Red Hat 3scale API Management 2.14

安装 Red Hat 3scale API Management

安装和配置 3scale API 管理。

Last Updated: 2024-06-29

Red Hat 3scale API Management 2.14 安装 Red Hat 3scale API Management

安装和配置 3scale API 管理。

法律通告

摘要

本指南提供用于安装和配置 3scale API 管理的信息。

目录

前言

本指南将帮助您安装和配置 3scale。

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。

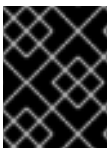
流程

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 的 REGISTRY 服务帐户

要在带有 Red Hat 3scale API Management 2.14 的共享环境中使用 **registry.redhat.io** 中的容器镜像，您必须使用 *Registry Service* 帐户而不是单独的*用户的客户门户网站凭证*。



重要

在通过操作器（operator）在 OpenShift 上部署 3scale 之前，请按照本章中所述的步骤，因为部署使用 registry 身份验证。

要创建和修改 registry 服务帐户，请执行以下部分中所述的步骤：

- 创建 registry 服务帐户
- 配置容器 registry 身份验证
- 修改 registry 服务帐户

1.1. 创建 REGISTRY 服务帐户

要创建 registry 服务帐户，请按照以下步骤操作。

流程

1. 进入 Registry Service Accounts 页面并登录。
2. 点 **New Service Account**。
3. 在 *Create a New Registry Service Account* 页面上填写表单。
   a. 为 *服务帐户* 添加名称。
   备注：您将在表单字段前面看到一个固定长度、随机生成的数字字符串。
   b. 输入 *描述*。
   c. 点 **Create**。
4. 切回到您的 *服务帐户*。
5. 点 *您创建的* 服务帐户。
6. 记录用户名，包括前缀字符串，如 **12345678|username** 和您的密码。此用户名和密码将用于登录 **registry.redhat.io**。

注意

Token Information 页面中提供了相应的选项卡，用于显示如何使用身份验证令牌。例如，*Token Information* 选项卡显示格式为 **12345678|username** 的用户名，其下的密码字符串为。

1.2. 配置容器 REGISTRY 身份验证

作为 3scale 管理员，在 OpenShift 中部署 3scale 之前，使用 **registry.redhat.io** 配置身份验证。

先决条件

- 具有管理员凭证的 Red Hat OpenShift Container Platform (OCP) 帐户。
- 已安装 OpenShift **oc** 客户端工具。如需了解更多详细信息，请参阅 OpenShift CLI 文档。

流程

1. 以管理员身份登录您的 OpenShift 集群：

```
$ oc login -u <admin_username>
```

2. 打开您要在其中部署 3scale 的项目：

```
$ oc project your-openshift-project
```

3. 使用您的红帽客户门户网站帐户创建一个 **docker-registry** secret，将 **threescale-registry-auth** 替换为要创建的 secret:

```
$ oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username="customer_portal_username" \
  --docker-password="customer_portal_password" \
  --docker-email="email_address"
```

您将看到以下输出：

```
secret/threescale-registry-auth created
```

4. 将机密链接到您的服务帐户，以使用机密拉取镜像。服务帐户名称必须与 OpenShift 容器集使用的名称匹配。这个示例使用 **default** 服务帐户：

```
$ oc secrets link default threescale-registry-auth --for=pull
```

5. 将 secret 链接到 **builder** 服务帐户，以使用 secret 推送和拉取构建镜像：

```
$ oc secrets link builder threescale-registry-auth
```

其他资源

- 红帽容器镜像身份验证
- Red Hat registry 服务帐户

1.3. 修改 REGISTRY 服务帐户

您可以使用表中每个身份验证令牌右侧的弹出菜单，从 *Registry Service Account* 页面编辑或删除服务帐户。

警告

重新生成或删除 *服务帐户* 会影响使用该令牌对 **registry.redhat.io** 进行身份验证和检索内容的系统。

每个功能的描述如下：

- **重新生成令牌：**允许授权用户重置与*服务帐户*关联的密码。
  备注：您无法修改*服务帐户*的用户名。
- **更新描述：**允许授权的用户更新*服务帐户*的描述。
- **删除帐户：**允许授权的用户删除*服务帐户*。

1.4. 其他资源

- Red Hat Container Registry 身份验证
- 启用了身份验证的红帽 registry

第 2 章 在 OPENSHIFT 上安装 3SCALE API MANAGEMENT

本节介绍了在 OpenShift 中部署 Red Hat 3scale API Management 2.14 的步骤。

用于内部部署的 3scale 解决方案包括：

- 两个应用程序编程接口(API)网关：嵌入式 APIcast。
- 一个带有持久性存储的 3scale 管理门户和开发者门户。

注意

- 在部署 3scale 时，必须首先将 registry 身份验证配置到 Red Hat Container registry。请参阅配置容器 registry 身份验证。
- 3scale Istio 适配器是一个可选适配器，允许在 Red Hat OpenShift Service Mesh 中标记运行的服务，并将该服务与 3scale 管理集成。如需更多信息，请参阅 3scale 适配器 文档。

先决条件

- 您必须为 UTC（协调世界时间）配置 3scale 服务器。
- 使用创建 registry 服务帐户中的步骤创建用户凭证。

要在 OpenShift 上安装 3scale，请执行以下小节中介绍的步骤：

- 在 OpenShift 上安装 3scale API 管理的系统要求
- 使用 Operator 部署 3scale API 管理
- 使用 Operator 进行 3scale API 管理的外部数据库
- 使用 Operator 在 OpenShift 上 3scale API Management 的部署配置选项
- 使用 Oracle 作为系统数据库的操作器安装 3scale API 管理
- 常见 3scale API 管理安装问题的故障排除

2.1. 在 OPENSHIFT 上安装 3SCALE API 管理的系统要求

本节列出了在 OpenShift 上安装 Red Hat 3scale API Management 的系统要求。

2.1.1. 环境要求

3scale 需要一个在支持的配置中指定的环境。

注意

对持久性卷的要求因不同的部署类型而异。当使用外部数据库部署时，不需要持久性卷。对于某些部署类型，Amazon S3 存储桶可以充当持久性卷的替代。如果使用本地文件系统存储，请考虑特定的部署类型及其持久性卷的相关要求。

持久性卷（PV）

- 4 RWO (ReadWriteOnce)用于 Redis、MySQL 和 System-searchd 持久性的持久性卷。
- 1 个用于开发人员门户内容和 System-app 资源的 RWX (ReadWriteMany)持久性卷。

将 RWX 持久卷配置为可写入组。如需支持所需访问模式的持久卷类型列表，请参阅 OpenShift 文档。

注意

3scale 仅支持 RWX 卷的网络文件系统 (NFS)。

对于 IBM Power (ppc64le) 和 IBM Z (s390x)，使用以下命令置备本地存储：

Storage

- NFS

如果您在内容管理系统(CMS)存储中使用 Amazon Simple Storage Service(Amazon S3)存储桶：

持久性卷（PV）

- 3 RWO (ReadWriteOnce)用于 Redis 和 MySQL 持久性的持久性卷。

Storage

- 1 Amazon S3 存储桶
- NFS

2.1.2. 硬件要求

硬件要求取决于您的使用需求。红帽建议您测试和配置您的环境，以满足您的特定要求。以下是在 OpenShift 中为 3scale 配置环境时的建议：

- 计算优化的节点用于云环境（AWS c4.2xlarge 或 Azure Standard_F8）上的部署。
- 如果内存要求超过您当前节点的可用 RAM，则非常大型安装可能需要单独节点（AWS M4 系列或 Azure Av2 系列）。
- 路由和计算任务之间分隔的节点。
- 用于 3scale 特定任务的专用计算节点.

其他资源

- 了解持久性存储

2.2. 在 OPENSHIFT 上安装 3SCALE API MANAGEMENT OPERATOR

注意

3scale 支持 OpenShift Container Platform(OCP)的最后两个通用版本(GA)。如需更多信息，请参阅 Red Hat 3scale API 管理支持的配置 页面。

本文档演示了如何：

- 创建一个新项目。
- 部署红帽 3scale API 管理实例。
- 通过 Operator Lifecycle Manager(OLM)安装 3scale Operator。
- 部署 Operator 后，部署自定义资源。

先决条件

- 使用具有管理员权限的帐户访问受支持的 OpenShift Container Platform 4 集群版本。
  - 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。

警告

在另一个新创建的空*项目*中部署 3scale 操作器和自定义资源定义(CRD)。如果您将其部署到包含基础架构的现有项目中，则可能会更改或删除现有的元素。

要在 OpenShift 上安装 3scale 操作器，请执行以下部分中所述的步骤：

- 创建新的 OpenShift 项目
- 使用 OLM 安装和配置 3scale API 管理 Operator

2.2.1. 创建新的 OpenShift 项目

此流程说明了如何创建名为 **3scale-project** 的新 OpenShift 项目。将此项目名称替换为您自己的项目名称。

流程

要创建新 OpenShift 项目，请执行以下操作：

- 使用字母数字字符和短划线表示有效的名称。例如，运行以下命令来创建 **3scale-project** ：

```
$ oc new-project 3scale-project
```

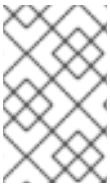
这会创建新的 *OpenShift 项目*，其中将安装 operator、*APIManager* 自定义资源(CR)和 *Capabilities* 自定义资源。Operator 通过该项目中的 OLM 管理自定义资源。

2.2.2. 使用 OLM 安装和配置 3scale API Management Operator

使用 Operator Lifecycle Manager (OLM)通过 OCP 控制台中的 OperatorHub 在 OpenShift Container Platform (OCP) 4.12 （或以上）集群中安装 3scale Operator。您可以使用以下安装模式安装 3scale Operator：

- 集群范围内的 Operator 在集群中的所有命名空间中可用。

- 集群中的特定命名空间



注意

如果您在受限网络或断开连接的集群中使用 OpenShift Container Platform，Operator Lifecycle Manager 将不再使用 OperatorHub。按照在 受限网络中标题为"使用 Operator Lifecycle Manager"的指南中设置和使用 OLM 的说明。

