  - configuration:
      http_proxy: http://example.com
      https_proxy: https://example.com
    enabled: true
    name: camel
    version: builtin
  - configuration: {}
    enabled: true
    name: apicast
    version: builtin
```

- 使用 secret 的配置

```
apiVersion: v1
kind: Secret
metadata:
  name: my-config-policy
type: Opaque
stringData:
  configuration: "{\"http_proxy\":\"http://secret.com\"}"
---
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product2
spec:
  name: "OperatedProduct 2"
  policies:
  - configurationRef:
      name: my-config-policy
    enabled: true
    name: camel
    version: builtin
  - configuration: {}
    enabled: true
    name: apicast
    version: builtin
```

对于每个策略，请指定以下字段：

- 当策略没有参数时，**配置** 是一对空的大括号。当策略有参数时，请在此处指定它们。有关您需要指定的任何参数的名称，请参阅 管理 API 网关 APIcast 标准策略中的相关策略 文档。

- **启用** 是打开或关闭策略的布尔值开关。
- **name** 标识策略。这是 **Product** CR 链接的租户范围内的唯一名称。要识别策略名称，请参阅 标准策略中相关策略的文档，以更改默认的 3scale API 管理 APIcast 行为。
- **版本 内置** 有标准策略或用于自定义策略的用户定义的字符串。例如，您可以将自定义策略的版本设置为 **1.0**。
  如果 **Product** CR 没有指定 **apicast** 策略，Operator 会添加它。

  如果在管理门户中已定义了策略链，您可以运行 3scale toolbox **export** 命令以 **.yaml** 格式导出策略链。您可以将 **导出** 的输出粘贴到 **产品** CR 中。例如，如果 **api-provider-account-one** 是 3scale 供应商帐户的名称，并且 **my-api-product-one** 是您要导出的策略链的产品名称，您将运行以下命令：

```
$ 3scale policies export api-provider-account-one my-api-product-one
```

2. 部署包含策略链的**产品** CR。例如，如果您更新了 **product1.yaml** 文件，请运行以下命令：

```
$ oc create -f product1.yaml
```

对于给定示例，输出为：

```
$ product.capabilities.3scale.net/product1 created
```

8.9.12. 产品自定义资源的状态

status 字段显示对最终用户有用的资源信息。它不应手动更新，在每次资源更改时都会同步。

这些是 *status* 字段的属性：

- **productId**
  3scale 产品的内部标识符。
- **conditions**
  表示 **status.Conditions** Kubernetes 通用模式。它有这些类型或状态：
  - **Failed**

    同步过程中出现错误。操作将重试。
  - **同步**
    产品已成功同步。
  - **无效**
    无效的对象。这不是临时错误，但它报告无效规格，应该进行更改。Operator 不会重试。
  - **orphan**
    该规范引用了不存在的资源。操作器将重试。
- **observedGeneration**
  确认状态信息已使用最新的资源规格更新。
- **state**
  3scale 产品内部状态从 3scale API 中读取。

- **providerAccountHost**
  将后端同步到的 3scale 提供程序帐户 URL。

同步资源示例：

```
status:
  conditions:
  - lastTransitionTime: "2020-10-21T18:07:01Z"
    status: "False"
    type: Failed
  - lastTransitionTime: "2020-10-21T18:06:54Z"
    status: "False"
    type: Invalid
  - lastTransitionTime: "2020-10-21T18:07:01Z"
    status: "False"
    type: Orphan
  - lastTransitionTime: "2020-10-21T18:07:01Z"
    status: "True"
    type: Synced
  observedGeneration: 1
  productId: 2555417872138
  providerAccountHost: https://3scale-admin.example.com
  state: incomplete
```

8.9.13. 链接到租户帐户的产品自定义资源

当 3scale 操作器找到新的 3scale 资源时，*LookupProviderAccount* 进程从识别拥有该资源的租户开始。

进程检查租户凭据来源。如果未找到，则引发错误。

以下步骤描述了进程如何验证租户凭证源：

1. 检查 *providerAccountRef resource* 属性中的凭据。这是一个 secret 本地引用，例如 **mytenant**:

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Product
metadata:
  name: product1
spec:
  name: "OperatedProduct 1"
  providerAccountRef:
    name: mytenant
```

mytenant secret 必须具有填充租户凭据的 *adminURL* 和 *token* 字段。例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: mytenant
type: Opaque
stringData:
  adminURL: https://my3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

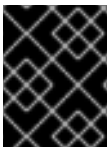
2. 检查默认的 **threescale-provider-account** 机密。例如： **adminURL=https://3scale-admin.example.com** 和 **token=123456**:

```
$ oc create secret generic threescale-provider-account --from-literal=adminURL=https://3scale-admin.example.com --from-literal=token=123456
```

3. 在 3scale 部署的相同命名空间中检查默认供应商账户：如果 3scale 安装位于与自定义资源相同的命名空间中，Operator 将自动为默认的 3scale 租户（provider 帐户）收集所需的凭证。

8.9.14. 删除产品自定义资源

您可以通过删除管理它的自定义资源(CR)来删除 3scale 产品实体。当您删除 **Product** CR 时，3scale Operator 不会更新引用已删除产品的对象。



重要

删除 **产品** CR 定义的唯一方法是按照此处所述的步骤进行操作。不要使用 Admin Portal 或 3scale API 来删除部署为 CR 的产品。

先决条件

- 3scale 管理员权限或具有删除命名空间中具有删除权限的 OpenShift 角色，其中包含您要删除的 CR。要识别谁可以删除特定 **的产品** CR，请运行 **oc policy who-can delete** 命令。例如，如果 CR 中的名称是 **myproduct**，则运行这个命令：

```
$ oc policy who-can delete product.capabilities.3scale.net/myproduct
```

- 要删除到有效租户的链接的 **Product** CR。

流程

- 运行 **oc delete** 命令删除 **产品** CR。例如，如果您部署了 **myproduct.yaml** 文件中定义的 **产品**，则可运行以下命令：

```
$ oc delete -f myproduct.yaml
```

或者，您可以运行 **oc delete** 命令并指定其定义中指定的产品名称。例如：

```
$ oc delete product.capabilities.3scale.net/myproduct
```

8.10. 与功能相关的应用程序自定义资源

在新创建的租户中使用 OpenShift Container Platform (OCP)，您将为其对应的应用程序计划配置应用程序。

先决条件

除了常规先决条件中列出的安装要求外，还有以下事项：

- 一个 3scale 开发人员帐户。
- 一个 3scale 产品。

- 一个应用程序计划。

8.10.1. 部署与功能相关的应用程序自定义资源

在新创建的租户中使用 OpenShift Container Platform (OCP)，您将配置新应用程序。

流程

1. 在 OpenShift 帐户中，进入到 **Installed operator**。
2. 点 **Red Hat Integration - 3scale**。
3. 在 **Application** 选项卡中，点 **Create application**。
4. 选择 YAML 视图。
5. 创建指向特定应用程序计划的 3scale 应用程序：
   例如：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: Application
metadata:
  name: example
  namespace: 3scale_project_name
spec:
  accountCR:
    name: developer_account
  applicationPlanName: application_plan_name
  productCR:
    name: product_custom_resource
  name: application_name
  description: describe_your_application
```

6. 要保存您的更改，请点击 **Create**。
7. 等待几秒钟，以便在 OpenShift 和 3scale 帐户中创建应用。然后，您可以执行以下操作：
   a. 检查 **3scale 应用概述** 中，确认 OpenShift 中已创建了该应用。**Ready** 条件值应该为 **True**。
   b. 进入 3scale 帐户，您会看到该应用已创建好。在上例中，您将看到一个使用您给定的唯一名称的新应用。

8.10.2. 删除应用程序自定义资源

您可以通过删除管理它的应用程序自定义资源 (CR) 来删除**应用程序**实体。

重要

删除 **Application** CR 定义的唯一方法是按照此处所述的步骤进行操作。不要使用 Admin Portal 或 3scale API 删除部署为 CR 的应用程序。

先决条件

- 3scale 管理员权限或具有删除命名空间中具有删除权限的 OpenShift 角色，其中包含您要删除的 **Application** CR。要识别谁可以删除特定的 **Application** CR，请运行 **oc policy who-can delete** 命令。例如，如果 CR 中的名称是 **myapplication**，则运行这个命令：

```
$ oc policy who-can delete application.capabilities.3scale.net/myapplication
```

- 要删除到有效租户的链接的 **Application** CR。

流程

- 运行 **oc delete** 命令删除 **Application** CR。例如，如果您部署了 **myapplication.yaml** 文件中定义的 **Application** CR，则可运行以下命令：

```
$ oc delete -f myapplication.yaml
```

或者，您可以运行 **oc delete** 命令并指定其定义中指定的应用程序名称。例如：

```
$ oc delete application.capabilities.3scale.net/myapplication
```

8.11. 部署 3SCALE API MANAGEMENT CUSTOMPOLICYDEFINITION 自定义资源

您可以使用 **CustomPolicyDefinition** 自定义资源定义(CRD)从管理门户在 3scale 产品中配置自定义策略。

当 3scale 操作器找到新的 **CustomPolicyDefinition** 自定义资源(CR)时，Operator 会标识拥有 CR 的租户，如 How the 3scale API Management operator 如何标识自定义资源链接的租户。

先决条件

- 已安装 3scale Operator。
- 您有一个自定义策略文件以供部署。
- 您已在 网关 中注入自定义策略。

流程

1. 定义 **CustomPolicyDefinition** CR，并将它保存到，如 **my-apicast-custom-policy-definition.yaml** 文件：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: CustomPolicyDefinition
metadata:
  name: custompolicydefinition-sample
spec:
  version: "0.1"
  name: "APIcast Example Policy"
  schema:
    name: "APIcast Example Policy"
    version: "0.1"
    $schema: "http://apicast.io/policy-v1/schema#manifest#"
    summary: "This is just an example."
```

```
    configuration:
      type: object
      properties: {}
```

2. 部署 **CustomPolicyDefinition** CR：

```
$ oc create -f my-apicast-custom-policy-definition.yaml
```

其他资源

- **CustomPolicyDefinition CRD 参考**

8.12. 部署租户自定义资源

租户自定义资源 (CR)也称为 *Provider Account*。

当您部署使用 3scale Operator 部署 3scale 的 **APIManager** CR 时。默认 3scale 安装包括可供使用的默认租户。另外，您可以通过部署租户 CR 来创建 其他租户。

先决条件

- 一般先决条件
- 在新的 **租户** CR 命名空间中具有 **create** 权限的 OpenShift 角色。

流程

1. 进入到安装 3scale 的 OpenShift 项目。例如，如果项目名称是 **my-3scale-project**，请运行以下命令：

```
$ oc project my-3scale-project
```

2. 创建一个包含新租户 3scale admin 帐户密码的 secret。在 **Tenant** CR 的定义中，将 **passwordCredentialsRef** 属性设置为此 secret 的名称。在第 4 步定义的 **租户** CR 示例中，**ADMIN_SECRET** 是此 secret 的占位符。以下命令提供了创建 secret 的示例：

```
$ oc create secret generic ecorp-admin-secret --from-literal=admin_password=<admin
password value>
```

3. 获取 3scale master 帐户主机名。当您使用操作器部署 3scale 时，master 帐户具有具有此模式的固定 URL：**master.${wildcardDomain}**
   如果可以访问安装 3scale 的命名空间，您可以使用这个命令获取 master 帐户主机名：

```
$ oc get routes --field-selector=spec.to.name==system-master -o jsonpath="
{.items[].spec.host}"
```

   在第 4 步中定义 **租户** CR 的示例中，**MASTER_HOSTNAME** 是此名称的占位符。

4. 创建定义新 **租户 CR 的文件**。
   在 **Tenant** CR 的定义中，将 **masterCredentialsRef.name** 属性设置为 **system-seed**。您只能使用 3scale master 帐户凭证（最好使用访问令牌）来执行租户管理任务。在部署 **APIManager** CR 时，Operator 会创建包含 master 帐户凭证的 secret。secret 的名称是 **system-seed**。

如果以 cluster wide 的形式安装了 3scale，您可以在与包含 3scale 的命名空间不同的命名空间中部署新租户。要做到这一点，将 **masterCredentialsRef.namespace** 设置为包含 3scale 安装的命名空间。

以下示例假设以 cluster wide 模式下安装 3scale。

```
apiVersion: capabilities.3scale.net/v1alpha1
kind: Tenant
metadata:
  name: ecorp-tenant
  namespace: <namespace-in-which-to-create-Tenant-CR>
spec:
  username: admin
  systemMasterUrl: https://<MASTER_HOSTNAME>
  email: admin@ecorp.com
  organizationName: ECorp
  masterCredentialsRef:
    name: system-seed
    namespace: <namespace-where-3scale-is-deployed>
  passwordCredentialsRef:
    name: <ADMIN_SECRET>
  tenantSecretRef:
    name: tenant-secret
```

5. 创建 **Tenant** CR。例如，如果您在 **mytenant.yaml** 文件中保存了前面的示例 CR，您将运行以下命令：

```
$ oc create -f mytenant.yaml
```

因此：

- Operator 通过设置 **spec.systemMasterUrl** 属性在 3scale 安装中部署一个租户。
- 3scale operator 创建一个 secret，其中包含新租户的凭证。secret 的名称是您为 **tenantSecretRef.name** 属性指定的值。此 secret 包含新的租户的 admin URL 和访问令牌。作为参考，这是 Operator 创建的 secret 示例：

```
apiVersion: v1
kind: Secret
metadata:
  name: tenant-secret
type: Opaque
stringData:
  adminURL: https://my3scale-admin.example.com:443
  token: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

- 现在，您可以部署链接到您的新组合的 **Product**, **Backend**, **OpenAPI**, **DeveloperAccount**, 和 **DeveloperUser** CR。

删除 租户 自定义资源

要删除 **部署的 租户** CR，您可以指定包含资源定义的文件名称，例如：

```
$ oc delete -f mytenant.yaml
```

另外，您可以运行 **oc delete** 命令，在 **租户** CR 中指定名称，并指定租户的命名空间。例如：

```
$ oc delete tenant.capabilities.3scale.net mytenant -n mynamespace
```

当删除租户时，3scale Operator 会进行以下操作：

- 从 3scale 安装中隐藏租户。
- 删除租户拥有的部署自定义资源。
- 标记要在 15 天后删除的租户。

删除租户后，您无法恢复它。您有 15 天时间来备份所有拥有租户的资源，并且只能在管理门户中执行此操作。15 天后，3scale 删除租户。为了遵守数据保护法律，一些数据将保留下来以备将来参考。

重要

如果您使用租户 CR 部署了该租户，请不要在管理门户中删除 **租户**。如果这样做，Operator 会将 CR 用于您要删除的租户。

其他资源

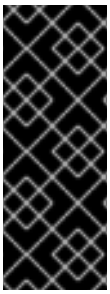
- 租户 CRD 参考

8.13. 通过部署自定义资源来管理 3SCALE API 管理开发人员

作为 3scale 管理员，您可以使用自定义资源(CR)来部署将各个开发人员用户分组在一起的开发人员帐户。通过这些帐户，您可以在开发人员门户中组织和管理开发人员对 3scale 管理的 API 的访问权限。

租户可以包含任意数量的开发人员帐户，每个开发人员帐户链接到正好一个租户。开发人员帐户可以包含任意数量的开发人员用户和每个开发人员用户链接到正好一个开发人员帐户。租户计划决定您可以创建的开发人员帐户数量以及每个开发人员帐户中可以分组多少开发人员用户。

若要使用开发人员自定义资源，3scale 操作员必须已安装 3scale。您只能在包含 3scale 操作器的命名空间中部署开发人员自定义资源。部署开发人员自定义资源是使用 3scale 管理门户或 3scale 内部 API 来管理开发人员的替代选择。



重要

通过部署自定义资源创建开发人员帐户或开发人员用户时，您无法使用管理门户或内部 3scale API 来更新这些开发人员帐户或开发人员用户。务必要注意这一点，因为在部署开发人员 CR 后，管理门户在其**帐户**页面中会显示新的开发人员帐户或新开发人员用户。如果您尝试使用管理门户或 API 更新使用 CR 部署的开发人员帐户或开发人员用户，3scale 操作器将还原更改以反映部署的 CR。这是预期在以后的发行版本中删除的限制。但是，您可以使用管理门户或 API 删除您通过部署 CR 创建的开发人员帐户或开发人员用户。

8.13.1. 先决条件

- 3scale 由 3scale 操作器安装。
- 在 **帐户管理** API 范围内具有读写权限的访问令牌，为 3scale 提供管理员特权。

8.13.2. 通过部署 DeveloperAccount 自定义资源来管理 3scale API 管理开发人员帐户

当使用 3scale 操作器安装 3scale 时，您可以部署 **DeveloperAccount** 和 **DeveloperUser** 自定义资源 (CR)。这些自定义资源允许您创建和更新开发人员对 Developer Portal 中 3scale 管理的 API 的访问帐户。

要部署新的 **DeveloperAccount** CR，还必须为具有 **admin** 角色的用户部署 **DeveloperUser** CR。此处提供的步骤是部署新的 **DeveloperAccount** CR。部署 **DeveloperAccount** CR 后，更新或删除它的过程与任何其他 CR 相同。

您只能在包含 3scale Operator 的命名空间中部署自定义资源。

先决条件

- 了解 3scale API 管理操作器如何标识自定义资源链接到 的租户。
- 如果您要创建一个 **DeveloperAccount** CR，它没有链接到同一命名空间中的 3scale 实例中的默认租户，则包含 **DeveloperAccount** CR 的命名空间包含一个 secret，用于标识 **DeveloperAccount** CR 链接的租户。secret 的名称是以下之一：
  - **threescale-provider-account**
  - 用户定义

  此机密包含 3scale 实例的 URL 和令牌，其中包含用于访问 3scale 实例中某一租户的凭据。
- 至少有一个开发人员用户的用户名、密码和电子邮件地址，该用户将在新的 **DeveloperAccount** CR 中拥有 **admin** 角色。

流程

1. 在包含 3scale 操作器的命名空间中，创建并保存资源文件，以定义一个含有开发人员帐户资源中的 **admin** 角色的用户用户名和密码的机密。例如，my **username01.yaml** 文件可能包含：

```
apiVersion: v1
kind: Secret
metadata:
  name: myusername01
stringData:
  password: "123456"
```

2. 创建 secret.例如：

```
$ oc create -f myusername01.yaml
```

对于给定示例，输出为：

```
$ secret/myusername01 created
```

3. 创建并保存一个 **.yaml** 文件，为拥有 **admin** 角色的用户定义 **DeveloperUser** CR。3scale 操作器需要此 **DeveloperUser** CR 来部署新的 **DeveloperAccount** CR。例如，**developeruser01.yaml** 文件可能包含：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: DeveloperUser
metadata:
  name: developeruser01
```

```
spec:
  username: myusername01
  email: myusername01@example.com
  passwordCredentialsRef:
    name: myusername01
  role: admin
  developerAccountRef:
    name: developeraccount1
  providerAccountRef:
    name: mytenant
```

在 **DeveloperUser** CR 中：

- 开发人员用户帐户名称、用户名和电子邮件必须在包含 **DeveloperAccount** 链接的租户中唯一。
- 您在此指定的开发人员帐户名称必须与在此流程中 **部署的 DeveloperAccount** CR 的名称匹配。在创建此 **DeveloperUser** CR 之前或之后创建 **DeveloperAccount** CR 都无关紧要。
- **DeveloperUser** CR 链接的租户必须是与指定的 **DeveloperAccount** CR 链接相同的租户。

4. 创建您刚刚定义的资源。例如：

```
$ oc create -f developeruser01.yaml
```

对于给定示例，输出为：

```
$ developeruser.capabilities.3scale.net/developeruser01 created
```

5. 创建并保存定义 **DeveloperAccount** CR 的 **.yaml** 文件。在这个 **.yaml** 文件中，**spec.OrgName** 字段必须指定一个机构名称。例如，**developeraccount01.yaml** 文件可能包含：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: DeveloperAccount
metadata:
  name: developeraccount01
spec:
  orgName: Ecorp
  providerAccountRef:
    name: mytenant
```

6. 创建您刚刚定义的资源。例如：

```
$ oc create -f developeraccount01.yaml
```

对于给定示例，输出为：

```
$ developeraccount.capabilities.3scale.net/developeraccount01 created
```

后续步骤

3scale 操作器需要几秒钟时间才能更新 3scale 配置来反映新的或更新的自定义资源。要检查 Operator 是否成功传播自定义资源信息，请检查 **DeveloperAccount** CR **status** 字段或运行 **oc wait** 命令，例如：

```
$ oc wait --for=condition=Ready --timeout=30s developeraccount/developeraccount1
```

如果失败，自定义资源的 **status** 字段指示错误是临时还是永久性的，并提供有助于解决问题的错误消息。

通知任何新的开发人员用户，他们可以登录开发人员门户。您可能还需要传达其登录凭据。

您可以像更新任何其他自定义资源一样更新部署的 **DeveloperAccount** CR。例如，在包含拥有您要更新的 **DeveloperAccount** CR 的租户的 OpenShift 项目中，您将运行以下命令更新 **devaccount1** CR：

```
$ oc edit developeraccount devaccount1
```

其他资源

- 删除 DeveloperAccount 或 DeveloperUser 自定义资源
- DeveloperAccount CRD 参考
- DeveloperUser CRD 参考

8.13.3. 通过部署 DeveloperUser 自定义资源来管理 3scale API 管理开发人员用户

当使用 3scale 操作器安装 3scale 时，您可以部署 **DeveloperUser** 自定义资源(CR)来管理开发人员对 Developer Portal 中 3scale 管理的 API 的访问。此处提供的步骤是部署新的 **DeveloperUser** CR。部署 **DeveloperUser** CR 后，更新或删除它的过程与任何其他 CR 相同。

您只能在包含 3scale Operator 的命名空间中部署 CR。

先决条件

- 了解 3scale 操作器如何标识自定义资源链接的租户。
- 至少一个部署的 **DeveloperAccount** CR 包含至少一个部署的 **DeveloperUser** CR，用于具有 **admin** 角色的用户。如果您要创建一个 **DeveloperUser** CR，它没有链接到同一命名空间中的 3scale 实例中的默认租户，则包含 **DeveloperUser** CR 的命名空间包含一个 secret，用于标识 **DeveloperUser** CR 链接的租户。secret 的名称是以下之一：
  - **threescale-provider-account**
  - 用户定义

  此机密包含 3scale 实例的 URL 和令牌，其中包含用于访问 3scale 实例中某一租户的凭据。
- 对于新的 **DeveloperUser** CR，您有该开发人员的用户名、密码和电子邮件地址。

流程

1. 在包含 3scale 操作器的命名空间中，创建并保存一个资源文件，该文件定义包含开发人员用户的用户名和密码的 secret。例如，**myusername02.yaml** 文件可能包含：

```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: myusername02
stringData:
  password: "987654321"
```

2. 创建 secret.例如：

```
$ oc create -f myusername02.yaml
```

对于给定示例，输出为：

```
$ secret/myusername02 created
```

3. 创建并保存定义 **DeveloperUser** CR 的一个 **.yaml** 文件。在 **spec.role** 字段中，指定 **admin** 或 **member**。例如，**developeruser02.yaml** 文件可能包含：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: DeveloperUser
metadata:
  name: developeruser02
spec:
  username: myusername02
  email: myusername02@example.com
  passwordCredentialsRef:
    name: myusername02
  role: member
  developerAccountRef:
    name: developeraccount1
  providerAccountRef:
    name: mytenant
```

在 **DeveloperUser** CR 中：

- 开发人员用户名（在 **metadata.name** 字段中指定），用户名和电子邮件必须在包含 **DeveloperAccount** 链接的租户中唯一。
- **developerAccountRef** 字段必须指定已部署的 **DeveloperAccount** CR 的名称。
- **DeveloperUser** CR 链接的租户必须是与指定的 **DeveloperAccount** CR 链接相同的租户。

4. 创建您刚刚定义的资源。例如：

```
$ oc create -f developefuser02.yaml
```

对于给定示例，输出为：

```
$ developeruser.capabilities.3scale.net/developeruser02 created
```

后续步骤

3scale 操作器需要几秒钟时间才能更新 3scale 配置来反映新的或更新的自定义资源。要检查 Operator 是否成功传播自定义资源信息，请检查 **DeveloperUser** CR **status** 字段或运行 **oc wait** 命令，例如：

```
$ oc wait --for=condition=Ready --timeout=30s developeruser/developeruser02
```

如果失败，自定义资源的 **status** 字段指示错误是临时还是永久性的，并提供有助于解决问题的错误消息。

通知任何新的开发人员用户，他们可以登录开发人员门户。您可能还需要传达其登录凭据。

您可以像更新任何其他自定义资源一样更新部署的 **DeveloperUser** CR。

其他资源

- 删除 DeveloperAccount 或 DeveloperUser 自定义资源
- DeveloperUser CRD 参考

8.13.4. 删除 DeveloperAccount 或 DeveloperUser 自定义资源

您可以通过删除管理它的自定义资源(CR)来删除 3scale 开发人员实体。当您删除 **DeveloperAccount** CR 时，3scale Operator 也会删除链接到已删除 **DeveloperAccount** CR 的任何 **DeveloperUser** CR。

重要

删除自定义资源中定义的开发人员帐户或开发人员用户的唯一方法是按照此处所述的步骤进行操作。不要使用 Admin Portal 或 3scale API 来删除部署为自定义资源的开发人员实体。

先决条件

- 3scale 管理员权限或具有删除命名空间中具有删除权限的 OpenShift 角色，其中包含您要删除的自定义资源。要确认您可以删除特定的自定义资源，请运行 **oc auth can-i delete** 命令。例如，如果 **DeveloperAccount** CR 中的名称是 **devaccount1**，则运行以下命令：

```
$ oc auth can-i delete developeraccount.capabilities.3scale.net/devaccount1 -n my-namespace
```

- 要删除到有效租户的 **DeveloperAccount** 或 **DeveloperUser** CR 的链接。

流程

- 在包含自定义资源链接的租户的 OpenShift 项目中，运行 **oc delete** 命令删除 **DeveloperAccount** 或 **DeveloperUser** CR。例如，如果您部署了 **devaccount1.yaml** 文件中定义的 **DeveloperAccount** CR，则可运行以下命令：

```
$ oc delete -f devaccount1.yaml
```

或者，您可以运行 **oc delete** 命令并指定其定义中指定的 CR 名称。例如：

```
$ oc delete developeraccount.capabilities.3scale.net/devaccount1
```

8.14. 3SCALE API 管理 OPERATOR 功能的限制

在 Red Hat 3scale API Management 2.14 中，3scale Operator 包含了这些限制：

- 无法协调后端自定义资源定义(CRD)：3scale 中的现有后端不会被删除。

- 不协调删除产品 CRD：3scale 中的现有产品不会被删除。
- 删除 **DeveloperAccount** 或 **DeveloperUser** 自定义资源(CR)不会被协调。虽然操作器收到删除事件，但操作器不会对事件做出反应。开发人员帐户或开发人员用户保留。如果您尝试创建一个新的开发人员帐户或 developer 用户，其帐户名称、用户名或电子邮件地址与您删除的自定义资源相同，您会收到该帐户已存在的错误。您必须指定不同的参数来创建帐户。
- 管理门户的单点登录(SSO)身份验证。
- Developer Portal 的 SSO 身份验证。
- ActiveDocs CRD 当前不可用。
- 网关策略 CRD 目前不可用。
- 产品 CRD 网关不支持响应自定义代码和错误
- 3scale 拥有 OAS3 的 operator CRD 不作为 3scale 产品配置的真实来源引用。

8.15. 其他资源

如需更多信息，请查看以下指南：

- 后端 CRD 参考
- 产品 CRD 参考
- CustomPolicyDefinition CRD 参考
- 租户 CRD 参考

第 9 章 3SCALE API 管理备份和恢复

本节为您提供了作为 Red Hat 3scale API 管理安装的管理员所需的信息：

- 为持久数据设置备份程序。
- 从持久数据的备份中执行恢复。

如果一个或多个 MySQL 数据库出现问题，您可以将 3scale 正确恢复到之前的操作状态。

9.1. 先决条件

- 3scale 2.14 实例。有关如何安装 3scale 的更多信息，请参阅在 OpenShift 上安装 3scale API 管理。
- OpenShift Container Platform 4.x 用户帐户，在 OpenShift 集群中具有以下角色之一：
  - cluster-admin
  - admin
  - edit

注意

在 3scale 安装的命名空间中本地绑定了 *edit* 集群角色的用户，可以执行备份和恢复过程。

以下包含有关如何为持久数据设置备份程序的信息，从持久数据备份中执行恢复。如果一个或多个 MySQL 数据库失败，我将能够将 3scale 正确恢复到其以前的操作状态。

- 持久性卷和注意事项
- 使用数据集
- 备份系统数据库
- 恢复系统数据库

9.2. 持久性卷和注意事项

持久性卷（PV）

在 OpenShift 上的 3scale API 管理部署中 ：

- 由底层基础架构向集群提供的持久性卷(PV)。
- 集群外部的存储服务。这可以位于同一数据中心或其它位置。

注意事项

持久数据的备份和恢复过程因使用的存储类型而异。为确保备份和恢复保留数据一致性，仅备份数据库基础 PV 是不够的。例如，请勿只捕获部分写入和部分事务。改为使用数据库的备份机制。

数据的某些部分在不同的组件之间同步。一个副本被视为数据集的*真实来源*。另一个是不在本地修改的副本，但从 *事实来源* 同步。在这些情况下，在完成时，*应恢复数据源*，并从它同步的其他组件中复制。

9.3. 使用数据集

本节详细介绍了不同持久存储中的不同数据集、其用途、使用的存储类型，以及是否是 *数据源*。

3scale 部署的完整状态存储在以下 **DeploymentConfig** 对象及其 PV 中：

Name	描述
system-mysql	MySQL 数据库(**mysql-storage**)
system-storage	文件的卷
backend-redis	redis 数据库(**backend-redis-storage**)
system-redis	redis 数据库(**system-redis-storage**)

9.3.1. 定义 system-mysql

system-mysql 是一个关系数据库，存储在 3scale 管理控制台中有关用户、帐户、API、计划等的信息。

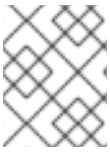
与服务相关的信息子集被同步到**后端**组件，并存储在 **backend-redis** 中。**system-mysql** 是此信息的*真实来源*。