先决条件

- 您必须在您在 创建新的 OpenShift 项目中定义的项目中安装并部署 3scale operator。

流程

1. 在 OpenShift Container Platform 控制台中，使用具有管理员特权的帐户登录。
2. 点 **Operators > OperatorHub**。
3. 在 **Filter by keyword** 框中，键入 *3scale operator* 来查找 *Red Hat Integration - 3scale* 。
4. 点 *Red Hat Integration - 3scale* 。此时会显示有关 Operator 的信息。
5. 阅读有关 Operator 的信息，再点 **Install**。此时会打开 *Install Operator* 页面。
6. 在 **Install Operator** 页面中，在 **Update channel** 部分选择要更新的频道。
7. 在 *Installation mode* 部分中，选择安装 Operator 的位置。
    a. **All namespaces on the cluster(default)** - operator 在集群的所有命名空间中可用。
    b. **A specific namespace on the cluster**- operator 只能在您选择的集群的特定单一命名空间中可用。
8. 点 **Install**。
9. 安装完成后，系统会显示确认信息，指示 Operator 已准备就绪。
10. 验证 3scale operator ClusterServiceVersion (CSV) 是否已正确安装。另外，检查它报告 Operator 的安装是否成功：
    - 点 **Operators > Installed Operators**。
    - 点 **Red Hat Integration - 3scale** operator。
    - 在 **Details** 选项卡中，向下滚动到 **Conditions** 部分，其中 **Succeeded** 条件应该在 **Reason** 列中读取 **InstallSucceeded**。

除了指定的步骤外，在受限网络中使用 OCP 时，创建一个在 3scale 开发人员门户中使用的允许域的列表。请考虑以下示例：

- 您打算添加到开发人员门户的任何链接。
- 通过第三方 SSO 提供程序（如 GitHub）进行单点登录(SSO)集成。
- 计费。
- 触发外部 URL 的 Webhook。

2.2.2.1. 在断开连接的环境中的限制

以下列表概述了 3scale 2.14 在断开连接的环境中的当前限制：

- GitHub 登录 Developer Portal 不可用。
- 支持链接不可用。
- 到外部文档的链接无法正常运行。
- Developer Portal 中的 OpenAPI 规范(OAS)验证器不正常运行，会影响外部服务的链接。
- 在 **ActiveDocs** 的产品 *概述* 页面中，到 OAS 的链接不起作用。
  - 在创建新的 ActiveDocs 规格时，还需要检查 *Skip swagger 验证*选项。

其他资源

- OpenShift Container Platform 文档
- 在受限网络中使用 Operator Lifecycle Manager
- 为断开连接的安装 mirror 镜像
- Red Hat 3scale API Management Platform 支持的配置

2.2.3. 使用 OLM 升级 3scale API Management Operator

要将 3scale Operator 从单一命名空间升级到基于 Operator 部署的所有命名空间中集群范围的安装，您必须从命名空间中删除 3scale Operator，然后在集群中重新安装 Operator。

集群管理员可以使用 Web 控制台从所选命名空间中删除已安装的 Operator。卸载 Operator 不会卸载现有的 3scale 实例。

从命名空间中卸载 3scale Operator 后，您可以使用 OLM 在集群范围模式下安装 Operator。

先决条件

- 3scale 管理员权限或具有命名空间删除权限的 OpenShift 角色。

流程

1. 在 OpenShift Container Platform 控制台中，使用具有管理员特权的帐户登录。
2. 点 **Operators > OperatorHub**。此时会显示安装的 Operator 页面。
3. 在 *Filter by name* 中输入 *3scale* 以查找 Operator 并点它。
4. 在 *Operator Details* 页面中，从 **Actions** 下拉菜单中选择 **Uninstall Operator**，将其从特定命名空间中删除。
5. 此时会显示 *Uninstall Operator?* 对话框，提醒您：

> Removing the operator will not remove any of its custom resource definitions or managed resources. If your operator has deployed applications on the cluster or configured off-cluster resources, these will continue to run and need to be cleaned up manually.

> This action removes the operator as well as the Operator deployments and pods, if any. Any operands and resources managed by the operator, including CRDs and CRs, are not removed. The web console enables dashboards and navigation items for some operators. To remove these after uninstalling the operator, you might need to manually delete the operator CRDs.

6. 选择 **Uninstall**。此 operator 会停止运行，并且不再接收更新。
7. 在 OpenShift Container Platform 控制台中点 **Operators > OperatorHub**。
8. 在 **Filter by keyword** 框中，键入 *3scale operator* 来查找 *Red Hat Integration - 3scale* 。
9. 点 *Red Hat Integration - 3scale* 。此时会显示有关 Operator 的信息。
10. 点 **Install**。此时会打开 *Install Operator* 页面。
11. 在 **Install Operator** 页面中，在 **Update channel** 部分选择要更新的频道。
12. 在 *Installation mode* 部分中，选择 **All namespaces on the cluster(default)**。Operator 将在集群中的所有命名空间中可用。
13. 点 **Subscribe**.*3scale operator* 详情页面会显示，您可以查看 *Subscription Overview*。
14. 确认订阅**升级状态**显示为 **Up to date**。
15. 验证是否显示了 3scale operator ClusterServiceVersion(CSV)。

其它资源

- 在 OpenShift 上安装 3scale API Management Operator

2.2.3.1. 配置微版本的自动应用程序

警告

如果您使用外部 Oracle 数据库，请将 3scale 更新策略设置为 *Manual*。使用外部 Oracle 数据库时，会手动更新数据库和 **.spec.system.image**。*Automatic* 设置不会更新 **.spec.system.image**。请参阅 Migrating 3scale 指南，以使用外部 Oracle 数据库更新基于 Operator 的安装。

要获得自动更新，3scale 操作器必须将批准策略设置为 *Automatic*。这允许它自动应用微版本更新。下面描述了*自动*和*手动*设置之间的区别，并概述了从一个设置切换到另一个流程的步骤。

自动和手动：

- 在安装过程中，*自动*设置是所选选项。随着新的更新可用，就会安装新的更新。您可以在安装过程中或之后更改。
- 如果您在安装过程中或以后选择了 *Manual* 选项，会在有可用情况下收到更新。接下来，您必须批准 *Install Plan*，并自行应用。

流程

1. 点 **Operators > Installed Operators**。
2. 从 *Installed Operators* 列表中，单击 **Red Hat Integration - 3scale**。
3. 点 **Subscription** 标签页。在 *Subscription Details* 标题下，您将看到子标题 *Approval*。
4. 点 *Approval* 下的链接。默认情况下，链接设置为 **Automatic**。将弹出带有标题 *Change Update Approval Strategy* 的模态。
5. 选择您的首选选项：*Automatic （默认）* 或 *Manual*，然后点 **Save**。

其他资源

- 在命名空间中安装 Operator

2.3. 在 OPENSHIFT 上安装 APICAST OPERATOR

本指南提供通过 OpenShift Container Platform(OCP)控制台安装 APIcast Operator 的步骤。

先决条件

- OCP 4.x 或更高版本，具有管理员特权。

流程

1. 在 **Projects > Create Project** 中创建新项目 **operator-test**。
2. 点 **Operators > OperatorHub**。
3. 在 **Filter by keyword** 框中，键入 *apicast operator* 来查找 *Red Hat Integration - 3scale APIcast 网关*。
4. 点 *Red Hat Integration - 3scale APIcast 网关* 。此时会显示有关 APIcast operator 的信息。
5. 点 *Install*。*Create Operator Subscription* 页面会打开。
6. 点 *Install* 接受 *Create Operator Subscription* 页面中的所有默认选择。

注意

您可以选择不同的 Operator 版本和安装模式，如集群范围的或特定于命名空间的选项。每个集群只能有一个集群范围的安装。

   a. 订阅 **升级状态** 显示为 **Up to date**。

7. 点 **Operators > Installed Operators** 来验证 APIcast operator *ClusterServiceVersion* (CSV)状态是否在 **operator-test** 项目中显示为 *InstallSucceeded*。

2.4. 使用 OPERATOR 部署 3SCALE API 管理

本节介绍了使用 APIManager 自定义资源(CR)通过 3scale 操作器安装和部署 3scale 解决方案。

注意

- 自 3scale 2.6 起，已删除 通配符路由。
  - 这个功能由 Zync 在后台处理。
- 创建、更新或删除 API 提供程序时，路由会自动反映这些更改。

先决条件

- 配置容器 registry 身份验证
- 要确保您收到 3scale 的微版本的自动更新，您必须在 3scale operator 中启用自动批准功能。*自动* 是默认的批准设置。要随时根据您的特定需求更改，请使用 配置微版本的自动应用程序 的步骤。
- 首先使用操作器部署 3scale API 管理，要求您按照在 OpenShift 上安装 3scale API Management operator 中的步骤进行操作。
- OpenShift 容器平台 4.x.
  - 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。

按照以下步骤，使用 Operator 部署 3scale：

- 部署 APIManager 自定义资源
- 获取管理门户 URL
- 获取 APIManager 管理门户和主管理门户凭证
- 使用 Operator 进行 3scale API 管理的外部数据库

2.4.1. 部署 APIManager 自定义资源

注意

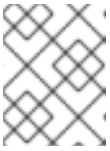
如果您决定使用 Amazon Simple Storage Service (Amazon S3)，请参阅 Amazon Simple Storage Service 3scale API Management fileStorage 安装。

Operator 会监视 APIManager CR，并部署在 APIManager CR 中指定的所需的 3scale 解决方案。

流程

1. 点 **Operators > Installed Operators**。
   a. 从 *Installed Operators* 列表中，点 *Red Hat Integration - 3scale* 。
2. 点 *API Manager* 选项卡。
3. 点 **Create APIManager**。
4. 清除示例内容，并将以下 *YAML* 定义添加到编辑器中，然后单击 **Create**。

- 在 3scale 2.8 之前，您可以通过将 **HighAvailability** 字段设置为 **true** 来配置自动添加副本。从 3scale 2.8 开始，对副本的添加会通过 APIManager CR 中的 replicas 字段来控制，如下例所示。



注意

wildcardDomain 参数的值必须是解析到 OpenShift Container Platform (OCP) 路由器地址的有效域名。例如 **apps.mycluster.example.com**。

- 具有最低要求的 APIManager CR：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-sample
spec:
  wildcardDomain: example.com
```

- 配置副本的 APIManager CR：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-sample
spec:
  system:
    appSpec:
      replicas: 1
    sidekiqSpec:
      replicas: 1
  zync:
    appSpec:
      replicas: 1
    queSpec:
      replicas: 1
  backend:
    cronSpec:
      replicas: 1
    listenerSpec:
      replicas: 1
    workerSpec:
      replicas: 1
  apicast:
    productionSpec:
      replicas: 1
    stagingSpec:
      replicas: 1
  wildcardDomain: example.com
```

2.4.2. 获取管理门户 URL

当您使用 operator 部署 3scale 时，会创建一个默认租户，它有一个固定的 URL: **3scale-admin.${wildcardDomain}**。

3scale 控制面板显示租户的新门户 URL。例如，如果 *<wildCardDomain>* 为 **3scale-project.example.com**，则管理门户 URL 为 **https://3scale-admin.3scale-project.example.com**。

wildcardDomain 是您在安装过程中提供的 *<wildCardDomain>* 参数。使用这个命令在浏览器中打开这个唯一 URL：

```
$ xdg-open https://3scale-admin.3scale-project.example.com
```

另外，您还可以在 *MASTER 门户 URL* 上创建新的租户：**master.${wildcardDomain}**。

2.4.3. 获取 APIManager 管理门户和主管理门户凭证

要在基于 operator 的部署后登录到 3scale 管理门户或 Master 管理门户，您需要每个单独的门户的凭证。要获得以下凭证：

1. 运行以下命令来获取 Admin Portal 凭证：

```
$ oc get secret system-seed -o json | jq -r .data.ADMIN_USER | base64 -d
$ oc get secret system-seed -o json | jq -r .data.ADMIN_PASSWORD | base64 -d
```

   a. 以 Admin Portal 管理员身份登录，以验证这些凭据是否正常工作。

2. 运行以下命令来获取主管理门户凭证：

```
$ oc get secret system-seed -o json | jq -r .data.MASTER_USER | base64 -d
$ oc get secret system-seed -o json | jq -r .data.MASTER_PASSWORD | base64 -d
```

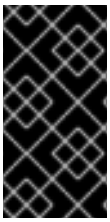
   a. 以 Master Admin Portal 管理员身份登录，以验证这些凭据是否正常工作。

其他资源

参考文档.