9.3.2. 定义 system-storage

system-storage 存储要由 **系统** 组件读取和写入的文件。

它们分为两个类别：

- **系统** 组件运行时读取的配置文件
- 静态文件，如 *HTML*、*CSS*、*JS*，通过其 CMS 功能上传到系统，以便创建开发人员门户



注意

可以通过上传和读取所述静态文件的多个 pod 来水平扩展 **系统**，因此对 ReadWriteMany(RWX) **PersistentVolume**的需求。

9.3.3. 定义 backend-redis

backend-redis 包含由 **后端** 组件使用的多个数据集：

- **Usages**：这是 **Backend** 聚合的 API 使用量信息。**后端** 用于速率限制决策，**系统** 则用于在 UI 或 API 中显示分析信息。
- **Config**：这是关于服务、速率限制等的配置信息，它们通过内部 API 从 **System** 同步。这不是此信息的*真实来源*，但 **System** 和 **system-mysql** 是。
- **Queues**：这是由 worker 进程执行的后台作业队列。这些是临时的，会在处理后删除。

9.3.4. 定义 system-redis

system-redis 包含在后台处理作业的队列。这些是临时的，会在处理后删除。

9.4. 备份系统数据库

以下命令不按特定顺序运行，您可以根据需要使用这些命令来备份和归档系统数据库。

9.4.1. 备份 system-mysql

执行 MySQL 备份命令：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq -r '.items[0].metadata.name')
bash -c 'export MYSQL_PWD=${MYSQL_ROOT_PASSWORD}; mysqldump --single-transaction -
hsystem-mysql -uroot system' | gzip > system-mysql-backup.gz
```

9.4.2. 备份 system-storage

将 **system-storage** 文件归档到另一个存储中：

```
$ oc rsync $(oc get pods -l 'deploymentConfig=system-app' -o json | jq '.items[0].metadata.name' -
r):/opt/system/public/system ./local/dir
```

9.4.3. 备份 backend-redis

从 redis 备份 **dump.rdb** 文件：

```
$ oc cp $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq '.items[0].metadata.name' -
r):/var/lib/redis/data/dump.rdb ./backend-redis-dump.rdb
```

9.4.4. 备份 system-redis

从 redis 备份 **dump.rdb** 文件：

```
$ oc cp $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq '.items[0].metadata.name' -
r):/var/lib/redis/data/dump.rdb ./system-redis-dump.rdb
```

9.4.5. 备份 zync-database

备份 **zync_production** 数据库：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq -r '.items[0].metadata.name')
bash -c 'pg_dump zync_production' | gzip > zync-database-backup.gz
```

9.4.6. 备份 OpenShift secret 和 ConfigMap

以下是 OpenShift secret 和 ConfigMap 的命令列表：

9.4.6.1. OpenShift secret

```
$ oc get secrets system-smtp -o json > system-smtp.json
$ oc get secrets system-seed -o json > system-seed.json
```

```
$ oc get secrets system-database -o json > system-database.json
$ oc get secrets backend-internal-api -o json > backend-internal-api.json
$ oc get secrets system-events-hook -o json > system-events-hook.json
$ oc get secrets system-app -o json > system-app.json
$ oc get secrets system-recaptcha -o json > system-recaptcha.json
$ oc get secrets system-redis -o json > system-redis.json
$ oc get secrets zync -o json > zync.json
$ oc get secrets system-master-apicast -o json > system-master-apicast.json
```

9.4.6.2. ConfigMaps

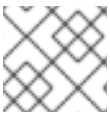
```
$ oc get configmaps system-environment -o json > system-environment.json
$ oc get configmaps apicast-environment -o json > apicast-environment.json
```

9.5. 恢复系统数据库

重要

通过缩减 pod（如 **system-app** 或禁用路由）来阻止记录创建。

在下面的命令和代码片段示例中，将 **${DEPLOYMENT_NAME}** 替换为您在创建 3scale 部署时定义的名称。



注意

确保输出至少包含大括号 **{}** 的一对，且不为空。

流程

1. 存储当前的副本数以便以后扩展：

```
SYSTEM_SPEC=`oc get APIManager/${DEPLOYMENT_NAME} -o
jsonpath='{.spec.system.appSpec}'`
```

2. 验证上一命令的结果并检查 **$SYSTEM_SPEC** 的内容：

```
echo $SYSTEM_SPEC
```

3. 使用以下命令对 APIManager CR 进行补丁，将副本数扩展到 **0** ：

```
$ oc patch APIManager/${DEPLOYMENT_NAME} --type merge -p '{"spec": {"system":
{"appSpec": {"replicas": 0}}}}'
```

另外，要缩减 **system-app**，请编辑现有的 **APIManager/${DEPLOMENT_NAME}**，并将系统副本数设置为零，如下例所示：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: <DEPLOYMENT_NAME>
spec:
```

```
system:
  appSpec:
    replicas: 0
```

使用以下步骤恢复 OpenShift secret 和系统数据库：

- 恢复基于 Operator 的部署
- 恢复 system-mysql
- 恢复 system-storage
- 恢复 zync-database
- 确保后端和系统之间的信息一致性

9.5.1. 恢复基于 Operator 的部署

使用以下步骤恢复基于 Operator 的部署。

流程

1. 在 OpenShift 上安装 3scale API Management 操作器。
2. 在创建 APIManager 资源前恢复 secret：

```
$ oc apply -f system-smtp.json
$ oc apply -f system-seed.json
$ oc apply -f system-database.json
$ oc apply -f backend-internal-api.json
$ oc apply -f system-events-hook.json
$ oc apply -f system-app.json
$ oc apply -f system-recaptcha.json
$ oc apply -f system-redis.json
$ oc apply -f zync.json
$ oc apply -f system-master-apicast.json
```

3. 在创建 APIManager 资源前恢复 ConfigMap：

```
$ oc apply -f system-environment.json
$ oc apply -f apicast-environment.json
```

4. 使用 APIManager CR 通过 Operator 部署 3scale API 管理。

9.5.2. 恢复 system-mysql

流程

1. 将 MySQL 转储复制到 system-mysql pod:

```
$ oc cp ./system-mysql-backup.gz $(oc get pods -l 'deploymentConfig=system-mysql' -o json
| jq '.items[0].metadata.name' -r):/var/lib/mysql
```

2. 解压缩备份文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq -r
'.items[0].metadata.name') bash -c 'gzip -d ${HOME}/system-mysql-backup.gz'
```

3. 恢复 MySQL DB 备份文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-mysql' -o json | jq -r
'.items[0].metadata.name') bash -c 'export MYSQL_PWD=${MYSQL_ROOT_PASSWORD};
mysql -hsystem-mysql -uroot system < ${HOME}/system-mysql-backup'
```

9.5.3. 恢复 system-storage

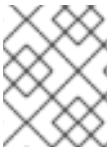
将备份文件恢复到 system-storage：

```
$ oc rsync ./local/dir/system/ $(oc get pods -l 'deploymentConfig=system-app' -o json | jq
'.items[0].metadata.name' -r):/opt/system/public/system
```

9.5.4. 恢复 zync-database

为 3scale Operator 部署恢复 **zync-database** 的说明。

9.5.4.1. 基于 Operator 的部署



注意

按照使用 operator 部署 3scale API 管理 中的说明，特别是 部署 APIManager CR 以重新部署 3scale 实例。

流程

1. 通过将 **${DEPLOYMENT_NAME}** 替换为您在创建 3scale 部署时定义的名称来存储副本数量：

```
ZYNC_SPEC=`oc get APIManager/${DEPLOYMENT_NAME} -o json | jq -r '.spec.zync'`
```

2. 将 zync DeploymentConfig 缩减为 0 个 pod：

```
$ oc patch APIManager/${DEPLOYMENT_NAME} --type merge -p '{"spec": {"zync":
{"appSpec": {"replicas": 0}, "queSpec": {"replicas": 0}}}}'
```

3. 将 Zync 数据库转储复制到 **zync-database** pod:

```
$ oc cp ./zync-database-backup.gz $(oc get pods -l 'deploymentConfig=zync-database' -o
json | jq '.items[0].metadata.name' -r):/var/lib/pgsql/
```

4. 解压缩备份文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq -r
'.items[0].metadata.name') bash -c 'gzip -d ${HOME}/zync-database-backup.gz'
```

5. 恢复 zync 数据库备份文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=zync-database' -o json | jq -r
'.items[0].metadata.name') bash -c 'psql zync_production -f ${HOME}/zync-database-backup'
```

6. 恢复到原始副本数：

```
$ oc patch APIManager/${DEPLOYMENT_NAME} --type json -p '[{"op": "replace", "path":
"/spec/zync", "value":"$ZYNC_SPEC"}]'
```

- 如果以下命令的输出不包含 **replicas** 键：

```
$ echo $ZYNC_SPEC
```

- 然后，运行以下命令来扩展 **zync** ：

```
$ oc patch dc/zync -p '{"spec": {"replicas": 1}}'
```

9.5.4.2. 使用 backend-redis 和 system-redis 恢复 3scale API 管理选项

通过恢复 3scale，您将恢复 **backend-redis** 和 **system-redis**。这些组件具有以下功能：

backend-redis** ：在 3scale 中支持应用程序身份验证和速率限制的数据库。它还用于统计存储和临时作业存储。system-redis** ：为 3scale 的后台作业提供临时存储，也用作 **system-app** pod 的 Ruby 处理的消息总线。

backend-redis 组件

backend-redis 组件有两个数据库，即 **data** 和 **queues**。在默认的 3scale 部署中，**数据**和**队列**部署在 Redis 数据库中，但在不同的逻辑数据库索引 **/0** 和 **/1** 中。恢复 **数据** 数据库不会有任何问题，但恢复 **队列** 数据库可能会导致重复的作业。

关于作业重复，在 3scale 中，后端 worker 以毫秒为单位处理后台作业。如果 **backend-redis** 在最后一次数据库快照后 30 秒失败，并且您尝试恢复它，则在 30 秒内发生的后台作业会执行两次，因为后端没有系统来避免重复。

在这种情况下，您必须恢复备份，因为 **/0** 数据库索引包含没有在其它任何位置保存的数据。恢复 **/0** 数据库索引意味着您必须还原 **/1** 数据库索引，因为一个在没有数据库索引的情况下将无法存储。当您选择在不同的服务器中分隔数据库，而不是在不同的索引中分隔一个数据库时，队列的大小大约为零，因此最好不要恢复备份并丢失几个后台作业。在 3scale 托管设置中会出现这种情况，因此您需要为两者应用不同的备份和恢复策略。

'system-redis'component

大多数 3scale 系统后台作业是幂等的，也就是说，无论您运行它们多少次，相同的请求都会返回相同的结果。

以下是系统中由后台作业处理的事件示例：

- 通知作业，例如计划试用到期、即将到期的信用卡、激活提醒、计划更改、发票状态更改、PDF 报告.
- 例如开票和收费。
- 删除复杂对象。
- 后端同步作业。

- 索引作业，例如使用 searchd。
- 清理工作，例如发票 ID。
- Janitorial 任务，如清除审计、用户会话、到期令牌、日志条目、暂停不活动帐户等。
- 流量更新。
- 代理配置更改监控和代理部署。
- 后台注册作业，
- Zync 作业，如单点登录(SSO)同步、路由创建。

如果您要恢复上述后台作业列表，3scale 的系统会维护每个恢复的作业的状态。在恢复完成后检查系统的完整性非常重要。

9.5.5. 确保后端和系统之间的信息一致性

恢复 **backend-redis** 后，应强制同步**系统**的配置信息，以确保 **后端** 中的信息与 **系统**中的信息一致，这是*事实来源*。

9.5.5.1. 管理 backend-redis 的部署配置

这些步骤旨在运行 **backend-redis** 的实例。

流程

1. 编辑 **redis-config** configmap：

```
$ oc edit configmap redis-config
```

2. 注释 **redis-config** configmap 中的 **SAVE** 命令：

```
#save 900 1
#save 300 10
#save 60 10000
```

3. 在 **redis-config** configmap 中将 **appendonly** 设置为 *no* ：

```
appendonly no
```

4. 重新部署 **backend-redis** 以加载新配置：

```
$ oc rollout latest dc/backend-redis
```

5. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/backend-redis
```

6. 重命名 **dump.rdb** 文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/dump.rdb ${HOME}/data/dump.rdb-
old'
```

7. 重命名 **appendonly.aof** 文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/appendonly.aof
${HOME}/data/appendonly.aof-old'
```

8. 将备份文件移到 POD 中：

```
$ oc cp ./backend-redis-dump.rdb $(oc get pods -l 'deploymentConfig=backend-redis' -o json
| jq '.items[0].metadata.name' -r):/var/lib/redis/data/dump.rdb
```

9. 重新部署 **backend-redis** 以加载备份：

```
$ oc rollout latest dc/backend-redis
```

10. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/backend-redis
```

11. 创建 **附加文件** ：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'redis-cli BGREWRITEAOF'
```

12. 片刻后，请确保完成 AOF 重写：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=backend-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'redis-cli info' | grep aof_rewrite_in_progress
```

- **aof_rewrite_in_progress = 1** 时，执行正在进行中。
- 定期检查，直到 **aof_rewrite_in_progress = 0**。零表示执行已经完成。

13. 编辑 **redis-config** configmap：

```
$ oc edit configmap redis-config
```

14. 在 **redis-config** configmap 中取消注释 **SAVE** 命令：

```
save 900 1
save 300 10
save 60 10000
```

15. 在 **redis-config** configmap 中将 **appendonly** 设置为 *yes* ：

```
appendonly yes
```

16. 重新部署 **backend-redis** 以重新载入默认配置：

```
$ oc rollout latest dc/backend-redis
```

17. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/backend-redis
```

9.5.5.2. 管理 system-redis 的部署配置

这些步骤旨在运行 **system-redis** 实例。

流程

1. 编辑 **redis-config** configmap：

```
$ oc edit configmap redis-config
```

2. 注释 **redis-config** configmap 中的 **SAVE** 命令：

```
#save 900 1
#save 300 10
#save 60 10000
```

3. 在 **redis-config** configmap 中将 **appendonly** 设置为 *no* ：

```
appendonly no
```

4. 重新部署 **system-redis** 以加载新配置：

```
$ oc rollout latest dc/system-redis
```

5. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/system-redis
```

6. 重命名 **dump.rdb** 文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/dump.rdb ${HOME}/data/dump.rdb-
old'
```

7. 重命名 **appendonly.aof** 文件：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'mv ${HOME}/data/appendonly.aof
${HOME}/data/appendonly.aof-old'
```

8. 将 **备份** 文件移到 POD 中：

```
$ oc cp ./system-redis-dump.rdb $(oc get pods -l 'deploymentConfig=system-redis' -o json |
jq '.items[0].metadata.name' -r):/var/lib/redis/data/dump.rdb
```

9. 重新部署 **system-redis** 以载入备份：

```
$ oc rollout latest dc/system-redis
```

10. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/system-redis
```

11. 创建 **附加文件** ：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'redis-cli BGREWRITEAOF'
```

12. 片刻后，请确保完成 AOF 重写：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-redis' -o json | jq
'.items[0].metadata.name' -r) bash -c 'redis-cli info' | grep aof_rewrite_in_progress
```

- **aof_rewrite_in_progress = 1** 时，执行正在进行中。
- 定期检查，直到 **aof_rewrite_in_progress = 0**。零表示执行已经完成。

13. 编辑 **redis-config** configmap：

```
$ oc edit configmap redis-config
```

14. 在 **redis-config** configmap 中取消注释 **SAVE** 命令：

```
save 900 1
save 300 10
save 60 10000
```

15. 在 **redis-config** configmap 中将 **appendonly** 设置为 *yes* ：

```
appendonly yes
```

16. 重新部署 **system-redis** 以重新载入默认配置：

```
$ oc rollout latest dc/system-redis
```

17. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/system-redis
```

9.5.6. 恢复 backend-worker

这些步骤旨在恢复 **backend-worker**。

流程

1. 恢复到 **backend-worker** 的最新版本：

```
$ oc rollout latest dc/backend-worker
```

2. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/backend-worker
```

9.5.7. 恢复 system-app

这些步骤旨在恢复 **system-app**。

流程

1. 要扩展 **system-app**，请编辑现有的 **APIManager/${DEPLOYMENT_NAME}**，并将 **.spec.system.appSpec.replicas** 改为原始副本数，或运行以下命令来应用之前存储的规格：

```
$ oc patch APIManager/${DEPLOYMENT_NAME} --type json -p '[{"op": "replace", "path": "/spec/system/appSpec", "value":'"$SYSTEM_SPEC"'}]'
```

- 如果以下命令的输出不包含 **replicas** 键：

```
$ echo $SYSTEM_SPEC
```

- 然后，运行以下命令来扩展 **system-app** ：

```
$ oc patch dc/system-app -p '{"spec": {"replicas": 1}}'
```

2. 恢复到最新版本的 **system-app** ：

```
$ oc rollout latest dc/system-app
```

3. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/system-app
```

9.5.8. 恢复 system-sidekiq

这些步骤旨在恢复 **system-sidekiq**。

流程

1. 恢复到 **system-sidekiq** 的最新版本：

```
$ oc rollout latest dc/system-sidekiq
```

2. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/system-sidekiq
```

9.5.8.1. 恢复 system-searchd

这些步骤旨在恢复 **system-searchd**。

流程

1. 恢复到最新版本的 **system-searchd** ：

```
$ oc rollout latest dc/system-searchd
```

2. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/system-searchd
```

9.5.8.2. 恢复由 zync 管理的 OpenShift 路由

- 强制 zync 重新创建缺少的 OpenShift 路由：

```
$ oc rsh $(oc get pods -l 'deploymentConfig=system-sidekiq' -o json | jq
'.items[0].metadata.name' -r) bash -c 'bundle exec rake zync:resync:domains'
```

第 10 章 为 3SCALE API 管理配置 RECAPTCHA

本文档论述了如何为红帽 3scale API 管理内部配置 reCAPTCHA，以防止垃圾邮件。

先决条件

- 在 受支持的 OpenShift 版本 上安装并配置了 3scale 内部部署实例。
- 获取 reCAPTCHA v2 的站点密钥和 secret key。请参阅 Register a new site web 页面。
- 如果要使用域名验证，请将 Developer Portal 域添加到 allowlist 中。

要为 3scale 配置 reCAPTCHA，请执行以下步骤中介绍的步骤：

- 第 10.1 节 “为 3scale API 管理中的垃圾邮件保护配置 reCAPTCHA”

10.1. 为 3SCALE API 管理中的垃圾邮件保护配置 RECAPTCHA

要为垃圾邮件保护配置 reCAPTCHA，有两个选项来修补包含 reCAPTCHA 的 secret 文件。这些选项位于 OpenShift Container Platform(OCP)用户界面中，或者使用命令行界面(CLI)。

流程

1. **OCP 4.x:** 进入到 **Project: [Your_project_name] > Workloads > Secrets**
2. 编辑 **system-recaptcha** secret 文件。
   reCAPTHCA 服务的 **PRIVATE_KEY** 和 **PUBLIC_KEY** 必须使用 base64 格式编码。手动将密钥转换为 base64 编码。

注意

CLI reCAPTCHA 选项不需要 base64 格式编码。

- **CLI:** 键入以下命令：

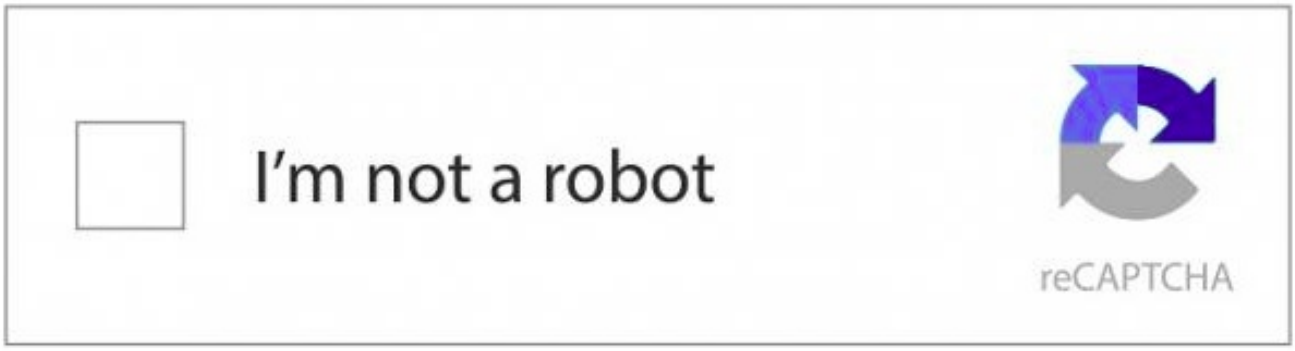
```
$ oc patch secret/system-recaptcha -p '{"stringData": {"PUBLIC_KEY": "public-key-from-service", "PRIVATE_KEY": "private-key-from-service"}}'
```

流程后的步骤

- 完成上述选项之一后重新部署系统 pod。
- 在 3scale 管理门户中，打开对未签名用户的垃圾邮件保护：
  1. 导航到 **Audience > Developer Portal > Spam Protection**
  2. 选择以下选项之一：
     - **Always**
       当向未登录的用户显示表单时，reCAPTCHA 始终会显示。
     - **仅可疑**
       只有在自动检查检测到可能的垃圾邮件时，才会显示 reCAPTCHA。

- **Never**
  关闭 Spam 保护。

重新部署 **system-app** 后，开发人员门户上使用垃圾邮件保护的页面将显示 reCAPTCHA *I'm not a robot* 复选框。



其他资源

- 如需更多信息、指南和支持，请参阅 ReCAPTCHA 主页。

第 11 章 3SCALE API MANAGEMENT WEBASSEMBLY 模块

threescale-wasm-auth 模块是一个 WebAssembly 模块，插入到 Service Mesh 中，它允许使用 Red Hat 3scale API Management 授权传入的请求。它扩展了 Service Mesh 功能，并提供完整的 API 管理功能，包括微服务的身份验证、分析和计费。

Service Mesh 侧重于基础架构层，它带有流量管理、服务发现、负载均衡和安全性等功能。API 管理侧重于创建、发布和管理 API。

Service Mesh 和 3scale 可以改进微服务和 API 的可靠性、可扩展性、安全性和性能。

注意

threescale-wasm-auth 模块在 3scale 2.11 或更高版本与 Red Hat OpenShift Service Mesh 2.1.0 或更高版本集成上运行。

先决条件

- 具有管理员特权的 3scale 帐户。
- Service Mesh 2.4 或更高版本安装。
  - Service Mesh 2.3 当前无法正常工作，因为 OSSM-3647。
  - 对于 Service Mesh 2.1 和 2.2，请参阅 使用 3scale API Management WebAssembly 模块。
- 在 Service Mesh 中运行的应用程序。
  - 使用 Bookinfo 示例应用程序。

OpenShift Container Platform (OCP) 上的集群管理员可配置 **threescale-wasm-auth** 模块，以通过 WasmPlugin 自定义资源授权 HTTP 请求到 3scale。然后，Service Mesh 会将模块注入 sidecar，公开主机服务，并允许您使用模块处理代理请求。

从 3scale 的角度来看，**threescale-wasm-auth** 模块充当一个网关，并在与 Service Mesh 集成时替换 APIcast。这意味着无法使用一些 APIcast 功能，特别是策略和暂存和生产环境。

11.1. 将 BOOKINFO 应用程序部署到 SERVICE MESH

您可以使用 Service Mesh 中的示例 Bookinfo 应用程序来演示使用 3scale 配置 Service Mesh 的步骤。

流程

1. 部署 Bookinfo 应用程序：
   - 请参阅 Bookinfo 示例应用程序。
2. 验证应用程序是否可用：

```
$ export GATEWAY_URL=$(oc -n istio-system get route istio-ingressgateway -o
jsonpath='{.spec.host}')

$ curl -I "http://$GATEWAY_URL/productpage"
HTTP/1.1 200 OK
```

11.2. 在 3SCALE API 管理中创建产品

产品是面向客户的 API，它可以重定向或使用多个内部 API（称为后端）。当在 Service Mesh 中使用 3scale 时，不会使用后端。产品和私有基本 URL 之间的链接在网格中进行。因此，只需要产品。

流程

1. 创建新产品、应用程序计划和应用程序。请参阅 创建新产品来测试 API 调用。
2. 将部署改为 **Istio** ：
   - 导航到 **[Your_product_name] > Integration > Settings**。
   - 将部署改为 **Istio**。
   - 点 **Update Product** 更新配置。
3. 提升配置：
   - 进入 **[Your_product_name] > Integration > Configuration**。
   - 单击 **Update Configuration**。

11.3. 使用 SERVICE MESH 连接 3SCALE API 管理

重要

在 service-mesh/istio-system 命名空间或 info 命名空间中创建 **ServiceEntry** 自定义资源 (CR) 和 **DestinationRule** CR。它应该位于包含 ServiceMeshControlPlane 的命名空间中。

要从 Service Mesh 访问 3scale，您必须通过 **ServiceEntry** CR 和 **DestinationRule** CR 将租户和后端 URL 配置为外部服务。这可让 **threescale-wasm-auth** 模块访问处理请求授权的后端，以及从中获取产品配置的系统。

11.3.1. 在 Service Mesh 中添加 3scale API 管理 URL

ServiceEntry 需要允许来自 Service Mesh 中的服务的请求，而 **DestinationRule** 则可用于为 3scale 服务配置安全连接。

11.3.1.1. 在 Service Mesh 中添加租户 URL

流程

1. 收集系统租户 URL：
   - 这是用于创建产品的 3scale 管理门户的 URL。
2. 为系统创建 **ServiceEntry** ：

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
```

```
  name: <service_entry_threescale_system>
spec:
  hosts:
  - <system_hostname>
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
EOF
```

3. 为系统创建 **DestinationRule** ：

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: <destination_rule_threescale_system>
spec:
  host: <system_hostname>
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: <system_hostname>
EOF
```

其他资源

- 了解服务条目
- 了解目的地规则

11.4. 在 SERVICE MESH 中添加后端 URL

通过将 3scale 后端 URL 合并到 Service Mesh 设置中，您可以在微服务和 3scale 后端之间建立安全通信频道。该集成支持实现在 Service Mesh 环境中管理 API 的身份验证、分析和计费功能。可以使用公开的路由并在内部使用 OpenShift 服务访问后端。

11.4.1. 在 Service Mesh 的不同集群中使用 3scale API 管理

流程

1. 收集后端 URL：
   - 对于托管的 3scale，后端 URL 为：**su1.3scale.net**
   - 对于 3scale 内部部署，使用以下命令获取 URL：

```
$ oc get -n <3scale_namespace> route backend --template="{{.spec.host}}"
```

2. 为后端创建 **ServiceEntry** ：

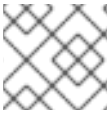
```
oc apply -n <info> -f -<<EOF
```

```
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: <service_entry_threescale_backend>
spec:
  hosts:
  - <backend_hostname>
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
EOF
```

3. 为后端创建 **DestinationRule** ：

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: <destination_rule_threescale_backend>
spec:
  host: <backend_hostname>
  trafficPolicy:
    tls:
      mode: SIMPLE
      sni: <backend_hostname>
EOF
```

11.5. 在与 SERVICE MESH 相同的集群中使用 3SCALE API 管理



注意

以下流程是向服务网格添加后端 URL 的替代选择。

要让 **threescale-wasm-auth** 模块授权针对 3scale 的请求，该模块必须有权访问 3scale 服务。您可以通过应用外部 **ServiceEntry** 对象和用于 *TLS* 配置的对应 **DestinationRule** 对象来使用 *HTTPS* 协议在 Red Hat OpenShift Service Mesh 中执行此操作。

自定义资源(CR)设置服务条目和目的地规则，以便从 Service Mesh 内安全访问 3scale，用于服务管理 API 和帐户管理 API 的后端和系统组件。Service Management API 接收每个请求的授权状态查询。帐户管理 API 为您的服务提供 API 管理配置设置。

流程

1. 为后端创建 **ServiceEntry** ：

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: <service_entry_threescale_backend>
spec:
```

```
  hosts:
  - backend-listener.<3scale_namespace>.svc.cluster.local
  ports:
  - number: 80
    name: http
    protocol: HTTP
  location: MESH_EXTERNAL
  resolution: DNS
EOF
```

2. 为后端创建 **DestinationRule** ：

```
oc apply -n <info> -f -<<EOF
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: <destination_rule_threescale_backend>
spec:
  host: backend-listener.<3scale_namespace>.svc.cluster.local
EOF
```

11.6. 创建 WASMPLUGIN 自定义资源

Service Mesh 提供了一个自定义资源定义 (CRD) 来指定并应用 Proxy-WASM 扩展到 sidecar 代理，称为 **WasmPlugin**。Service Mesh 将自定义资源 (CR) 应用到需要使用 3scale 管理 *HTTP* API 的工作负载集合。

流程

1. 在您的服务网格部署中识别要将其模块部署到的 OpenShift Container Platform (OCP) 命名空间，如 info 项目。

2. 使用 registry.redhat.io 凭证获取 pull secret。

   - 在与 **WasmPlugin** 相同的命名空间中创建新的 pull secret 资源。

3. 您必须声明部署 **threescale-wasm-auth** 模块的命名空间，以及一个选择器来标识模块要应用到的应用程序集合：以下示例是 **threescale-wasm-auth** 模块的 CR 的 YAML 格式：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
 namespace: <namespace>
spec:
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  imagePullSecret: <pull_secret_resource>
  phase: AUTHZ
  priority: 100
  match:
   - mode: CLIENT
  selector:
    matchLabels:
      app: <selector>
  pluginConfig:
```

```
    api: v1
    system:
      name: system
      upstream:
        name: outbound|443||<system_host>
        url: <system_url>
        timeout: 5000
      token: <access_token>

    backend:
      name: backend
      upstream:
        name: outbound|<backend_port>||<backend_host>
        url: <backend_url>
        timeout: 5000
      extensions:
      - no_body
    services:
    - id: '<product_id>'
      authorities:
      - "*"
      credentials:
        user_key:
          - query_string:
              keys:
                - user_key
          - header:
              keys:
                - user_key
```