另外，您还可以在 *MASTER 门户 URL* 上创建新的租户：**master.${wildcardDomain}**。

2.4.4. 使用 Operator 进行 3scale API 管理的外部数据库



重要

从 Red Hat 3scale API 管理部署外部化数据库时，这意味着与应用程序分离，并对在数据库一级的服务中断有一定的弹性。服务中断的恢复取决于您托管数据库的基础架构或平台供应商提供的服务级别协议 (SLA)。3scale 不提供此功能。有关您选择的部署提供的数据库外部化的详情，请查看相关的文档。

当您使用 operator 为 3scale 使用外部数据库时，如果一个或多个数据库失败，则目的是提供不间断的运行时间。

如果您在基于 3scale Operator 的部署中使用外部数据库，请注意以下几点：

- 在外部配置和部署 3scale 关键数据库。关键数据库包括系统数据库、系统 redis 和后端 redis 组件。确保您以使其具有高可用性的方式部署和配置这些组件。
- 在部署 3scale 前，通过创建对应的 Kubernetes secret 来为 3scale 指定到这些组件的连接端点。
  - 如需更多信息，请参阅外部数据库安装。

    - 如需有关非数据库部署配置的更多信息，请参阅启用 Pod Disruption Budgets。

- 在 **APIManager** CR 中，设置 **.spec.externalComponents** 属性来指定系统数据库、系统 redis 和 backend redis 是外部：

```
externalComponents:
  backend:
    redis: true
  system:
    database: true
    redis: true
  zync:
    database: true
```

另外，如果您希望 zync 数据库高度可用，为了避免 zync 可能会在重启时丢失队列作业数据，请注意：

- 在外部部署和配置 zync 数据库。确保以高可用性方式部署和配置数据库。
- 通过在部署 3scale 前创建对应的 Kubernetes secret，指定到 3scale 的 zync 数据库的连接端点。
    - 请参阅 Zync 数据库 secret。
- 通过将 **APIManager** CR 中的 **.spec.externalComponents.zync.database** 属性设为 **true** 来配置 3scale，以指定 zync 数据库是一个外部数据库。

2.5. 使用 OPERATOR 在 OPENSHIFT 上 3SCALE API MANAGEMENT 的部署配置选项

重要

本备注中包含的链接仅出于方便用户而提供。红帽没有审阅链接的内容，并不对其内容负责。包含任何指向外部网站的链接并不表示红帽认可该网站或其实体、产品或服务。您同意红帽对因您使用（或依赖）外部网站或内容而导致的任何损失或费用不承担任何责任。

本节介绍了使用操作器在 OpenShift 上红帽 3scale API 管理的部署配置选项。

先决条件

- 配置容器 registry 身份验证
- 首先使用 Operator 部署 3scale API 管理需要遵循在 OpenShift 上安装 3scale API ManagementOperator 中的步骤
- OpenShift Container Platform 4.x
    - 在 OpenShift 集群中具有管理员特权的用户帐户。

2.5.1. 为嵌入式 APIcast 配置代理参数

作为 3scale 管理员，您可以为嵌入式 APIcast staging 和 production 配置代理参数。本节提供了在 APIManager 自定义资源(CR)中指定代理参数的参考信息。换句话说，您使用 3scale 操作器，一个 APIManager CR，用于在 OpenShift 中部署 3scale。

您可以在首次部署 APIManager CR 时指定这些参数，或者您可以更新部署的 APIManager CR，Operator 会协调更新。请参阅部署 APIManager 自定义资源。

嵌入式 APIcast 有四个与代理相关的配置参数：

- **allProxy**
- **httpProxy**
- **httpsProxy**
- **noProxy**

allProxy

allProxy 参数指定在请求没有指定协议相关的代理时，用来连接到服务的 HTTP 或 HTTPS 代理。

设置代理后，通过将 **allProxy** 参数设置为代理的地址来配置 APIcast。代理不支持身份验证。换句话说，APIcast 不会将经过身份验证的用户发送到代理。

allProxy 参数的值是一个字符串，没有默认值，且不需要该参数。使用此格式设置 **spec.apicast.productionSpec.allProxy** 参数或 **spec.apicast.stagingSpec.allProxy** 参数：

<scheme>://<host>:<port>

例如：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      allProxy: http://forward-proxy:80
    stagingSpec:
      allProxy: http://forward-proxy:81
```

httpProxy

httpProxy 参数指定用于连接 HTTP 服务的 HTTP 代理。

设置代理后，通过将 **httpProxy** 参数设置为代理的地址来配置 APIcast。代理不支持身份验证。换句话说，APIcast 不会将经过身份验证的用户发送到代理。

httpProxy 参数的值是一个字符串，没有默认值，且不需要该参数。使用此格式设置 **spec.apicast.productionSpec.httpProxy** 参数或 **spec.apicast.stagingSpec.httpProxy** 参数：

http://<host>:<port>

例如：

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

```
  apicast:
    productionSpec:
      httpProxy: http://forward-proxy:80
    stagingSpec:
      httpProxy: http://forward-proxy:81
```

httpsProxy

httpsProxy 参数指定用于连接服务的 HTTPS 代理。

设置代理后，通过将 **httpsProxy** 参数设置为代理的地址来配置 APIcast。代理不支持身份验证。换句话说，APIcast 不会将经过身份验证的用户发送到代理。

httpsProxy 参数的值是一个字符串，没有默认值，且不需要该参数。使用此格式设置 **spec.apicast.productionSpec.httpsProxy** 参数或 **spec.apicast.stagingSpec.httpsProxy** 参数：

https://<host>:<port>

例如：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      httpsProxy: https://forward-proxy:80
    stagingSpec:
      httpsProxy: https://forward-proxy:81
```

noProxy

noProxy 参数指定以逗号分隔的主机名和域名列表。当请求包含其中一个名称时，APIcast 不会代理请求。

如果您需要停止对代理的访问，例如在维护操作过程中，将 **noProxy** 参数设置为星号(*)。这与所有请求中指定的所有主机匹配，并有效地禁用任何代理。

noProxy 参数的值是一个字符串，没有默认值，且不需要该参数。指定以逗号分隔的字符串，以设置 **spec.apicast.productionSpec.noProxy** 参数或 **spec.apicast.stagingSpec.noProxy** 参数。例如：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  apicast:
    productionSpec:
      noProxy: theStore,company.com,big.red.com
    stagingSpec:
      noProxy: foo,bar.com,.extra.dot.com
```

其他资源

- APIcast staging 配置的自定义资源定义在一个 APIManager 自定义资源中
- APIcast 生产环境配置的自定义资源定义在一个 APIManager 自定义资源中

2.5.2. 使用 3scale API 管理 Operator 注入自定义虚拟环境

在使用嵌入式 APIcast 的 3scale 安装中，您可以使用 3scale 操作器注入自定义环境。嵌入式 APIcast 也称为受管或托管 APIcast。自定义环境定义 APIcast 适用于网关服务的所有上游 API 的行为。要创建自定义环境，请在 Lua 代码中定义全局配置。

您可在 3scale 安装前或之后注入自定义环境。在注入自定义环境并在 3scale 安装后，您可以删除自定义环境。3scale 操作器协调更改。

先决条件

- 已安装 3scale Operator。

流程

1. 编写用于定义您要注入的自定义环境的 Lua 代码。例如，以下 **env1.lua** 文件显示了 3scale 操作器为所有服务加载的自定义日志策略。

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

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

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

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

2. 从定义自定义环境的 Lua 文件创建 secret。例如：

```
$ oc create secret generic custom-env-1 --from-file=./env1.lua
```

secret 可以包含多个自定义虚拟环境。为定义自定义环境的每个文件指定 **-from-file** 选项。Operator 会加载每个自定义环境。

3. 定义一个 APIManager 自定义资源(CR)来引用您刚才创建的 secret。以下示例显示了相对于引用定义自定义环境的 secret 的内容。

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

```
wildcardDomain: <desired-domain>
apicast:
  productionSpec:
    customEnvironments:
      - secretRef:
          name: custom-env-1
  stagingSpec:
    customEnvironments:
      - secretRef:
          name: custom-env-1
```

APIManager CR 可以引用定义自定义虚拟环境的多个 secret。Operator 会加载每个自定义环境。

4. 创建添加自定义环境的 APIManager CR。例如：

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

后续步骤

您无法更新定义自定义环境的 secret 的内容。如果需要更新自定义环境，您可以执行以下操作之一：

- 推荐的选项是创建带有不同名称的 secret，并更新 APIManager CR 字段 **customEnvironments[].secretRef.name**。Operator 会触发滚动更新并加载更新的自定义环境。
- 另外，您可以更新现有的 secret，将 **spec.apicast.productionSpec.replicas** 或 **spec.apicast.stagingSpec.replicas** 设置为 0 来重新部署 APIcast，然后通过将 **spec.apicast.productionSpec.replicas** 或 **spec.apicast.stagingSpec.replicas** 设置为之前的值来再次重新部署 APIcast。

2.5.3. 使用 3scale API 管理 Operator 注入自定义策略

在使用嵌入式 APIcast 的 3scale 安装中，您可以使用 3scale 操作器来注入自定义策略。嵌入式 APIcast 也称为受管或托管 APIcast。注入自定义策略会将策略代码添加到 APIcast。然后，您可以使用以下任一策略将自定义策略添加到 API 产品策略链中：

- 3scale API
- **Product** 自定义资源 (CR)

要使用 3scale 管理门户将自定义策略添加到产品策略链中，还必须使用 **CustomPolicyDefinition** CR 注册自定义策略的 schema。自定义策略注册是只有在您要使用管理门户配置产品策略链时才需要。

您可以将自定义策略注入为 3scale 安装的一部分或之后。在注入自定义策略并在 3scale 安装后，您可以通过从 APIManager CR 中删除其规格来删除自定义策略。3scale 操作器协调更改。

先决条件

- 您需要安装或您之前安装了 3scale Operator。
- 您已定义一个自定义策略，如写您自己的策略中所述。即，已创建了定义自定义策略的 **my-policy.lua**、**apicast-policy.json** 和 **init.lua** 文件，

流程

1. 从定义一个自定义策略的文件创建 secret。例如：