- **spec.pluginConfig** 字段因应用程序而异。所有其他字段在该自定义资源的多个实例之间都存在。
- 此特定的 **WasmPlugin spec.pluginConfig** 配置有查询字符串中提供的 **user_key** 身份验证。
- 解释：
  - **name**
    指定 3scale 中 **WasmPlugin** 的唯一名称或标识符。
  - **namespace**
    工作负载的命名空间。
  - **imagePullSecret**
    在第 2 步中创建的 pull secret 的名称。
  - **selector**
    工作负载标签选择器。使用 info 项目的 productpage。
  - **backend-port**
    取决于您使用的 3scale。请参阅在 Service Mesh 中添加 3scale URL 。例如，内部 3scale 使用端口 80，外部 3scale 使用端口 443。
  - **backend-host** 和 **system-host**
    使用您在将 *3scale URL 添加到 Service Mesh* 时使用的同一主机。

- **system-url 和 backend-url**
  使用对应的主机并添加协议。例如： **https://<system-host>**。
- **access-token**
  系统租户的访问令牌。
- **product_id**
  要使用的产品的 ID。如果您希望有多个产品，在 services 部分定义多个产品。

- 在 **spec.pluginConfig** 和其余自定义资源中配置了模块后，使用 **oc apply** 命令应用它：

```
$ oc apply -f threescale-wasm-auth-info.yaml
```

11.6.1. 3scale API 管理 WasmPlugin 身份验证选项

这些是 3scale User 密钥 (App id/App key) 身份验证的配置示例。

用户密钥

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
 ...
  pluginConfig:
  ...
    services:
    - id: '<service_id>'
      authorities:
      - "*"
      credentials:
        user_key:
          - query_string:
              keys:
                - user_key
          - header:
              keys:
                - user_key
```

App Id 和 App key

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  ...
  pluginConfig:
    ...
    services:
    - id: '<service_id>'
      authorities:
      - "*"
```

```
    credentials:
      app_id:
        - query_string:
            keys:
              - app_id
        - header:
            keys:
              - app_id
      app_key:
        - query_string:
            keys:
              - app_key
        - header:
            keys:
              - app_key
```

OIDC

除了用于 OpenID Connect (OIDC)的 **WasmPlugin** 本身外，您还需要一个名为 **RequestAuthentication** 的额外自定义资源。在应用 **RequestAuthentication** 时，它会使用原生插件配置 **Envoy** 以验证 JWT 令牌。代理会在运行模块前验证所有内容，因此任何失败的请求都不会将其发送到 3scale WebAssembly 模块。

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: <info>
spec:
  selector:
    matchLabels:
      app: <productpage>
  jwtRules:
  - issuer: >-
      "<url>/auth/realms/<realm_name>"
    jwksUri: >-
      "<url>/auth/realms/<realm_name>/protocol/openid-connect/certs"
```

解释

- **<url>** : 当使用 keycloak 配置时，OIDC 实例的 URL 指定用于身份验证配置的 keycloak OIDC 供应商的元数据端点。
- **<realm_name>**: OIDC 中使用的域的名称。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  ...
  pluginConfig:
    ...
    services:
    - id: '<service_id>'
```

```
    authorities:
    - "*"
    credentials:
app_id:
  - filter:
      path:
        - envoy.filters.http.jwt_authn
        - "0"
      keys:
        - azp
        - aud
      ops:
        - take:
            head: 1
```

其他资源

- 限制使用 JSON Web 令牌的访问
- Wasm Plugin

11.7. 测试配置的 API

您可以通过在向应用程序发出调用时执行身份验证检查来验证 API 配置的有效性。通过全面测试身份验证机制，您可以确保只处理授权请求、维护应用程序的安全性和完整性。

流程

1. 尝试使用 **WasmPlugin** 应用调用 Bookinfo 应用程序。它应该被拒绝，因为我们没有包含任何身份验证：

```
$ export GATEWAY_URL=$(oc -n istio-system get route istio-ingressgateway -o
jsonpath='{.spec.host}')

$ curl -I "http://$GATEWAY_URL/productpage"
HTTP/1.1 403
```

2. 检索用于身份验证的用户密钥：
   - 进入 **[Your_product_name] > Applications > Listings**。
   - 选项您的应用程序
   - 查找 **Authentication > User Key**。
3. 使用显示的用户密钥尝试再次进行调用。

```
$ curl -I "http://$GATEWAY_URL/productpage?user_key=$USER_KEY"
HTTP/1.1 200 OK
```

4. 验证在指标中注册了 hit。
   - 进入 **[Your_product_name] > Analytics > Traffic**。
   - 您应看到注册了您的调用。

11.8. 3SCALE API MANAGEMENT WEBASSEMBLY 模块配置

WasmPlugin 自定义资源规格提供了 **Proxy-WASM** 模块从中读取的配置。

该 spec 嵌入主机中，并由 **Proxy-WASM** 模块读取。通常，配置采用要解析的模块的 JSON 文件格式。但是，**WasmPlugin** 资源可以将 spec 值解释为 YAML，并将其转换为 JSON 以供模块使用。

如果您在独立模式中使用 **Proxy-WASM** 模块，则必须使用 JSON 格式编写配置。使用 JSON 格式意味着在 **host** 配置文件中根据需要使用转义和引用，如 **Envoy**。当您将 WebAssembly 模块与 **WasmPlugin** 资源搭配使用时，配置采用 YAML 格式。在这种情况下，无效的配置会强制模块根据其 JSON 表示将诊断显示到 sidecar 的日志记录流。

重要

EnvoyFilter 自定义资源(CR)不是受支持的 API，但它可在 3scale Istio 适配器或 Service Mesh 版本中使用。不建议使用 **EnvoyFilter** CR。使用 **WasmPlugin** API 而不是 **EnvoyFilter** CR。如果需要使用 **EnvoyFilter** CR，则必须以 JSON 格式指定 spec。

11.8.1. 配置 3scale API Management WebAssembly 模块

3scale WebAssembly 模块配置的架构取决于 3scale 帐户和授权服务，以及要处理的服务列表。

前提条件

在所有情形中，先决条件都是一组最低必填字段：

- 对于 3scale 帐户和授权服务：**backend-listener** URL。
- 要处理的服务列表：服务 ID 和至少一个凭据查找方法以及查找方法。
- 您将找到处理 **userkey**、以及带有 **appkey** 的 **appid**，以及 OpenID Connect(OIDC)模式的示例。
- WebAssembly 模块使用您在静态配置中指定的设置。例如，如果您向模块中添加映射规则配置，它将始终适用，即使 3scale 管理门户没有这样的映射规则。其余的 **WasmPlugin** 资源围绕 **spec.pluginConfig** YAML 条目存在。

11.8.2. 3scale WebAssembly API Management 模块 api 对象

3scale WebAssembly 模块的 **api** 顶级字符串定义模块要使用的配置版本。

注意

api 对象的不存在或不受支持的版本会导致 3scale WebAssembly 模块无法正常运行。

api 顶级字符串示例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
  namespace: <info>
spec:
```

```
  pluginConfig:
    api: v1
...
```

api 条目定义配置的其余值。唯一接受的值是 **v1**。破坏与当前配置兼容性或需要更多使用 **v1** 的模块无法处理的逻辑的新设置将需要不同的值。

11.8.3. 3scale API Management WebAssembly 模块系统对象

system 顶级对象指定如何访问特定帐户的 3scale 帐户管理 API。**upstream** 字段是对象最重要的部分。**system** 对象是可选的，但建议使用，除非您为 3scale WebAssembly 模块提供完全静态的配置。如果您不想提供与 3scale 的 *系统组件* 连接，则后者是一个选项。

当您在 **system** 对象之外提供静态配置对象时，静态配置对象始终优先。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    system:
      name: <saas_porta>
      upstream: <object>
      token: <my_account_token>
      ttl: 300
    ...
```

表 11.1. system 对象字段

名称	描述	必需
name	3scale 服务的标识符，目前没有在别处引用。	选填
upstream	要联系的网络主机的详细信息。**upstream** 代表 3scale 帐户管理 API 主机，称为 system。	是
token	具有读取权限的 3scale 个人访问令牌。	是
ttl	在尝试获取新更改之前，将从此主机检索到的配置视为有效的最少秒数。默认为 600 秒（10 分钟）。**注意：**没有最大值，但模块通常会在此 TTL apsaps 之后合理时间段内获取任何配置。	选填

11.8.4. 3scale API Management WebAssembly 模块上游对象

upstream 对象描述代理可以对其执行调用的外部主机。

```
apiVersion: maistra.io/v1
upstream:
  name: outbound|443||multitenant.3scale.net
  url: "https://myaccount-admin.3scale.net/"
  timeout: 5000
...
```

表 11.2. upstream 对象字段

名称	描述	必需
name	**name** 不是自由格式的标识符。它是外部主机的标识符，如代理配置中所定义。对于独立 **Envoy** 配置，它会映射到一个集群的名称，在其他代理中也称为 **上游（upstream）**。**注：** 这个字段的值，因为 Service Mesh 和 3scale Istio 适配器 control plane 根据使用竖线(|)作为多个字段分隔符的格式来配置名称。对于此集成，请始终使用格式：**outbound|<port>|| <hostname>**。	是
url	用于访问所描述服务的完整 URL。除非被方案所暗示，否则您必须包含 TCP 端口。	是
Timeout（超时）	超时时间（毫秒），使得响应时间超过响应时间的连接将被视为错误。默认值为 1000 秒。	选填

11.8.5. 3scale API Management WebAssembly 模块后端对象

backend 顶级对象指定如何访问 3scale Service Management API 来授权和报告 HTTP 请求。此服务由 3scale 的 *Backend* 组件提供。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    backend:
      name: backend
      upstream: <object>
    ...
```

表 11.3. backend 对象字段

名称	描述	必需
name	3scale 后端的标识符，目前没有在别处引用。	选填
upstream	要联系的网络主机的详细信息。这必须引用 3scale 帐户管理 API 主机，即已知系统。	是。最重要和必填字段。

11.8.6. 3scale API Management WebAssembly 模块服务对象

services 顶级对象指定由 **module** 的特定实例处理哪些服务标识符。

您必须指定要处理哪些服务，因为帐户有多个服务。其余的配置会围绕如何配置服务。

services 字段是必需的。它是必须至少包含一个服务的数组，才可使用。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    services:
    - id: "2555417834789"
      token: service_token
      authorities:
        - "*.app"
        - 0.0.0.0
        - "0.0.0.0:8443"
      credentials: <object>
      mapping_rules: <object>
    ...
```

services 数组中的每个元素代表 3scale 服务。

表 11.4. services 对象字段

名称	描述	必填
id	此 3scale 服务的标识符，目前没有在别处引用。	是

名称	描述	必填
token	此 **token** 可以在您的系统中的服务的代理配置中找到，也可以使用以下 **curl** 命令从系统检索它： **curl "\https://<system_host>/admin/api/services/<service_id>/proxy/configs/production/latest.json?access_token=<access_token>" | jq '.proxy_config.content.backend_authentication_value'**	选填
authorities	一个字符串数组，每个字符串代表要匹配的 *URL* 的*颁发机构*。这些字符串接受支持星号(*)加号(+)和问号(?)匹配器的 glob 模式。	是
credentials	定义要查找和在哪里查找的凭据的对象。	是
mapping_rules	代表要命中映射规则和 3scale 方法的一组对象。	选填

11.8.7. 3scale API Management WebAssembly 模块凭证对象

credentials 对象是 **service** 对象的组件。**credentials** 指定要查找的凭证类型，以及执行此操作的步骤。

所有字段均为可选，但您必须至少指定一个 **user_key** 或 **app_id**。指定每个凭据的顺序无关紧要，因为它由模块预先建立。仅指定每个凭证的一个实例。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    services:
    - credentials:
        user_key: <array_of_lookup_queries>
        app_id: <array_of_lookup_queries>
        app_key: <array_of_lookup_queries>
      ...
```

表 11.5. credentials 对象字段

名称	描述	必需
user_key	这是一组查询，用于定义 3scale 用户密钥。用户密钥通常称为 API 密钥。	选填
app_id	这是一组查询，用于定义 3scale 应用标识符。应用程序标识符由 3scale 提供，或使用 Red Hat Single Sign-On (RH-SSO) 或 OpenID Connect(OIDC)等身份提供程序来提供。此处指定的查找查询的解析（只要成功并解析为两个值），它会设置 **app_id** 和 **app_key**。	选填
app_key	这是一组用于定义 3scale 应用键的查询。没有解析的 **app_id** 的应用程序密钥是无用的，因此仅在指定 **app_id** 时指定此字段。	选填

11.8.8. 3scale API Management WebAssembly 模块查找查询

lookup query 对象是 **credentials** 对象中任何字段的一部分。它指定如何查找和处理给定凭证字段。评估之后，成功解析意味着找到一个或多个值。失败的解决方案意味着没有找到任何值。

lookup queries 的数组描述了一个短电路或关系：成功解析其中一个查询会停止评估任何剩余查询，并将值或值分配到指定的凭证类型。数组中的每个查询相互独立。

lookup queries 由单个字段（一个源对象）组成，它可以是多个源类型之一。请参见以下示例：

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    services:
    - credentials:
        user_key:
          - <source_type>: <object>
          - <source_type>: <object>
          ...
        app_id:
          - <source_type>: <object>
          ...
        app_key:
          - <source_type>: <object>
          ...
    ...
```

source 对象作为任何 **credentials** 对象字段中的源数组的一部分存在。对象字段名称，称为 **source** 类型代表以下任意一个：

- **header** ：查找查询接收 HTTP 请求标头作为输入。
- **query_string** ：**lookup query** 接收 URL 查询字符串参数作为输入。
- **filter**：**lookup query** 接收过滤器元数据作为输入。

所有 **source** 类型对象至少具有以下两个字段：

表 11.6. source 类型对象字段

名称	描述	必需
keys	一个字符串数组，各自对应一个 **key**，引用输入数据中找到的条目。	是
ops	用于执行 **key** 项匹配的**操作**数组。该数组是操作在下一个操作上接收输入并生成输出的管道。如果 **operation** 无法提供一个输出将会被解析为 **lookup query** 失败。操作的管道顺序决定了评估顺序。	选填
path	显示用于查找数据的元数据中的路径。但是，当使用 **header** 或 **query_string** 源类型时不需要它，但在使用 **filter** source-type 时是必需的。	选填

当 **key** 与输入数据匹配时，不会评估其余的密钥，而且源解析算法会跳转到执行指定的**操作**（**ops**），如果存在。如果没有指定 **ops**，则返回匹配 **key** 的结果值（若有）。

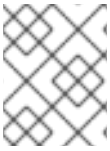
Operations 提供了一种方式，用于您在第一阶段查找 **key** 后为输入指定某些条件和转换。当您需要转换、解码和断言属性时，请使用 **Operations**，但它们不提供成熟的语言来满足所有需求并缺少 *Turing-completeness*。

存储 **operations** 输出的堆栈。评估时，**lookup query** 通过在堆栈的底部分配值或值来完成，具体取决于凭据使用的值。

11.8.9. 3scale API Management WebAssembly 模块操作对象

属于特定 **source type** 的 **ops** 数组中的每个元素都是 **operation** 对象，可以应用转换到值或执行测试。用于此类对象的字段名称是 **operation** 本身的名称，任何值都是 **operation** 的参数，可以是结构对象，例如，带有字段和值、列表或字符串的映射。