```
$ oc create secret generic my-first-custom-policy-secret \
 --from-file=./apicast-policy.json \
 --from-file=./init.lua \
 --from-file=./my-first-custom-policy.lua
```

如果您有多个自定义策略，请为每个自定义策略创建一个 secret。secret 只能包含一个自定义策略。

2. 使用 3scale 操作器监控机密更改。添加 **apimanager.apps.3scale.net/watched-by=apimanager** 标签以开始 3scale operator secret 更改监控：

```
$ oc label secret my-first-custom-policy-secret apimanager.apps.3scale.net/watched-by=apimanager
```

注意

默认情况下，3scale 操作器不会跟踪对 secret 的更改。使用标签时，3scale Operator 会在您对 secret 进行更改时自动更新 APIcast 部署。这在使用 secret 的暂存和生产环境中发生。3scale 操作器不会以任何方式获取 secret 的所有权。

3. 定义一个 APIManager CR，它引用包含自定义策略的每个 secret。您可以为 APIcast staging 和 APIcast production 指定相同的 secret。以下示例显示有关引用包含自定义策略的 secret 的内容。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager-apicast-custom-policy
spec:
  apicast:
    stagingSpec:
      customPolicies:
        - name: my-first-custom-policy
          version: "0.1"
          secretRef:
            name: my-first-custom-policy-secret
        - name: my-second-custom-policy
          version: "0.1"
          secretRef:
            name: my-second-custom-policy-secret
    productionSpec:
      customPolicies:
        - name: my-first-custom-policy
          version: "0.1"
          secretRef:
            name: my-first-custom-policy-secret
        - name: my-second-custom-policy
          version: "0.1"
          secretRef:
            name: my-second-custom-policy-secret
```

APIManager CR 可以引用定义不同自定义策略的多个 secret。Operator 加载每个自定义策略。

4. 创建 APIManager CR，以引用包含自定义策略的 secret。例如：

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

后续步骤

应用 **apimanager.apps.3scale.net/watched-by=apimanager** 标签时，3scale 操作器将开始监控 secret 中的更改。现在，您可以在 secret 中修改自定义策略，Operator 将启动滚动更新，加载更新的自定义环境。

- 另外，您可以更新现有的 secret，将 **spec.apicast.productionSpec.replicas** 或 **spec.apicast.stagingSpec.replicas** 设置为 0 来重新部署 APIcast，然后通过将 **spec.apicast.productionSpec.replicas** 或 **spec.apicast.stagingSpec.replicas** 设置为之前的值来再次重新部署 APIcast。

2.5.4. 使用 3scale API 管理操作器配置 OpenTracing

在使用嵌入式 APIcast 的 3scale 安装中，您可以使用 3scale operato 来配置 OpenTracing。您可以在 stage 或生产环境或这两种环境中配置 OpenTracing。通过启用 OpenTracing，您可以在 APIcast 实例上获取更多见解和更好的可观察性。

先决条件

- 3scale Operator 已安装，或者在安装过程中。

流程

1. 在 **stringData.config** 中包含您的 OpenTracing 配置详情的 secret。这是包含您的 OpenTracing 配置详细信息的属性的唯一有效值。任何其他规格都阻止 APIcast 收到您的 OpenTracing 配置详细信息。以下示例显示了有效的 secret 定义：

```
apiVersion: v1
kind: Secret
metadata:
  name: myjaeger
stringData:
  config: |-
     {
     "service_name": "apicast",
     "disabled": false,
     "sampler": {
       "type": "const",
       "param": 1
     },
     "reporter": {
       "queueSize": 100,
       "bufferFlushInterval": 10,
       "logSpans": false,
       "localAgentHostPort": "jaeger-all-in-one-inmemory-agent:6831"
     },
     "headers": {
       "jaegerDebugHeader": "debug-id",
       "jaegerBaggageHeader": "baggage",
       "TraceContextHeaderName": "uber-trace-id",
       "traceBaggageHeaderPrefix": "testctx-"
```

```
    },
    "baggage_restrictions": {
      "denyBaggageOnInitializationFailure": false,
      "hostPort": "127.0.0.1:5778",
      "refreshInterval": 60
    }
    }
type: Opaque
```

2. 创建 secret.例如，如果您将以前的 secret 定义保存到 **myjaeger.yaml** 文件中，您将运行以下命令：

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

3. 定义一个指定 **OpenTracing** 属性的 APIManager 自定义资源(CR)。在 CR 定义中，将 **openTracing.tracingConfigSecretRef.name** 属性设置为包含您的 OpenTracing 配置详情的 secret 的名称。以下示例显示了相对于配置 OpenTracing 的内容：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: apimanager1
spec:
  apicast:
    stagingSpec:
      ...
      openTracing:
        enabled: true
        tracingLibrary: jaeger
        tracingConfigSecretRef:
          name: myjaeger
    productionSpec:
      ...
        openTracing:
          enabled: true
          tracingLibrary: jaeger
          tracingConfigSecretRef:
            name: myjaeger
```

4. 创建配置 OpenTracing 的 APIManager CR。例如，如果您在 **apimanager1.yaml** 文件中保存 APIManager CR，您将运行以下命令：

```
$ oc apply -f apimanager1.yaml
```

后续步骤

根据 OpenTracing 的安装方式，您应该在 Jaeger 服务用户界面中看到 trace。

其他资源

- APIManager 自定义资源定义

2.5.5. 使用 3scale API Management operator 在 pod 级别启用 TLS

3scale 部署两个 APIcast 实例，一个用于 production，另一个用于 staging。TLS 可以只为生产环境或只为 staging 环境启用，或为两个实例都启用 TLS。

先决条件

- 启用 TLS 的有效证书。

流程

1. 从有效证书创建 secret，例如：

```
$ oc create secret tls mycertsecret --cert=server.crt --key=server.key
```

配置会公开 APIManager 自定义资源(CR)中的 secret 引用。您可以创建 secret，然后引用 APIManager CR 中的 secret 名称，如下所示：

- Production: APIManager CR 在 **.spec.apicast.productionSpec.httpsCertificateSecretRef** 字段中公开证书。
- Staging: APIManager CR 在 **.spec.apicast.stagingSpec.httpsCertificateSecretRef** 字段中公开证书。
  另外，您还可以配置以下内容：
- **httpsPort** 表示哪个端口 APIcast 应开始侦听 HTTPS 连接。如果这与 HTTP 端口 APIcast 有冲突，则仅将此端口用于 HTTPS。
- **httpsVerifyDepth** 定义客户端证书链的最大长度。

注意

提供有效的证书和来自 APIManager CR 的引用。如果配置可以访问 **httpsPort**，但不能访问 **httpsCertificateSecretRef**，APIcast 使用嵌入的自签名证书。不建议这样做。

2. 点 **Operators > Installed Operators**。
3. 从 **Installed Operators** 列表中，点 **3scale Operator**。
4. 点 **API Manager** 选项卡。
5. 点 **Create APIManager**。
6. 在编辑器中添加以下 YAML 定义。
   a. 如果 *为生产环境* 启用，请配置以下 YAML 定义：

```
spec:
  apicast:
    productionSpec:
      httpsPort: 8443
      httpsVerifyDepth: 1
      httpsCertificateSecretRef:
        name: mycertsecret
```

   b. 如果为 *staging* 启用，请配置以下 YAML 定义：

```
spec:
  apicast:
    stagingSpec:
      httpsPort: 8443
      httpsVerifyDepth: 1
      httpsCertificateSecretRef:
        name: mycertsecret
```

7. 点 **Create**。

2.5.6. 评估部署的概念验证

以下小节描述了适用于 3scale 评估部署的概念验证的配置选项。此部署默认使用内部数据库。

重要

外部数据库的配置是生产环境的标准部署选项。

2.5.6.1. 默认部署配置

- 容器将具有 Kubernetes 资源限值和请求。
  - 这样可确保最低性能水平。
  - 它限制资源以允许外部服务和解决方案分配。
- 部署内部数据库.
- 文件存储将基于 Persistence 卷(PV)。
  - 一个系统将需要读取、写入、执行(RWX)访问模式。
  - OpenShift 配置为在请求时提供它们。
- 将 MySQL 部署为内部关系数据库。

默认配置选项适合客户的概念验证(PoC)或评估。

可以使用 APIManager 自定义资源(CR)中的特定字段值覆盖一个或多个默认配置选项。3scale 操作器允许所有可用组合。例如，3scale 操作器允许在评估模式和外部数据库模式中部署 3scale。

2.5.6.2. 评估安装

对于和评估安装，容器将没有指定 kubernetes 资源限值和请求。例如：

- 内存占用较小
- 快速启动
- 可以在笔记本电脑上运行
- 适用于售前/销售演示

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
```

```
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  resourceRequirementsEnabled: false
```

其他资源

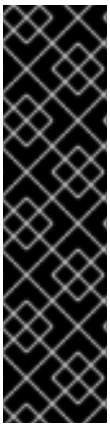
- APIManager

2.5.7. 安装外部数据库

重要

从 Red Hat 3scale API 管理部署外部化数据库时，这意味着与应用程序分离，并对在数据库一级的服务中断有一定的弹性。服务中断的恢复取决于您托管数据库的基础架构或平台供应商提供的服务级别协议 (SLA)。3scale 不提供此功能。有关您选择的部署提供的数据库外部化的详情，请查看相关的文档。

外部数据库安装适合您要提供不间断的运行时间或计划重复使用自己的数据库的生产环境。



重要

启用 3scale 外部数据库安装模式时，您可以将以下一个或多个数据库配置为 3scale 的外部：

- **backend-redis**
- **system-redis**
- **system-database** (**mysql**, **postgresql**, 或 **oracle**)
- **zync-database**

在创建 APIManager CR 以部署 3scale 之前，您必须使用 OpenShift secret 为外部数据库提供以下连接设置。

其他资源

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

2.5.7.1. 后端 Redis secret

部署两个外部 Redis 实例并填写连接设置，如下例所示：

```
apiVersion: v1
kind: Secret
metadata:
  name: backend-redis
stringData:
  REDIS_STORAGE_URL: "redis://backend-redis-storage"
  REDIS_STORAGE_SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-1.example.com:26379, redis://sentinel-2.example.com:26379"
```

```
  REDIS_STORAGE_SENTINEL_ROLE: "master"
  REDIS_QUEUES_URL: "redis://backend-redis-queues"
  REDIS_QUEUES_SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-
1.example.com:26379, redis://sentinel-2.example.com:26379"
  REDIS_QUEUES_SENTINEL_ROLE: "master"
type: Opaque
```