大多数 **operation** 都参与一个或多个输入，产生一个或多个输出。当它们消耗输入或生成输出时，它们与一个堆栈相关：操作消耗的每个值都从堆栈中弹出，最初填充任何 **source** 匹配。它们输出的值将推送到堆栈。其他 **operations** 没有使用或生成的输出不是声明的特定属性，但您检查值的堆栈。



注意

完成解析后，下一步获取的值，例如将值分配给 **app_id**、**app_key** 或 **user_key**，取自堆栈的底部值。

有几个不同的 **operations** 类别：

- **解码**
  这些通过解码来转换输入值，以获得不同的格式。
- **string**
  它们取字符串值作为输入，并对它执行转换和检查。
- **queue**
  它们取输入中的一组值，并执行多个堆栈转换和选择堆栈中的特定位置。
- **check**
  这些 assert 属性以一个副作用的方式有关一组操作。
- **Control （控制）**
  它们执行允许修改评估流程的操作。
- **格式**
  它们解析输入值的格式特定结构，并在其中查找值。

所有操作都由名称标识符以字符串形式指定。

其他资源

- 可用的操作

11.8.10. 3scale API Management WebAssembly 模块 mapping_rules 对象

mapping_rules 对象是 **service** 对象的一部分。它指定一组 REST 路径模式和相关 3scale 指标，并在模式匹配时指定要使用的递增数。

如果 **system** 顶级对象中没有提供动态配置，则需要该值。如果对象在 **system** 顶级条目外提供，则首先评估 **mapping_rules** 对象。

mapping_rules 是一个数组对象。该数组的每个元素都是 **mapping_rule** 对象。传入请求上评估的匹配映射规则提供了一组 3scale **methods**，用于授权并向 *APIManager* 报告。当多个匹配规则指代相同的 **methods** 时，调用 3scale 时会有一个 **deltas** 的总结。例如，如果两个规则使用 **deltas** 1 和 3 将 *Hits* 方法增加两次，则报告至 3scale 的 Hits 的单一方法条目的 **delta** 为 4。

11.8.11. 3scale API Management WebAssembly 模块 mapping_rule 对象

mapping_rule 对象是 **mapping_rules** 对象中的数组的一部分。

mapping_rule 对象字段指定以下信息：

- 要匹配的 *HTTP 请求方法*。
- 匹配路径的模式.
- 要报告的 3scale 方法以及要报告的数量。指定字段的顺序决定了评估顺序。

表 11.7. mapping_rule 对象字段

名称	描述	必需
方法	指定代表 HTTP 请求方法的字符串，也称为 verb。接受的值与接受的 HTTP 方法名称之一匹配，不区分大小写。任何方法都匹配的特殊值。	是
pattern	与 HTTP 请求的 URI 路径组件匹配的模式。此模式遵循与 3scale 中记录的相同语法。它允许使用大括号（如 **{this}**）之间的任意字符序列使用通配符（使用星号(*)字符）。	是
usages	**usage** 对象列表。当规则匹配时，所有带有其 **deltas** 的方法都会添加到发送到 3scale 的方法列表中，以进行授权和报告。 使用以下必填字段嵌入 **usages** 对象： ● **name**: 要报告的 **method** 系统名。**注意：name** 是区分大小写的。 ● **delta**: **method** 增加的数量。	是
last	当成功与此规则匹配，是否应停止评估更多映射规则。	可选布尔值。默认值为 **false**

以下示例独立于 3scale 中方法之间的现有层次结构。也就是说，在 3scale 侧运行的任何内容都不会受到影响。例如，*Hits* 指标可以是全部的父项，因此它存储了 4 个命中，因为授权请求中的所有报告方法总和，并调用 3scale **Authrep** API 端点。

以下示例使用到匹配所有规则的路径 **/products/1/sold** 的 **GET** 请求。

mapping_rules GET 请求示例

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  pluginConfig:
    ...
    mapping_rules:
      - method: GET
        pattern: /
```

```
      usages:
        - name: hits
          delta: 1
    - method: GET
      pattern: /products/
      usages:
        - name: products
          delta: 1
    - method: ANY
      pattern: /products/{id}/sold
      usages:
        - name: sales
          delta: 1
        - name: products
          delta: 1
...
```

所有 **usages** 都会添加到模块执行的请求中使用用量数据 3scale，如下所示：

- 命中：1
- 产品：2 个
- 销售：1

11.9. 凭证用例的 3SCALE API MANAGEMENT WEBASSEMBLY 模块示例

您将花费大部分时间应用配置步骤，在请求您的服务中获取凭证。

以下是 **credentials** 示例，您可以对其进行修改以符合特定用例的要求。

您可以组合使用它们，尽管当您指定多个源对象和自己的 **lookup queries** 时，会按照顺序对它们进行评估，直到其中一个成功解析为止。

11.9.1. 查询字符串参数中的 API 键 (user_key)

以下示例在查询字符串参数或相同名称的标头中查找 **user_key** ：

```
credentials:
  user_key:
    - query_string:
        keys:
          - user_key
    - header:
        keys:
          - user_key
```

11.9.2. 应用程序 ID 和密钥

以下示例在查询或标头中查找 **app_key** 和 **app_id** 凭据。

```
credentials:
  app_id:
    - header:
```

```
      keys:
        - app_id
  - query_string:
      keys:
        - app_id
app_key:
  - header:
      keys:
        - app_key
  - query_string:
      keys:
        - app_key
```

11.9.3. 授权标头

请求在 **authorization** 标头中包含 **app_id** 和 **app_key**。如果末尾至少输出了一个或两个值，您可以分配 **app_key**。

如果末尾输出了一两个或两个，此处的解决方法将分配 **app_key**。

authorization 标头使用授权类型指定值，其值编码为 **Base64**。这意味着，您可以通过空格字符来划分值，取第二个输出，然后使用冒号(:)作为分隔符再次分割它。例如，如果您使用这种格式 **app_id:app_key**，则标头类似以下示例 **credential** ：

```
aladdin:opensesame: Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

您必须使用小写标头字段名称，如下例所示：

```
credentials:
  app_id:
    - header:
        keys:
          - authorization
        ops:
          - split:
              separator: " "
              max: 2
          - length:
              min: 2
          - drop:
              head: 1
          - base64_urlsafe
          - split:
              max: 2
  app_key:
    - header:
        keys:
          - app_key
```

以上用例示例查看 **authorization** 标头：

1. 它接受字符串值并通过空格分割，检查它是否至少生成两个 **credential** 类型和 **credential** 本身，然后丢弃 **credential** 类型。

2. 然后，它会解码包含所需数据的第二个值，并使用冒号(:)字符进行拆分，使其具有一个包含 **app_id** 的操作堆栈，然后解码 **app_key** （若存在）。

   a. 如果授权标头中不存在 **app_key**，则会检查其特定的源。例如，在本例，标头中带有键 **app_key**。

3. 要向 **credentials** 添加额外条件，允许 **Basic** 授权，其中 **app_id** 是 **aladdin** 或 **admin**，或者任何 **app_id** 长度至少为 8 个字符。

4. **app_key** 必须包含一个值，并且至少具有 64 个字符，如下例所示：

```
credentials:
  app_id:
    - header:
        keys:
          - authorization
        ops:
          - split:
              separator: " "
              max: 2
          - length:
              min: 2
          - reverse
          - glob:
              - Basic
          - drop:
              tail: 1
          - base64_urlsafe
          - split:
              max: 2
          - test:
              if:
                length:
                  min: 2
              then:
                - strlen:
                    max: 63
                - or:
                    - strlen:
                        min: 1
                    - drop:
                        tail: 1
          - assert:
              - and:
                  - reverse
                  - or:
                      - strlen:
                          min: 8
                      - glob:
                          - aladdin
                          - admin
```

5. 选取 **authorization** 标头值后，您可以通过淘汰堆栈来获取 **Basic credential** 类型，使类型放置在顶部。

6. 在其上运行通配匹配。验证凭据并且凭据被解码和分割后，您将获得堆栈底部的 **app_id**，还可能获得顶部的 **app_key**。

7. 运行 **测试：** 如果堆栈中有两个值，表示已获取 **app_key**。

    a. 确保字符串长度介于 1 到 63 之间，包括 **app_id** 和 **app_key**。如果密钥的长度为零，则将其丢弃，并像不存在密钥一样继续。如果只有一个 **app_id** 且没有 **app_key**，则缺少的其他分支表示测试和评估成功。

assert，最后一个操作表示它使它进入堆栈没有副作用。然后您可以修改堆栈：

1. 颠倒堆栈，使 **app_id** 位于顶部。

    a. 无论是否存在 **app_key**，取代堆栈可确保 **app_id** 处于顶级。

2. 使用 **and** 在测试期间保留堆栈的内容。
然后使用以下可能性之一：

    - 确保 **app_id** 的字符串长度至少为 8。
    - 确保 **app_id** 与 **aladdin** 或 **admin** 匹配。

11.9.4. OpenID Connect(OIDC)用例

对于 Service Mesh 和 3scale Istio 适配器，您必须部署一个 **RequestAuthentication**，如下例所示，填入您自己的工作负载数据和 **jwtRules**:

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: <info>
spec:
  selector:
    matchLabels:
      app: <productpage>
  jwtRules:
  - issuer: >-
      "<url>/auth/realms/<realm_name>"
    jwksUri: >-
      "<url>/auth/realms/<realm_name>/protocol/openid-connect/certs"
```

应用 **RequestAuthentication** 时，它会使用原生插件配置 **Envoy** 以验证 **JWT** 令牌。代理会在运行模块前验证所有内容，因此任何失败的请求都不会将其发送到 3scale WebAssembly 模块。

验证 **JWT** 令牌时，代理将其内容存储在内部元数据对象中，其键取决于插件的具体配置。通过这个用例，您可以通过包含未知密钥名称的单一条目来查找结构对象。

OIDC 的 3scale **app_id** 与 OAuth **client_id** 匹配。这可在 **JWT** 令牌的 **azp** 或 **aud** 字段中找到。

要从 Envoy 的原生 **JWT** 身份验证过滤器获取 **app_id** 字段，请参阅以下示例：

```
credentials:
  app_id:
    - filter:
        path:
```

```
        - envoy.filters.http.jwt_authn
        - "0"
      keys:
        - azp
        - aud
      ops:
        - take:
            head: 1
```

示例指示模块使用 **filter** 源类型从 **Envoy**特定的 **JWT** 身份验证原生插件中查找对象的过滤器元数据。此插件包含 **JWT** 令牌，作为具有单个条目和预配置名称的结构对象的一部分。使用 **0** 指定您将仅访问单个条目。

生成值是一个结构，您要解析以下两个字段：

- **azp** ：找到 **app_id** 的值。
- **aud**: 也可以找到这个信息的值。

该操作可确保仅保留一个值进行分配。

11.9.5. 从标头中选取 JWT 令牌

有些设置可能具有 **JWT** 令牌验证过程，经过验证的令牌将通过 JSON 格式的标头访问此模块。

要获得 **app_id**，请参阅以下示例：

```
credentials:
  app_id:
    - header:
        keys:
          - x-jwt-payload
        ops:
          - base64_urlsafe
          - json:
              - keys:
                  - azp
                  - aud
          - take:
              head: 1
```

11.10. 3SCALE API MANAGEMENT WEBASSEMBLY 模块最小工作配置

以下是 3scale WebAssembly 模块最小工作配置的示例：您可以复制并粘贴此内容，并编辑它以便使用自己的配置。

```
apiVersion: extensions.istio.io/v1alpha1
kind: WasmPlugin
metadata:
  name: <threescale_wasm_plugin_name>
spec:
  url: oci://registry.redhat.io/3scale-amp2/3scale-auth-wasm-rhel8:0.0.3
  imagePullSecret: <pull_secret_resource>
  phase: AUTHZ
  match:
```

```
    - mode: SERVER
  priority: 100
  selector:
    matchLabels:
      app: <productpage>
  pluginConfig:
    api: v1
    system:
      name: <system_name>
      upstream:
        name: outbound|443||multitenant.3scale.net
        url: https://istiodevel-admin.3scale.net/
        timeout: 5000
      token: <token>
    backend:
      name: <backend_name>
      upstream:
        name: outbound|443||su1.3scale.net
        url: https://su1.3scale.net/
        timeout: 5000
      extensions:
      - no_body
    services:
    - id: '2555417834780'
      authorities:
      - "*"
      credentials:
        user_key:
          - query_string:
              keys:
                - <user_key>
          - header:
              keys:
                - <user_key>
        app_id:
          - query_string:
              keys:
                - <app_id>
          - header:
              keys:
                - <app_id>
        app_key:
          - query_string:
              keys:
                - <app_key>
          - header:
              keys:
                - <app_key>
```

其他资源

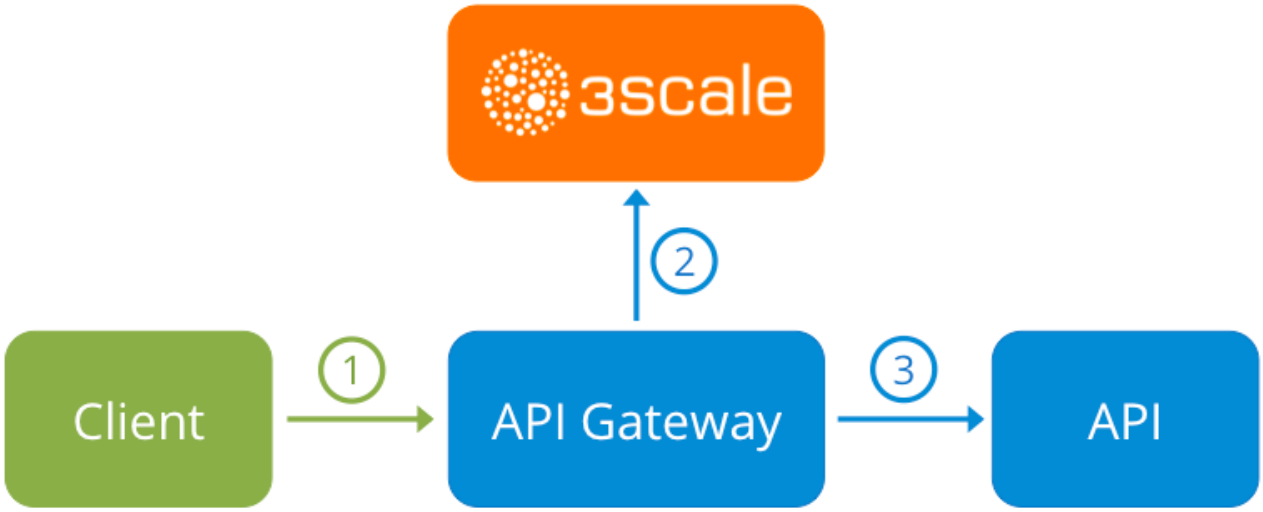
- 从 ServiceMeshExtension 迁移到 WasmPlugin 资源
- Kubernetes 自定义资源
- Wasm Plugin

第 12 章 对 API 基础架构进行故障排除

本指南旨在帮助您识别和修复 API 基础架构问题的原因。

API 基础架构是一个冗长而复杂的主题。但是，您的基础架构中至少需要三个移动部分：

1. API 网关
2. 3scale
3. API



这三种元素中的任何一个错误都会导致 API 用户无法访问您的 API。但是，很难找到导致失败的组件。本指南将为您提供对基础架构进行故障排除以确定问题的一些建议。

使用以下部分来识别并修复可能出现的常见问题：

- 常见集成问题
- 处理 API 基础架构问题
- 识别 API 请求问题
- 第 12.4 节 “ActiveDocs 问题”
- 第 12.5 节 “登录 NGINX”
- 第 12.6 节 “3scale 错误代码”

12.1. 常见集成问题

有一些证据可以指出您与 3scale 集成时存在一些非常常见的问题。它们将根据您是处于 API 项目开头、设置基础架构还是已存在于生产中而有所不同。

12.1.1. 集成问题

以下小节尝试概述了在与 3scale 集成的初始阶段，APIcast 错误日志中可能看到的一些常见问题：从使用 APIcast Hosted 开始，在启动之前，运行自我管理的 APIcast。

12.1.1.1. APIcast 托管

当您首次在 Service Integration 屏幕上将 API 与 APIcast Hosted 集成时，您可能会在页面中看到一些以下错误，或者通过测试调用返回，以检查集成是否成功。

- **测试请求失败：执行过期**
  检查您的 API 是否可从公共互联网访问。APIcast Hosted 无法与私有 API 一起使用。如果您不想使 API 公开可用于与 APIcast Hosted 集成，您可以在 APIcast Hosted 和 API 之间设置一个私有 secret，以拒绝任何来自 API 网关的调用。

- 可接受的格式是 **protocol://address(:port)**
  删除 API 专用基础 URL 末尾的所有路径。您可以在"映射规则"模式或 *API test GET request* 的开头添加它们。

- **使用 HTTP 代码 XXX 测试请求失败**

  - **405** ：检查端点是否接受 GET 请求。APIcast 只支持 GET 请求来测试集成。

  - **403: Authentication parameters missing**：如果您的 API 已有一些身份验证，APIcast 将无法发出测试请求。

  - **403: Authentication failed**：如果这不是您通过 3scale 创建的第一个服务，请检查您是否在服务下创建了带有凭据的应用，以发出测试请求。如果是您要集成的第一个服务，请确保您还没有删除在注册时创建的测试帐户或应用程序。

12.1.1.2. APIcast 自我管理

在成功测试了与 APIcast 自我管理的集成后，您可能希望自己托管 API 网关。以下是首次安装自我管理网关并通过它调用 API 时您可能会遇到的一些错误。

- **upstream timed out (110: Connection timed out) while connecting to upstream**
  检查 API 网关和公共 Internet 之间没有防火墙或代理，从而妨碍您自我管理的网关到达 3scale。

- **failed to get list of services: invalid status: 403 (Forbidden)**