Secret 名称必须是 **backend-redis**。

2.5.7.2. 系统 Redis secret

部署两个外部 Redis 实例并填写连接设置，如下例所示：

```
apiVersion: v1
kind: Secret
metadata:
  name: system-redis
stringData:
  URL: "redis://system-redis"
  SENTINEL_HOSTS: "redis://sentinel-0.example.com:26379,redis://sentinel-1.example.com:26379,
redis://sentinel-2.example.com:26379"
  SENTINEL_ROLE: "master"
  NAMESPACE: ""
type: Opaque
```

Secret 名称必须是 **system-redis**。

2.5.7.3. 系统数据库 secret

注意

- *Secret* 名称必须是 **system-database**。

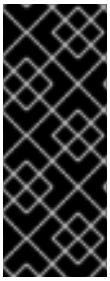
当您部署 3scale 时，系统数据库有三个替代方案。为每个替代的相关 secret 配置不同的属性和值。

- MySQL
- PostgreSQL
- Oracle 数据库

要部署 MySQL、PostgreSQL 或 Oracle Database 系统数据库 secret，请填写连接设置，如下例所示：

MySQL 系统数据库 secret

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "mysql2://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
type: Opaque
```



重要

要将 MySQL 8.0 与 3scale 2.12 搭配使用，您必须将身份验证插件设置为 **mysql_native_password**。在 MySQL 配置文件中添加以下内容：

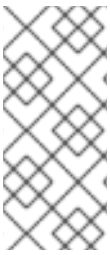
```
[mysqld]
default_authentication_plugin=mysql_native_password
```

PostgreSQL 系统数据库 secret

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "postgresql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
type: Opaque
```

Oracle 系统数据库 secret

```
apiVersion: v1
kind: Secret
metadata:
  name: system-database
stringData:
  URL: "oracle-enhanced://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
  ORACLE_SYSTEM_PASSWORD: "{SYSTEM_PASSWORD}"
type: Opaque
```



注意

- **{DB_USER}** 和 **{DB_PASSWORD}** 是常规的非系统用户的用户名和密码。
- **{DB_NAME}** 是 Oracle 数据库服务名称。
- **ORACLE_SYSTEM_PASSWORD** 是可选的，请参阅 配置数据库用户。

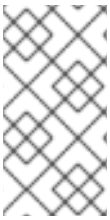
2.5.7.4. Zync 数据库 secret

在 zync 数据库设置中，当 **spec.externalComponents.zync.database** 字段被设置为 **true** 时，您必须在部署 3scale 前创建一个名为 **zync** 的 secret。在这个 secret 中，将 **DATABASE_URL** 和 **DATABASE_PASSWORD** 字段设置为指向外部 zync 数据库的值，例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: zync
stringData:
  DATABASE_URL: postgresql://<zync-db-user>:<zync-db-password>@<zync-db-host>:<zync-db-port>/zync_production
  ZYNC_DATABASE_PASSWORD: <zync-db-password>
type: Opaque
```

zync 数据库必须使用高可用性模式。

2.5.7.5. 用于部署 3scale API 管理的 APIManager 自定义资源



注意

- 当启用外部组件时，必须在部署 3scale 前为每个外部组件（**backend-redis**、**system-redis**、**system-database**、**zync**）创建一个 secret。
- 对于外部 **system-database**，仅选择一种数据库来外部化。

APIManager 自定义资源(CR)的配置取决于您的 3scale 部署外部您选择的数据库。

如果 **backend-redis**、**system-redis** 或 **system-database** 是 3scale 的外部，请填充 APIManager CR **externalComponents** 对象，如下例所示：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  externalComponents:
    system:
      database: true
```

其他资源

- 后端 redis secret
- 系统数据库 secret
- APIManager ExternalComponentsSpec
- Zync secret

2.5.8. 在 3scale API Management Operator 中启用 pod 关联性

您可以为每个组件在 3scale operator 中启用 pod 不受影响。这样可确保从集群的不同 **deploymentConfig** 中的 pod 副本分布，以便在不同的可用区 (AZ) 之间均匀平衡。

2.5.8.1. 组件级别的自定义节点关联性和容限

通过 APIManager CR 属性自定义 3scale 解决方案中的 kubernetes 关联性和 容限。然后，您可以自定义，将不同的 3scale 组件调度到 kubernetes 节点上。

例如，为 **backend-listener** 设置自定义节点关联性，为 **system-memcached** 设置自定义容限，请执行：

自定义关联性和容限

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
```

```
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: "kubernetes.io/hostname"
                operator: In
                values:
                - ip-10-96-1-105
              - key: "beta.kubernetes.io/arch"
                operator: In
                values:
                - amd64
  system:
    memcachedTolerations:
    - key: key1
      value: value1
      operator: Equal
      effect: NoSchedule
    - key: key2
      value: value2
      operator: Equal
      effect: NoSchedule
```

将以下关联性块添加到 **apicastProductionSpec** 或任何非数据库 **deploymentConfig** 中。这会使用 **preferredDuringSchedulingIgnoredDuringExecution** 添加一个软 **podAntiAffinity** 配置。调度程序将尝试在不同 AZ 的不同主机上运行此一组 **apicast-production** pod。如果无法实现，则允许它们在其它地方运行：

软 **podAntiAffinity**

```
affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchLabels:
                  deploymentConfig: apicast-production
              topologyKey: kubernetes.io/hostname
          - weight: 99
            podAffinityTerm:
              labelSelector:
                matchLabels:
                  deploymentConfig: apicast-production
              topologyKey: topology.kubernetes.io/zone
```

在以下示例中，使用 **requiredDuringSchedulingIgnoredDuringExecution** 设置硬 **podAntiAffinity** 配置。必须满足以下条件才能将 pod 调度到节点上。存在风险，例如，您将无法在具有低可用资源的集群中调度新 pod：

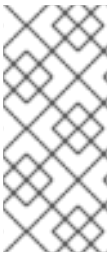
硬 podAntiAffinity

```
affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchLabels:
                  deploymentConfig: apicast-production
              topologyKey: kubernetes.io/hostname
          - weight: 99
            podAffinityTerm:
              labelSelector:
                matchLabels:
                  deploymentConfig: apicast-production
              topologyKey: topology.kubernetes.io/zone
```

其他资源

APIManager CDR 参考

2.5.9. 多个可用区中的多个集群



注意

如果失败，在将一个被动集群会变为主动模式期间会破坏服务的置备，直到过程完成为止。因为这个过程需要涉及到系统中断，请确保计划有一个维护窗口进行。

本文档重点介绍使用 Amazon Web Services (AWS) 的部署。相同的配置选项也适用于提供商管理的数据库服务提供的其他公有云供应商，例如支持多个可用性区域和多个区域。

如果要在多个 OpenShift 集群和高可用性 (HA) 区域上安装 3scale 时，您可以参考这里提供的选项。

在多个集群安装选项中，集群在主动/被动配置中工作，带有涉及几个手动步骤的故障转移过程。

2.5.9.1. 多集群安装的先决条件

在涉及使用多 OpenShift 集群的 3scale 安装中使用以下内容：

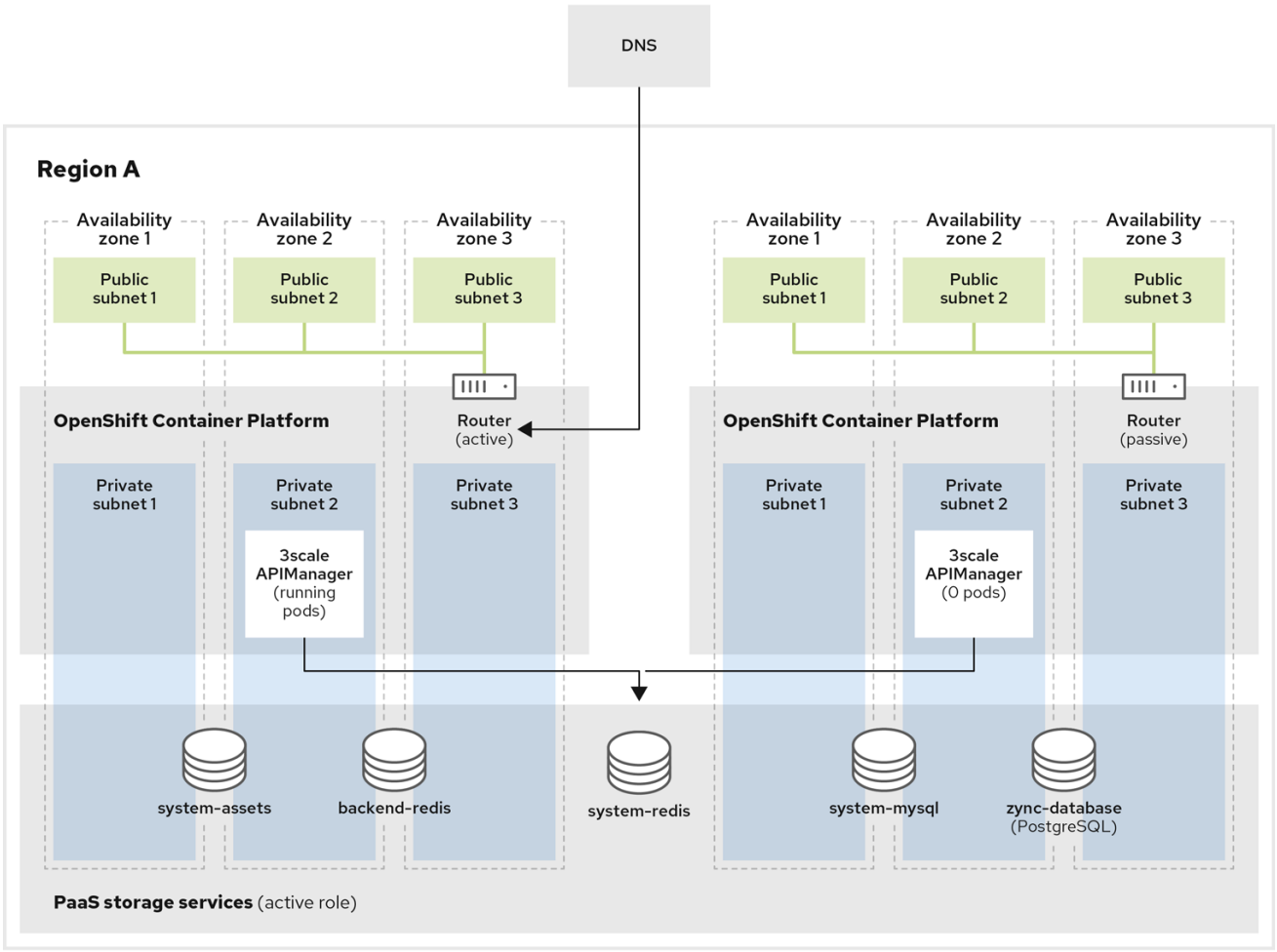
- 使用带有 APIManager 自定义资源(CR)中的 **kubernetes.io/hostname** 和 **topology.kubernetes.io/zone** 规则的 pod 等级。
- 在 APIManager CR 中使用 pod 中断预算。
- 通过多个集群的 3scale 安装必须在 APIManager CR 中使用相同的共享通配符Domain 属性规格。*在这个安装模式中不允许每个集群使用不同的域，因为存储在数据库中的信息会冲突。*
- *您必须在具有相同值的所有集群中手动部署包含凭证（如令牌和密码）的*secret*。3scale 操作器在每个集群中使用安全随机值创建它们。在这种情况下，两个集群中都需要相同的凭证。您将在 3scale 操作器文档中找到 secret 列表以及如何配置它们。以下是您必须在两个集群中镜像的 secret 列表：*
  - ***backend-internal-api***

- ***system-app***
- ***system-events-hook***
- ***system-master-apicast***
- ***system-seed***
  您必须使用 ***backend-redis***、***system-redis***、***system-database*** *和* ***zync*** *的数据库连接字符串手动部署 secret。请参阅外部数据库安装。*
- *在集群间共享的数据库都必须在所有集群中使用相同的值。*
- *如果每个集群都有自己的数据库，则必须在每个集群中使用不同的值。*

2.5.9.2. 使用共享数据库在同一区域上的主动 - 被动集群

此设置包括在同一区域中有两个或更多集群，并以主动-被动模式部署 3scale。一个集群处于活跃状态，在接收网络流量。其他集群处于待机模式，而不接收流量，因此准备假定活跃角色在活跃集群中存在故障。

在这个安装选项中，只有一个区域正在使用，数据库将在所有集群之间共享。



2.5.9.3. 配置并安装共享数据库

流程

1. 使用不同可用区 (AZ) 在同一区域中创建两个或多个 OpenShift 集群。建议至少有三个区域。
2. 创建启用了 Amazon Relational Database Service (RDS) Multi-AZ 的所有所需的 AWS ElastiCache (EC) 实例：
   a. 一个 AWS EC 用于后端 Redis 数据库
   b. 一个 AWS EC 用于系统 Redis 数据库
3. 创建启用了 Amazon RDS Multi-AZ 的所有所需的 AWS RDS 实例：
   a. 一个用于系统数据库的 AWS RDS
   b. 一个用于 Zync 数据库的 AWS RDS
4. 为系统资产配置 AWS S3 存储桶。
5. 在 AWS Route53 或 DNS 供应商中创建自定义域，并将其指向活跃集群的 OpenShift 路由器。这必须与 APIManager 自定义资源(CR)的 wildcardDomain 属性冲突。
6. 在被动集群中安装 3scale。APIManager CR 应该与上一步中使用的 CR 相同。当所有 pod 都运行时，更改 APIManager 以为所有 后端、system、zync 和 APIcast pod 部署 0 个副本。
   a. 将副本数设置为 0，以避免消耗活跃的数据库的作业。如果每个副本首次设置为 0，则部署会因为 pod 依赖项而失败。例如，检查其他人是否正在运行的 pod。首先以正常方式部署，然后将 replicas 设置为 0，如 APIManager spec 示例所示：