  ```
  2018/06/04 08:04:49 [emerg] 14#14: [lua] configuration_loader.lua:134: init(): failed to load configuration, exiting (code 1)
  2018/06/04 08:04:49 [warn] 22#22: *2 [lua] remote_v2.lua:163: call(): failed to get list of services: invalid status: 403 (Forbidden) url: https://example-admin.3scale.net/admin/api/services.json , context: ngx.timer
  ERROR: /opt/app-root/src/src/apicast/configuration_loader.lua:57: missing configuration
  ```

  检查您在 **THREESCALE_PORTAL_ENDOINT** 值中使用的访问令牌是否正确，以及帐户管理 API 范围。使用 **curl** 命令验证它： **curl -v "https://example-admin.3scale.net/admin/api/services.json?access_token=<YOUR_ACCESS_TOKEN>"**

  它应当通过 JSON 正文返回 200 响应。如果返回错误状态代码，请检查响应正文以了解详细信息。

- **主机 apicast.example.com 没有找到服务**

  ```
  2018/06/04 11:06:15 [warn] 23#23: *495 [lua] find_service.lua:24: find_service(): service not found for host apicast.example.com, client: 172.17.0.1, server: _, request: "GET / HTTP/1.1", host: "apicast.example.com"
  ```

此错误表示公共基本 URL 尚未正确配置。您应该确保配置的 Public Base URL 与用于自助管理的 APIcast 请求的相同。配置正确的公共基本 URL 后：

- 确保 APIcast 已配置为"生产环境"（如果不使用 **THREESCALE_DEPLOYMENT_ENV** 变量覆盖独立 APIcast 的默认配置）。确保您将配置提升到生产环境。
- 如果您使用 **APICAST_CONFIGURATION_CACHE** 和 **APICAST_CONFIGURATION_LOADER** 环境变量自动重新加载配置，请重新启动 APIcast。

以下是一些可能指向不正确 APIcast 自助管理的集成的其他症状：

- **Mapping rules not matched / Double counting of API calls**: 根据您定义 API 上方法和实际 URL 端点之间的映射方式，您可能会发现有时每个请求都不匹配的方法或被递增一次。要对此进行故障排除，请通过 3scale 调试标头 对 API 发出测试调用。这将返回 API 调用匹配的所有方法的列表。

- **Authentication parameters not found**：确保您将参数发送到 Service Integration 屏幕中指定的正确位置。如果没有将凭证作为标头发送，则凭据必须以 GET 请求和正文参数的查询参数形式发送，所有其他 HTTP 方法。使用 3scale 调试标头来再次检查从 API 网关的请求读取的凭据。

12.1.2. 产品问题

在全面测试了设置并且已使用 API 长时间使用 API 后，很少会出现与 API 网关相关的问题。但是，以下是您可能在真实生产环境中遇到的一些问题：

12.1.2.1. 可用性问题

可用性问题通常由 nginx error.log 中的 **上游超时** 错误定性；例如：

```
upstream timed out (110: Connection timed out) while connecting to upstream, client: X.X.X.X,
server: api.example.com, request: "GET /RESOURCE?CREDENTIALS HTTP/1.1", upstream:
"http://Y.Y.Y.Y:80/RESOURCE?CREDENTIALS", host: "api.example.com"
```

如果您遇到 3scale 的可用性问题，这可能是原因：

- 您解析了不再使用的旧 3scale IP。
  最新版本的 API 网关配置文件将 3scale 定义为每次强制 IP 解析的变量。为获得快速修复，重新加载您的 NGINX 实例。对于长期修复，请确保将其定义为每个服务器块中的变量，而不是在上游块中定义 3scale 后端；例如：

```
server {
  # Enabling the Lua code cache is strongly encouraged for production use. Here it is enabled
  .
  .
  .
  set $threescale_backend "https://su1.3scale.net:443";
```

引用它时：

```
location = /threescale_authrep {
  internal;
  set $provider_key "YOUR_PROVIDER_KEY";

  proxy_pass $threescale_backend/transactions/authrep.xml?
```

```
provider_key=$provider_key&service_id=$service_id&$usage&$credentials&log%5Bcode%5
D=$arg_code&log%5Brequest%5D=$arg_req&log%5Bresponse%5D=$arg_resp;
}
```

- 您缺少白名单中的一些 3scale IP。以下是 3scale 解析到的 IP 的当前列表：
  - 75.101.142.93
  - 174.129.235.69
  - 184.73.197.122
  - 50.16.225.117
  - 54.83.62.94
  - 54.83.62.186
  - 54.83.63.187
  - 54.235.143.255
    以上问题是指感知的 3scale 可用性问题。但是，如果您的 API 位于 AWS ELB 后面，您可能会遇到与 API 网关的 API 可用性类似的问题。这是因为 NGINX 默认在启动时执行 DNS 解析，然后缓存 IP 地址。但是，ELB 无法确保静态 IP 地址，它们可能会经常更改。当 ELB 更改为其他 IP 时，NGINX 无法访问它。

    此问题的解决方案与强制进行运行时 DNS 解析的修复程序类似。

    1. 通过在 **http** 部分的顶部添加以下行来设置特定的 DNS 解析器，如 Google DNS： **解析器 8.8.8.8 8.8.4.4;**。
    2. 将 API 基本 URL 设置为位于 **server** 部分顶部任何位置的变量。**set $api_base "http://api.example.com:80";**
    3. 在 **location** / 部分中，找到 **proxy_pass** 行并将其替换为 **proxy_pass $api_base;**。

12.1.3. 部署后的问题

如果您更改了 API，如添加新端点，您必须确保在下载 API 网关的新配置文件之前添加新方法和 URL 映射。

当您修改了从 3scale 下载的配置时，最常见问题是在 Lua 中的代码错误，这会导致 **500 - 内部服务器错误**，例如：

```
curl -v -X GET "http://localhost/"
* About to connect() to localhost port 80 (#0)
*   Trying 127.0.0.1... connected
> GET / HTTP/1.1
> User-Agent: curl/7.22.0 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
librtmp/2.3
> Host: localhost
> Accept: */*
>
< HTTP/1.1 500 Internal Server Error
< Server: openresty/1.5.12.1
< Date: Thu, 04 Feb 2016 10:22:25 GMT
```

```
< Content-Type: text/html
< Content-Length: 199
< Connection: close
<

<head><title>500 Internal Server Error</title></head>

<center><h1>500 Internal Server Error</h1></center>
<hr><center>openresty/1.5.12.1</center>


* Closing connection #0
```

您可以看到 nginx error.log 了解原因，例如：

```
2016/02/04 11:22:25 [error] 8980#0: *1 lua entry thread aborted: runtime error:
/home/pili/NGINX/troubleshooting/nginx.lua:66: bad argument #3 to '_newindex' (number expected,
got nil)
stack traceback:
coroutine 0:
  [C]: in function '_newindex'
  /home/pili/NGINX/troubleshooting/nginx.lua:66: in function 'error_authorization_failed'
  /home/pili/NGINX/troubleshooting/nginx.lua:330: in function 'authrep'
  /home/pili/NGINX/troubleshooting/nginx.lua:283: in function 'authorize'
  /home/pili/NGINX/troubleshooting/nginx.lua:392: in function  while sending to client, client:
127.0.0.1, server: api-2445581381726.staging.apicast.io, request: "GET / HTTP/1.1", host: "localhost"
```

在 access.log 中，类似如下：

```
127.0.0.1 - - [04/Feb/2016:11:22:25 +0100] "GET / HTTP/1.1" 500 199 "-" "curl/7.22.0 (x86_64-pc-
linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3"
```

上一节将向您介绍在 3scale 旅程的任何阶段可能会遇到的最常见、知名的问题。

如果已经检查所有这些内容，且您仍无法找到问题的原因和解决方案，您应该进入有关 识别 API 请求问题 的更详细部分。从 API 开始，再重新尝试确定故障点到客户端。

12.2. 处理 API 基础架构问题

如果您在连接到服务器时遇到失败，无论是 API 网关、3scale 或 API，以下故障排除步骤应该是您调用的第一个端口：

12.2.1. 我们可以连接吗？

使用 telnet 检查基本 TCP/IP 连接 **telnet api.example.com 443**

- 成功

```
telnet echo-api.3scale.net 80
Trying 52.21.167.109...
Connected to tf-lb-i2t5pgt2cfdnbdfh2c6qqoartm-829217110.us-east-1.elb.amazonaws.com.
Escape character is '^]'.
Connection closed by foreign host.
```

- 失败

```
telnet su1.3scale.net 443
Trying 174.129.235.69...
telnet: Unable to connect to remote host: Connection timed out
```

12.2.2. 服务器连接问题

尝试从不同的网络位置、设备和方向连接到同一服务器。例如，如果您的客户端无法访问您的 API，请尝试从应该有权访问 API 的机器（如 API 网关）连接到您的 API。

如果有任何尝试的连接成功，您可以排除实际服务器中的任何问题，并在它们之间的网络上集中进行故障排除，因为这是问题最有可能是的地方。

12.2.3. 这是 DNS 问题吗？

尝试使用 IP 地址而不是其主机名（如 **telnet 94.125.104.17 80** 而不是 **telnet apis.io 80**）连接到服务器

这将排除 DNS 中的任何问题。

您可以使用 **dig** 获取服务器的 IP 地址，例如 3scale **dig su1.3scale.net** 或 **dig any su1.3scale.net** （如果怀疑主机可能解析到多个 IP 地址）。

NB: Some hosts block `dig any`

12.2.4. 这是 SSL 问题吗？

您可以使用 OpenSSL 测试：

- 保护到主机或 IP 的连接，例如从 shell prompt **openssl s_client -connect su1.3scale.net:443** 输出：

```
CONNECTED(00000003)
depth=1 C = US, O = GeoTrust Inc., CN = GeoTrust SSL CA - G3
verify error:num=20:unable to get local issuer certificate
---
Certificate chain
 0 s:/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks, S.L./OU=IT/CN=*.3scale.net
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
 1 s:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIE8zCCA9ugAwIBAgIQcz2Y9JNxH7f2zpOT0DajUjANBgkqhkiG9w0BAQsFADBE
...
TRUNCATED
...
3FZigX+OpWLVRjYsr0kZzX+HCerYMwc=
-----END CERTIFICATE-----
subject=/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks,
S.L./OU=IT/CN=*.3scale.net
issuer=/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
---
```

```
Acceptable client certificate CA names
/C=ES/ST=Barcelona/L=Barcelona/O=3scale Networks, S.L./OU=IT/CN=*.3scale.net
/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
Client Certificate Types: RSA sign, DSA sign, ECDSA sign
Requested Signature Algorithms:
RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384
:RSA+SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA22
4:RSA+SHA1:DSA+SHA1:ECDSA+SHA1:RSA+MD5
Shared Requested Signature Algorithms:
RSA+SHA512:DSA+SHA512:ECDSA+SHA512:RSA+SHA384:DSA+SHA384:ECDSA+SHA384
:RSA+SHA256:DSA+SHA256:ECDSA+SHA256:RSA+SHA224:DSA+SHA224:ECDSA+SHA22
4:RSA+SHA1:DSA+SHA1:ECDSA+SHA1
Peer signing digest: SHA512
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 3281 bytes and written 499 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID:
A85EFD61D3BFD6C27A979E95E66DA3EC8F2E7B3007C0166A9BCBDA5DCA5477B8
    Session-ID-ctx:
    Master-Key:
F7E898F1D996B91D13090AE9D5624FF19DFE645D5DEEE2D595D1B6F79B1875CF935B3
A4F6ECCA7A6D5EF852AE3D4108B
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - a8 8b 6c ac 9c 3c 60 78-2c 5c 8a de 22 88 06 15   ..l..<`x,\.."...
    0010 - eb be 26 6c e6 7b 43 cc-ae 9b c0 27 6c b7 d9 13   ..&l.{C....'l...
    0020 - 84 e4 0d d5 f1 ff 4c 08-7a 09 10 17 f3 00 45 2c   ......L.z.....E,
    0030 - 1b e7 47 0c de dc 32 eb-ca d7 e9 26 33 26 8b 8e   ..G...2....&3&..
    0040 - 0a 86 ee f0 a9 f7 ad 8a-f7 b8 7b bc 8c c2 77 7b   ..........{...w{
    0050 - ae b7 57 a8 40 1b 75 c8-25 4f eb df b0 2b f6 b7   ..W.@.u.%O...+..
    0060 - 8b 8e fc 93 e4 be d6 60-0f 0f 20 f1 0a f2 cf 46   .......`.. ....F
    0070 - b0 e6 a1 e5 31 73 c2 f5-d4 2f 57 d1 b0 8e 51 cc   ....1s.../W...Q.
    0080 - ff dd 6e 4f 35 e4 2c 12-6c a2 34 26 84 b3 0c 19   ..nO5.,.l.4&....
    0090 - 8a eb 80 e0 4d 45 f8 4a-75 8e a2 06 70 84 de 10   ....ME.Ju...p...