```
spec:
  apicast:
    stagingSpec:
      replicas: 0
    productionSpec:
      replicas: 0
  backend:
    listenerSpec:
      replicas: 0
    workerSpec:
      replicas: 0
    cronSpec:
      replicas: 0
  zync:
    appSpec:
      replicas: 0
    queSpec:
      replicas: 0
  system:
    appSpec:
      replicas: 0
    sidekiqSpec:
      replicas: 0
```

2.5.9.4. 手动故障转移共享数据库

流程

1. 在活跃的集群中，将 后端、system、zync 和 APIcast pod 的副本缩减为 0。
    a. 这会变为新的被动集群，因此您可以确保新的被动集群不会消耗来自活跃数据库的作业。停机时间从这里开始。
2. 在被动集群中，编辑 APIManager 以扩展 后端、system、zync 和 APIcast pod 的副本，使其变为活跃集群。
3. 在新的活动集群中，重新创建 zync 创建的 OpenShift 路由。
    a. 从 **system-app** pod 的 **system-master** 容器运行 **zync:resync:domains** 命令：

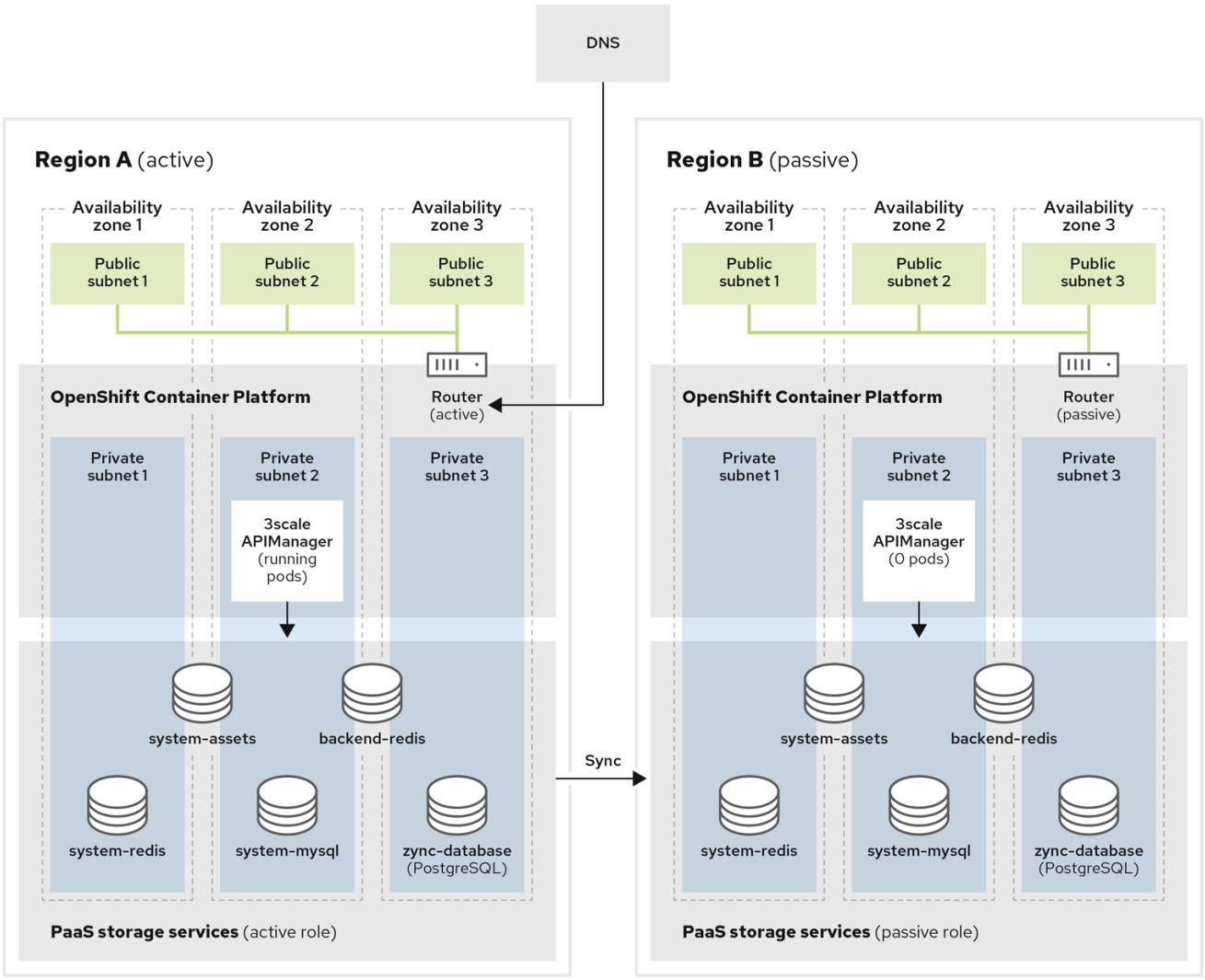
    ```
    bundle exec rake zync:resync:domains
    ```

4. 将 AWS Route53 中创建的自定义域指向新活跃集群的 OpenShift 路由器。
    a. 旧的被动集群将开始接收流量，并成为新的活跃集群。

2.5.9.5. 带有同步数据库的不同区域中的主动 - 被动集群

此设置包括在不同区域中有两个或更多集群，并以主动 - 被动模式部署 3scale。一个集群处于活动状态，接收流量，其他集群处于待机模式，而不接收流量，因此准备假定活跃角色在活跃集群中存在故障。

为确保良好的数据库访问延迟，每个集群都有自己的数据库实例。主动 3scale 安装中的数据库复制到 3scale 被动安装的 read-replica 数据库，以便数据在所有区域中可用，以实现可能的故障转移。



2.5.9.6. 配置并安装同步数据库

流程

1. *在不同的区域中创建两个或多个 OpenShift 集群，使用不同的可用区。建议至少有三个区域。*
2. *在每个区域上创建启用了 Amazon RDS Multi-AZ 的所有所需的 AWS ElastiCache 实例：*
   a. *两个用于后端 Redis 数据库的 AWS EC：每个区域一个。*
   b. *两个用于系统 Redis 数据库的 AWS EC：每个区域一个。*
   c. *使用启用 Global Datastore 功能的跨区域复制功能，因此来自被动区域的数据库会在活动区域从主数据库读取副本。*
3. *在每个区域都启用了 Amazon RDS Multi-AZ 创建所有必需的 AWS RDS 实例：*
   a. *系统数据库有两个 AWS RDS。*
   b. *两个用于 Zync 数据库的 AWS RDS。*
   c. *使用跨区域复制，因此来自被动区域的数据库是活动区域的主数据库读取副本。*

4. 使用跨区域复制，为每个区域上的系统资产配置 AWS S3 存储桶。

5. 在 AWS Route53 或 DNS 供应商中创建自定义域，并将其指向活跃集群的 OpenShift 路由器。这必须与 APIManager CR 中的 wildcardDomain 属性冲突。

6. 在被动集群中安装 3scale。APIManager CR 应该与上一步中使用的 CR 相同。当所有 pod 都运行时，更改 APIManager 以为所有 **后端**、**system**、**zync** 和 **APIcast** pod 部署 0 个副本。

    a. 将副本数设置为 0，以避免消耗活跃的数据库的作业。如果每个副本首次设置为 0，则部署会因为 pod 依赖项而失败。例如，检查其他人是否正在运行的 pod。首先以正常方式部署，然后将 replicas 设置为 0。

2.5.9.7. 手动故障转移同步数据库

流程

1. 执行手动故障转移共享数据库步骤 1、2 和 3。

    a. 每个集群都有自己的独立数据库：活动区域中的 master 的 read-replicas。

    b. 您必须在每个数据库上手动执行故障转移，以选择被动区域的新 master，然后成为活动区域。

2. 手动故障转移数据库要执行的是：

    a. AWS RDS: System 和 Zync。

    b. AWS ElastiCaches: Backend 和 System。

3. 执行手动故障转移共享数据库的第 4 步。

2.5.10. Amazon Simple Storage Service 3scale API Management fileStorage 安装

在创建 APIManager 自定义资源(CR)以部署 3scale 之前，请使用 OpenShift secret 为 Amazon Simple Storage Service (Amazon S3)服务提供连接设置。

重要

- 如果要使用本地文件系统存储部署 3scale，请跳过此部分。
- 您为 secret 选择的名称可以是任何名称，只要它不是现有 secret 名称，它将在 APIManager CR 中引用。
- 如果没有为 S3 兼容存储提供 **AWS_REGION**，请使用 **default**，否则部署将失败。
- 免责声明：包括在此处的外部网络链接仅为方便用户而提供。红帽没有审阅链接的内容，并不对其内容负责。包含任何指向外部网站的链接并不表示红帽认可该网站或其实体、产品或服务。您同意红帽对因您使用（或依赖）外部网站或内容而导致的任何损失或费用不承担任何责任。

2.5.10.1. Amazon S3 存储桶创建

先决条件

- 您必须有一个 Amazon Web Services (AWS)帐户。

流程

1. 创建用于存储系统资产的存储桶。
2. 使用开发人员门户的 logo 功能时，禁用 S3 的公共访问块程序。
3. 使用以下默认权限创建 Identity and Access Management(IAM)策略：

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "s3:ListAllMyBuckets",
         "Resource": "arn:aws:s3:::*"
      },
      {
         "Effect": "Allow",
         "Action": "s3:*",
         "Resource": [
            "arn:aws:s3:::<target_bucket_name>", (1)
            "arn:aws:s3:::<target_bucket_name>/*" (2)
         ]
      }
   ]
}
```