    Start Time: 1454932598
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
---
```

- SSLv3 支持（当前受 3scale 支持）
  openssl s_client -ssl3 -connect su.3scale.net:443

输出

```
CONNECTED(00000003)
140735196860496:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert handshake
failure:s3_pkt.c:1456:SSL alert number 40
140735196860496:error:1409E0E5:SSL routines:ssl3_write_bytes:ssl handshake
failure:s3_pkt.c:644:
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 0 bytes
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : SSLv3
    Cipher    : 0000
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1454932872
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
---
```

详情请查看 OpenSSL man page。

12.3. 识别 API 请求问题

要识别向您的 API 请求可能存在问题，请执行以下检查：

12.3.1. API

要确认 API 已启动并响应请求，请直接向 API 发出相同的请求（而不是通过 API 网关）。您应该确保发送与通过 API 网关的请求相同的参数和标头。如果您不确定失败的确切请求，请捕获 API 网关和 API 之间的流量。

如果调用成功，您可以排除 API 中的任何问题，否则您应该进一步对 API 进行故障排除。

12.3.2. API Gateway > API

要排除 API 网关和 API 之间的任何网络问题，请对您的 API 网关服务器发出与 before SAS-sule 相同的调用。

如果调用成功，您可以继续对 API 网关本身进行故障排除。

12.3.3. API 网关

可以通过多个步骤来检查 API 网关是否正常工作。

12.3.3.1. API 网关是否已启动并运行？

登录正在运行网关的计算机。如果此操作失败，您的网关服务器可能会停机。

登录后，检查 NGINX 进程是否正在运行。为此，请运行 **ps ax | grep nginx** 或 **htop**。

如果您看到列表中看到 **nginx master process** 和 **nginx worker process**，则代表 NGINX 正在运行。

12.3.3.2. 网关日志中是否存在任何错误？

以下是您可能在网关日志中看到的一些常见错误，例如在 error.log 中：

- API 网关无法连接到 API

> upstream timed out (110: Connection timed out) while connecting to upstream, client: X.X.X.X, server: api.example.com, request: "GET /RESOURCE?CREDENTIALS HTTP/1.1", upstream: "http://*Y.Y.Y.Y:80/RESOURCE?CREDENTIALS", host: "api.example.com"*

- API 网关无法连接到 3scale

> 2015/11/20 11:33:51 [error] 3578#0: *1 upstream timed out (110: Connection timed out) while connecting to upstream, client: 127.0.0.1, server: , request: "GET /api/activities.json?user_key=USER_KEY HTTP/1.1", subrequest: "/threescale_authrep", upstream: "https://*54.83.62.186:443/transactions/authrep.xml?provider_key=YOUR_PROVIDER_KEY&service_id=SERVICE_ID&usage[hits]=1&user_key=USER_KEY&log%5Bcode%5D=", host: "localhost"*

12.3.4. API gateway > 3scale API Management

确保 API 网关正确运行后，下一步是对 API 网关和 3scale 之间的连接进行故障排除。

12.3.4.1. API 网关能否访问 3scale API 管理？

如果您使用 NGINX 作为 API 网关，当网关无法联系 3scale 时，nginx 错误日志中会显示以下消息：

> 2015/11/20 11:33:51 [error] 3578#0: *1 upstream timed out (110: Connection timed out) while connecting to upstream, client: 127.0.0.1, server: , request: "GET /api/activities.json?user_key=USER_KEY HTTP/1.1", subrequest: "/threescale_authrep", upstream: "https://*54.83.62.186:443/transactions/authrep.xml?provider_key=YOUR_PROVIDER_KEY&service_id=SERVICE_ID&usage[hits]=1&user_key=USER_KEY&log%5Bcode%5D=", host: "localhost"*

在这里，记录下上游值。此 IP 对应于 3scale 产品解析到的 IP 地址之一。这意味着到达 3scale 存在问题。您可以通过调用 **nslookup** 进行反向 DNS 查找来检查 IP 的域。

例如，由于 API 网关无法访问 3scale，这并不表示 3scale 已关闭。其中一个最常见的原因是防火墙规则阻止 API 网关连接到 3scale。

网关和 3scale 之间可能存在网络问题，可能会导致连接超时。在这种情况下，您应该执行对 通用连接问题进行故障排除 的步骤，以确定问题所在的位置。

要排除网络问题，请使用 traceroute 或 MTR 来检查路由和数据包传输。您也可以从能够连接到 3scale 和 API 网关并比较输出的计算机运行相同的命令。

此外，若要查看 API 网关和 3scale 之间发送的流量，只要您临时切换为 3scale 产品(**su1.3scale.net**)的 HTTP 端点，就可以使用 tcpdump。

12.3.4.2. API 网关是否正确解析 3scale API 管理地址？

确保已将 resolver 指令添加到 nginx.conf 中。

例如，在 nginx.conf 中：

```
http {
  lua_shared_dict api_keys 10m;
  server_names_hash_bucket_size 128;
  lua_package_path ";;$prefix/?.lua;";
  init_by_lua 'math.randomseed(ngx.time()) ; cjson = require("cjson")';

  resolver 8.8.8.8 8.8.4.4;
```

您可以将 Google DNS（8.8.8.8 和 8.8.4.4）替换为您首选的 DNS。

要从 API 网关检查 DNS 解析，请按指定解析器 IP 调用 nslookup：

```
nslookup su1.3scale.net 8.8.8.8
;; connection timed out; no servers could be reached
```

上面的例子显示了如果无法到达 Google DNS，则返回的响应。如果出现这种情况，您必须更新解析器 IP。您可能还会在 nginx error.log 中看到以下警报：

```
2016/05/09 14:15:15 [alert] 9391#0: send() failed (1: Operation not permitted) while resolving,
resolver: 8.8.8.8:53
```

最后，运行 **dig any su1.3scale.net** 以查看 3scale 服务管理 API 当前运行的 IP 地址。请注意，这不是 3scale 可能使用的整个 IP 地址范围。一些可能因为容量原因而被交换和移出。此外，未来您可以为 3scale 服务添加更多域名。对于这一点，您应该始终针对集成期间为您提供的特定地址进行测试（如果适用）。

12.3.4.3. API 网关调用 3scale API 管理是否正确？

如果要检查 API 网关是否要进行 3scale 来满足故障排除的需要，将以下代码片段添加到 3scale authrep **nginx.conf**（/**threescale_authrep** 用于 API Key，App_id 验证模式）

```
body_filter_by_lua_block{
  if ngx.req.get_headers()["X-3scale-debug"] == ngx.var.provider_key then
    local resp = ""
    ngx.ctx.buffered = (ngx.ctx.buffered or "") .. string.sub(ngx.arg[1], 1, 1000)
    if ngx.arg[2] then
      resp = ngx.ctx.buffered
    end

    ngx.log(0, ngx.req.raw_header())
```

```
    ngx.log(0, resp)
  end
}
```

当发出 **X-3scale-debug 标头** 时，这个片断会将以下额外日志添加到 nginx 错误.log 中，如 **curl -v -H 'X-3scale-debug: yourR_ conclusion_KEY' -X GET "https://726e3b99.ngrok.com/api/contacts.json?access_token=7c6f24f5"**

这会生成以下日志条目：

```
2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:7: GET /api/contacts.json?
access_token=7c6f24f5 HTTP/1.1
Host: 726e3b99.ngrok.io
User-Agent: curl/7.43.0
Accept: */*
X-Forwarded-Proto: https
X-Forwarded-For: 2.139.235.79

 while sending to client, client: 127.0.0.1, server: pili-virtualbox, request: "GET /api/contacts.json?
access_token=7c6f24f5 HTTP/1.1", subrequest: "/threescale_authrep", upstream:
"https://54.83.62.94:443/transactions/oauth_authrep.xml?
provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f24f5", host:
"726e3b99.ngrok.io"
2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:8: <?xml version="1.0" encoding="UTF-
8"?><error code="access_token_invalid">access_token "7c6f24f5" is invalid: expired or never
defined</error> while sending to client, client: 127.0.0.1, server: pili-virtualbox, request: "GET
/api/contacts.json?access_token=7c6f24f5 HTTP/1.1", subrequest: "/threescale_authrep", upstream:
"https://54.83.62.94:443/transactions/oauth_authrep.xml?
provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f24f5", host:
"726e3b99.ngrok.io"
```

第一个条目(**2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:7:**)打印发送到 3scale 的请求标头，本例中为 Host、User-Agent、Accept、X-Forwarded-Proto 和 X-Forwarded-For。

第二个条目(**2016/05/05 14:24:33 [] 7238#0: *57 [lua] body_filter_by_lua:8:**)从 3scale 打印出响应，本例中为 **<error code="access_token_invalid">access_token "7c6f24f5" is invalid: expired or never defined</error>**。

两者都将打印出原始请求(**GET /api/contacts.json?access_token=7c6f24f5**)和 subrequest 位置(**/threescale_authrep**)以及上游请求（**upstream: "https://54.83.62.94:443/transactions/threescale_authrep.xml?provider_key=REDACTED&service_id=REDACTED&usage[hits]=1&access_token=7c6f24f5"**）通过这个最后一个值，您可以查看 3scale IP 中的哪些已被解决，以及向 3scale 发出的确切请求。

12.3.5. 3scale API 管理

12.3.5.1. 3scale API 管理是否返回错误？

3scale 也可能可用，但会向您的 API 网关返回错误，防止调用进入您的 API。尝试直接在 3scale 中发出授权调用并检查响应。如果遇到错误，请检查 #troubleshooting-api-error-codes[Error Codes] 部分以查看问题所在。

12.3.5.2. 使用 3scale API 管理调试标头

您还可以通过调用 **X-3scale-debug 标头来打开 3scale** 调试标头，例如：

curl -v -X GET "https://api.example.com/endpoint?user_key" X-3scale-debug: YOUR_SERVICE_TOKEN

这将返回带有 API 响应的以下标头：

```
X-3scale-matched-rules: /, /api/contacts.json
< X-3scale-credentials: access_token=TOKEN_VALUE
< X-3scale-usage: usage[hits]=2
< X-3scale-hostname: HOSTNAME_VALUE
```

12.3.5.3. 检查集成错误

您还可以检查管理门户中的集成错误，以检查是否报告流量到 3scale。请参阅 https://YOUR_DOMAIN-admin.3scale.net/apiconfig/errors。

集成错误的原因之一是在标头中发送凭据，并附带 **underscores_in_headers** 指令未在 server 块中启用。

12.3.6. 客户端 API 网关

12.3.6.1. API 网关是否可从公共互联网访问？

尝试将浏览器定向到网关服务器的 IP 地址（或域名）。如果此操作失败，请确保您已在相关端口上打开防火墙。

12.3.6.2. 客户端是否可访问 API 网关？

如果可能，尝试使用前面概述的方法之一（telnet、curl 等）从客户端连接到 API 网关。如果连接失败，问题在于这两者之间的网络。

否则，您应该继续对 API 发出调用的客户端进行故障排除。

12.3.7. 客户端

12.3.7.1. 使用其他客户端测试相同的调用

如果请求没有返回预期结果，请使用不同的 HTTP 客户端进行测试。例如，如果您使用 Java HTTP 客户端调用 API，您会看到错误，使用 cURL 进行交叉检查。

您还可以通过客户端和网关之间的代理调用 API，以捕获客户端发送的确切参数和标头。

12.3.7.2. 检查客户端发送的流量

使用 Wireshark 等工具来查看客户端发出的请求。这样，您可以识别客户端是否在调用 API 以及请求的详细信息。

12.4. ACTIVEDOCS 问题

当您从命令行调用 API 时，当通过 ActiveDocs 时，调用 会失败。

要启用 ActiveDocs 调用，我们通过我们的代理将这些内容发送出来。此代理会添加特定的标头，如果不需要它们，则有时会导致 API 上出现问题。要识别是否是这种情况，请尝试以下步骤：

12.4.1. 使用 petstore.swagger.io

swagger 在 petstore.swagger.io 处提供托管的 swagger-ui，您可以通过最新版本的 swagger-ui 测试您的 Swagger 规格和 API。如果 swagger-ui 和 ActiveDocs 都以同样方式失败，您可以排除 ActiveDocs 或 ActiveDocs 代理出现的任何问题，并将故障排除集中到您自己的规格上。或者，您可以检查 swagger-ui GitHub 存储库，以了解当前版本的 swagger-ui 是否存在已知问题。

12.4.2. 检查防火墙是否允许来自 ActiveDocs 代理的连接

我们建议您不要使用您的 API 将客户端的 IP 地址列入白名单。ActiveDocs 代理使用浮动 IP 地址来实现高可用性，目前没有机制通知对这些 IP 的任何更改。

12.4.3. 使用不正确的凭证调用 API

要确定 ActiveDocs 代理是否正常工作，一种方法是使用无效凭证调用您的 API。这将帮助您确认或排除 ActiveDocs 代理和 API 网关中的任何问题。

如果您从 API 调用中获取 403 代码（或者来自网关上为无效凭证配置的代码），问题在于您的 API，因为这些调用将到达您的网关。

12.4.4. 比较调用

要识别 ActiveDocs 与 ActiveDocs 外部调用之间的标头和参数差异，请通过 APItools 内部或运行作用域等服务运行调用。这将允许您在将 HTTP 调用发送到 API 之前检查并比较 HTTP 调用。然后，您将能够识别可能导致问题的请求中潜在的标头和/或参数。

12.5. 登录 NGINX

有关此内容的综合指南，请参阅 NGINX 日志和监控 文档。

12.5.1. 启用调试日志

要了解更多有关启用调试日志的信息，请参阅 NGINX 调试日志文档。

12.6. 3SCALE 错误代码

要重复检查 3scale Service Management API 端点返回的错误代码，请查看 **3scale API 文档** 页面，按照以下步骤执行：

1. 单击位于管理门户右上角的问号(?)图标。
2. 选择 **3scale API 文档**。

以下是 3scale 返回的 HTTP 响应代码列表，以及返回它们的条件：

- **400**：错误请求。这可能是因为：
  - 无效的编码
  - 有效负载过大
  - 内容类型无效（用于 POST 调用）。**Content-Type** 标头的有效值为：**application/x-www-form-urlencoded**、**multipart/form-data** 或空标头。

- 403:
  - 凭证无效
  - 将正文数据发送到 3scale 的 GET 请求
- 404： 没有引用的实体，如应用程序、指标等。
- 409:
  - 超过用量限制
  - 应用程序未激活
  - 应用程序密钥无效或缺失（用于 **app_id/app_key** 身份验证方法）
  - 不允许或缺少推荐器（当启用和需要引用器过滤器时）
- 422： 缺少所需参数

大多数错误响应也将包含 XML 正文，其中含有计算机可读的错误类别和人类可读的说明。

使用标准 API 网关配置时，任何与 3scale 提供的 200 不同的返回代码都可能会通过以下代码之一响应客户端：

- 403
- 404