(1) 将 **<target_bucket_name>** 替换为您自己的值。

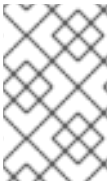
(2) 将 **<target_bucket_name>** 替换为您自己的值。

4. 使用以下规则创建 CORS 配置 ：

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
   <AllowedOrigin>https://*</AllowedOrigin>
   <AllowedMethod>GET</AllowedMethod>
</CORSRule>
</CORSConfiguration>
```

2.5.10.2. 创建 OpenShift secret

以下示例演示了使用 Amazon S3 而不是持久性卷声明(PVC)的 3scale 文件存储。



注意

AN AWS S3 兼容提供程序可以在带有 **AWS_HOSTNAME**、**AWS_PATH_STYLE** 和 **AWS_PROTOCOL** 可选密钥的 S3 机密中配置。如需了解更多详细信息，请参阅 fileStorage S3 credentials secret fields 表。

在以下示例中，Secret 名称可以是任意的，因为它在 APIManager CR 中被引用。

```
apiVersion: v1
kind: Secret
metadata:
  creationTimestamp: null
  name: aws-auth
stringData:
  AWS_ACCESS_KEY_ID: <ID_123456>
  AWS_SECRET_ACCESS_KEY: <ID_98765544>
  AWS_BUCKET: <mybucket.example.com>
  AWS_REGION: eu-west-1
type: Opaque
```

最后，创建 APIManager CR 以部署 3scale。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: <example_apimanager>
  namespace: <3scale_test>
spec:
  wildcardDomain: lvh.me
  system:
    fileStorage:
      simpleStorageService:
        configurationSecretRef:
          name: aws-auth
```

检查 APIManager SystemS3Spec。

下表显示了 Identity and Access Management (IAM)和安全令牌服务(STS)设置的 ***fileStorage*** *Amazon S3 凭证 secret 字段要求：*

- *使用安全令牌服务 (STS) 的S3 验证方法适用于短期、有有限权限的安全凭证。*
- *S3 Identity and Access Management (IAM) 用于长期权限安全凭证。*

表 2.1. Filestorage S3 credentials secret 字段

字段	描述	IAM 需要	STS 需要
AWS_ACCESS_KEY_ID	用于系统的 **fileStorage**的 S3 存储中使用的 AWS 访问密钥 ID	是	否
AWS_SECRET_ACCESS_KEY	用于系统的 **fileStorage**的 S3 存储中使用的 AWS 访问密钥 Secret	是	否

字段	描述	IAM 需要	STS 需要
AWS_BUCKET	用作资产的文件系统 **fileStorage** 的 S3 存储桶	是	是
AWS_REGION	用作资产的系统 **fileStorage** 的 S3 存储桶区域	是	是
AWS_HOSTNAME	默认 ：Amazon 端点 - AWS S3 兼容供应商端点主机名	否	否
AWS_PROTOCOL	默认：**HTTPS** AWS S3 兼容供应商端点协议	否	否
AWS_PATH_STYLE	默认：**false** - 当设置为 **true** 时，存储桶名称始终保留在请求 URI 中，并且永远不会作为子域移到主机。	否	否
AWS_ROLE_ARN	ARN 角色，它有一个附加的策略用于使用 AWS STS 进行身份验证	否	是
AWS_WEB_IDENTITY_TOKEN_FILE	挂载令牌文件位置的路径。例如：**/var/run/secrets/openshift/serviceaccount/token**	否	是

2.5.10.3. 使用 STS 的手动模式

*STS 验证模式必须从 APIManager CR 启用。您可以定义您的受众，但默认值为 **openshift**。*

先决条件

- *配置 OpenShift，以将临时凭据用于 AWS 安全令牌服务 (STS) 的不同组件。详情请查看在 Amazon Web Services Secure Token Service 中使用手动模式 。*

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: <apimanager_sample>
  namespace: <3scale_test>
spec:
  wildcardDomain: lvh.me
  system:
    fileStorage:
```

```
    simpleStorageService:
      configurationSecretRef:
        name: s3-credentials
      sts:
        enabled: true
        audience: openshift
```

*云凭证工具生成的secret 与 IAM secret 的不同。**AWS_ROLE_ARN** 和 **AWS_WEB_IDENTITY_TOKEN_FILE** 有两个新字段，替代了 **AWS_ACCESS_KEY_ID** 和 **AWS_SECRET_ACCESS_KEY**。*

STS secret 示例

```
kind: Secret
apiVersion: v1
metadata:
  name: s3-credentials
  namespace: 3scale
data:
  AWS_ROLE_ARN: arn:aws:iam::ID:role/ROLE
  AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/openshift/serviceaccount/token
  AWS_BUCKET: <mybucket.example.com>
  AWS_REGION: eu-west-1
type: Opaque
```

使用 STS 时，3scale 操作器会添加投射卷来请求令牌。以下 pod 有一个投射卷：

- ***system-app***
- ***system-app hook pre***
- ***system-sidekiq***

STS 的 Pod 示例

```
apiVersion: v1
kind: Pod
metadata:
  name: system-sidekiq-1-zncrz
  namespace: 3scale-test
spec:
  containers:
  ....
    volumeMounts:
    - mountPath: /var/run/secrets/openshift/serviceaccount
      name: s3-credentials
      readOnly: true
  ....
  volumes:
  - name: s3-credentials
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
```

```
    audience: openshift
    expirationSeconds: 3600
    path: token
```

其他资源

- APIManager SystemS3Spec
- S3 secret 参考
- 在 Amazon Web Services Secure Token Service 中使用手动模式
- 使用 AWS 安全令牌服务的简短凭证

2.5.11. PostgreSQL 安装

MySQL 内部关系数据库是默认的部署。此部署配置可以被覆盖来改用 PostgreSQL。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: lvh.me
  system:
    database:
      postgresql: {}
```

其他资源

- APIManager DatabaseSpec

2.5.12. 配置 SMTP 变量（可选）

3scale 使用电子邮件发送通知并邀请新用户。如果要使用这些功能，则必须提供自己的 SMTP 服务器并在 **system-smtp** 机密中配置 SMTP 变量。

执行以下步骤在 system-smtp secret 中配置 SMTP 变量：

流程

1. 如果您还没有登录，请登录到 OpenShift：

```
oc login
```

2. 使用 **oc patch** 命令，指定 **system-smtp** 是 secret 名称的 secret 类型，后跟 **-p** 选项，并在 JSON 中为以下变量写入新值：

表 2.2. system-smtp

字段	描述	默认值

字段	描述	默认值
address	这是要使用的远程邮件服务器的地址（主机名或 IP）。如果此值设为一个不是 **""** 的值，系统将使用邮件服务器来发送与 API 管理解决方案中发生的事件相关的邮件。	""
port	这是要使用的远程邮件服务器的端口。	""
domain	如果邮件服务器需要 HELO 域，使用**域**。	""
身份验证	如果邮件服务器需要身份验证时使用。设置身份验证类型：**plain** 以明文发送，**login** 中 Base64 编码的形式发送密码，或 **cram_md5** 根据 HMAC-MD5 算法组合一个质询/响应机制。	""
username	如果邮件服务器需要身份验证且身份验证类型需要，则使用 **username**。	""
password	如果邮件服务器需要身份验证且身份验证类型需要它，则使用 **password**。	""
openssl.verify.mode	使用 TLS 时，您可以设置 OpenSSL 如何检查证书。如果您需要验证自签名和/或通配符证书，这非常有用。您可以使用 OpenSSL 验证常量的名称：**none** 或 **peer**。	""
from_address	no-reply 邮件的 **from** 地址值。	""

例子

```
$ oc patch secret system-smtp -p '{"stringData":{"address":"<your_address>"}}'
$ oc patch secret system-smtp -p '{"stringData":{"username":"<your_username>"}}'
$ oc patch secret system-smtp -p '{"stringData":{"password":"<your_password>"}}'
```

3. *设置 secret 变量后，重新部署 **system-app** 和 **system-sidekiq** pod:*

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

4. 检查推出部署的状态，以确保它已完成：

```
$ oc rollout status dc/system-app
$ oc rollout status dc/system-sidekiq
```

2.5.13. 在组件级别自定义计算资源要求

通过 APIManager 自定义资源(CR)属性自定义 3scale 解决方案中的 Kubernetes 计算资源要求。https://kubernetes.io/docs/concepts/configuration/manage-resources-containers这样做可自定义分配给特定 APIManager 组件的计算资源要求，即 CPU 和内存。

以下示例概述了如何自定义 system-master 的 **system-provider** 容器、 **backend-listener** 和 **zync-database** 的计算资源要求：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      resources:
        requests:
          memory: "150Mi"
          cpu: "300m"
        limits:
          memory: "500Mi"
          cpu: "1000m"
  system:
    appSpec:
      developerContainerResources:
        limits:
          cpu: 1500m
          memory: 1400Mi
        requests:
          cpu: 150m
          memory: 600Mi
      masterContainerResources:
        limits:
          cpu: 1500m
          memory: 1400Mi
        requests:
          cpu: 150m
          memory: 600Mi
      providerContainerResources:
        limits:
          cpu: 1500m
          memory: 1400Mi
        requests:
          cpu: 150m
          memory: 600Mi
  zync:
    databaseResources:
      requests:
        memory: "111Mi"
```

```
    cpu: "222m"
  limits:
    memory: "333Mi"
    cpu: "444m"
```

其他资源

- APIManager CRD 参考

2.5.13.1. 默认 APIManager 组件计算资源

当您将 APIManager **spec.resourceRequirementsEnabled** 属性配置为 **true** 时，会为 APIManager 组件设置默认计算资源。

下表中显示了为 APIManager 组件设置的特定计算资源默认值。

2.5.13.1.1. CPU 和内存单元

下表说明了您将在计算资源默认值表中找到的单元。如需有关 CPU 和内存单元的更多信息，请参阅管理容器的资源。

资源单元解释

- m - milliCPU 或 millicore
- Mi - mebibytes
- Gi - gibibyte
- G - gigabyte

表 2.3. 计算资源默认值

组件	CPU 请求	CPU 限值	内存请求	内存限值
system-app 的 system-master	50m	1000m	600Mi	800Mi
system-app 的 system-provider	50m	1000m	600Mi	800Mi
system-app 的 system-developer	50m	1000m	600Mi	800Mi
system-sidekiq	100m	1000m	500Mi	2Gi
system-searchd	80m	1000m	250Mi	512Mi
system-redis	150m	500m	256Mi	32Gi
system-mysql	250m	无限制	512Mi	2Gi

组件	CPU 请求	CPU 限值	内存请求	内存限值
system-postgresql	250m	无限制	512Mi	2Gi
backend-listener	500m	1000m	550Mi	700Mi
backend-worker	150m	1000m	50Mi	300Mi
backend-cron	50m	150m	40Mi	80Mi
backend-redis	1000m	2000m	1024Mi	32Gi
apicast-production	500m	1000m	64Mi	128Mi
apicast-staging	50m	100m	64Mi	128Mi
zync	150m	1	250M	512Mi
zync-que	250m	1	250M	512Mi
zync-database	50m	250m	250M	2G

2.5.14. 组件级别的自定义节点关联性和容限

通过 APIManager CR 属性自定义 Red Hat 3scale API 管理解决方案中的 Kubernetes 关联性和 容限，以自定义安装的不同 3scale 组件如何调度到 Kubernetes 节点上。

*以下示例为后端设置自定义节点关联性。它还为 **system-memcached** 设置监听程序和自定义容限：*

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  backend:
    listenerSpec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: "kubernetes.io/hostname"
                operator: In
                values:
                - ip-10-96-1-105
              - key: "beta.kubernetes.io/arch"
                operator: In
                values:
                - amd64
  system:
```

```
memcachedTolerations:
- key: key1
  value: value1
  operator: Equal
  effect: NoSchedule
- key: key2
  value: value2
  operator: Equal
  effect: NoSchedule
```

其他资源

- APIManager CRD 参考

2.5.15. 3scale API 管理组件的 Pod 优先级

作为 3scale 管理员，您可以通过修改 APIManager 自定义资源(CR)来为各种 3scale 安装组件设置 pod 优先级。使用以下组件中提供的可选 **priorityClassName** ：

- **apicast-production**
- **apicast-staging**
- **backend-cron**
- **backend-listener**
- **backend-worker**
- **backend-redis**
- **system-app**
- **system-sidekiq**
- **system-searchd**
- **system-memcache**
- **system-mysql**
- **system-postgresql**
- **system-redis**
- **zync**
- **zync-database**
- **zync-que**

例如：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
```

```
    name: example-apimanager
spec:
    wildcardDomain: api.vmogilev01.0nno.s1.devshift.org
    resourceRequirementsEnabled: false
    apicast:
        stagingSpec:
            priorityClassName: openshift-user-critical
        productionSpec:
            priorityClassName: openshift-user-critical
    backend:
        listenerSpec:
            priorityClassName: openshift-user-critical
        cronSpec:
            priorityClassName: openshift-user-critical
        workerSpec:
            priorityClassName: openshift-user-critical
        redisPriorityClassName: openshift-user-critical
    system:
        appSpec:
            priorityClassName: openshift-user-critical
        sidekiqSpec:
            priorityClassName: openshift-user-critical
        searchdSpec:
            priorityClassName: openshift-user-critical
        searchdSpec:
            priorityClassName: openshift-user-critical
        memcachedPriorityClassName: openshift-user-critical
        redisPriorityClassName: openshift-user-critical
        database:
            postgresql:
                priorityClassName: openshift-user-critical
    zync:
        appSpec:
            priorityClassName: openshift-user-critical
        queSpec:
            priorityClassName: openshift-user-critical
        databasePriorityClassName: openshift-user-critical
```

2.5.16. 设置自定义标签

您可以通过应用于对应 pod 的每个 DeploymentConfig (DC)的 APIManager CR 标签属性自定义标签。

注意

如果您删除了自定义资源(CR)中定义的标签，它不会自动从关联的 DeploymentConfig (DC)中删除。您必须手动从 DC 中删除该标签。

apicast-staging 和 **backend-listener** 的示例：

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

```
  wildcardDomain: example.com
  resourceRequirementsEnabled: false
  backend:
   listenerSpec:
     labels:
       backendLabel1: sample-label1
       backendLabel2: sample-label2
  apicast:
    stagingSpec:
     labels:
       apicastStagingLabel1: sample-label1
       apicastStagingLabel2: sample-label2
```

其他资源

APIManager CRD 参考

2.5.17. 设置后端客户端以跳过证书验证

当控制器处理对象时，它会生成新的后端客户端来发出 API 调用。默认情况下，设置此客户端以确认服务器的证书链。但是，在开发和测试过程中，您可能需要客户端在处理对象时跳过证书验证。要做到这一点，将注解 ***"insecure_skip_verify": "true"*** *添加到以下对象：*

- *ActiveDoc*
- *Application*
- *后端*
- *CustomPolicyDefinition*
- *DeveloperAccount*
- *DeveloperUser*
- *OpenAPI - 后端和产品*
- *产品*
- *ProxyConfigPromote*
- *租户*

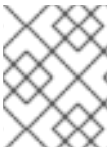
OpenAPI CR 示例：

```
apiVersion: capabilities.3scale.net/v1beta1
kind: OpenAPI
metadata:
  name: ownertest
  namespace: threescale
  annotations:
    "insecure_skip_verify": "true"
spec:
  openapiRef:
    secretRef:
```

```
    name: myopenapi
    namespace: threescale
productSystemName: testProduct
```

2.5.18. 设置自定义注解

在 3scale 中，组件的 pod 具有注解。这些是用于配置的键/值对。您可以使用 APIManager CR 为任何 3scale 组件更改这些注解。



注意

如果您删除了自定义资源(CR)中定义的注解，它不会自动从关联的 DeploymentConfig (DC)中删除。您必须手动从 DC 中删除注解。

apicast-staging 和 backend-listener 的 APIManager 注解

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  wildcardDomain: example.com
  apicast:
    stagingSpec:
      annotations:
        anno-sample1: anno1
  backend:
    listenerSpec:
      annotations:
        anno-sample2: anno2
```

其他资源

- APIManager CRD 参考

2.5.19. 协调

安装 3scale 后，3scale 操作器将启用更新自定义资源（CR）中的一组给定参数，以修改系统配置选项。可以通过 热交换 （即，不停止或关闭系统）进行修改。

当 3scale 操作器和 APIcast operator 中发生协调事件时，有两种可能的情况：

- 如果没有 **deploymentconfig**，并且 CR 有副本，则 **deploymentconfig** 值将与这些副本匹配。如果 CR 不包含副本，则 **deploymentconfig** 副本值将设置为 **1**。
- 当有 **deploymentconfig** 且 CR 有副本时，deploymentconfig 值会匹配这些副本，即使它是 **0**。如果 CR 不包含副本，则 **deploymentconfig** 值保持不变。

不是 APIManager CR 定义(CRD)的所有参数都是可协调的。

以下是可协调参数列表：

- Resources

- 后端副本
- APIcast 副本
- 系统副本
- Zync 副本

2.5.19.1. Resources

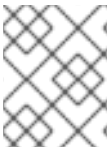
所有 3scale 组件的资源限制和请求.

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  resourceRequirementsEnabled: true/false
```

2.5.19.2. 后端副本

后端 组件 pod 数量。

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



注意

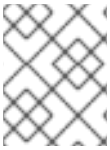
如果没有设置 replica 字段，Operator 不会协调副本。这允许第三方控制器管理副本，如 HorizontalPodAutoscaler 控制器。它还允许在部署对象上手动更新它们。

2.5.19.3. APIcast 副本

APIcast staging 和生产组件 pod 数量。

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

```
    replicas: X
  stagingSpec:
    replicas: Z
```



注意

如果没有设置 replica 字段，Operator 不会协调副本。这允许第三方控制器管理副本，如 HorizontalPodAutoscaler 控制器。它还允许在部署对象上手动更新它们。

2.5.19.4. 系统副本

系统 应用程序和系统 sidekiq 组件 pod 数量。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  system:
    appSpec:
      replicas: X
    sidekiqSpec:
      replicas: Z
```

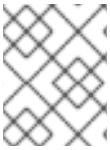
注意

如果没有设置 replica 字段，Operator 不会协调副本。这允许第三方控制器管理副本，如 HorizontalPodAutoscaler 控制器。它还允许在部署对象上手动更新它们。

2.5.19.5. Zync 副本

Zync app 和 que 组件 pod 数量。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  zync:
    appSpec:
      replicas: X
    queSpec:
      replicas: Z
```

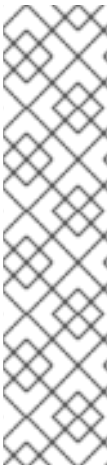


注意

如果没有设置 replica 字段，Operator 不会协调副本。这允许第三方控制器管理副本，如 HorizontalPodAutoscaler 控制器。它还允许在部署对象上手动更新它们。

2.6. 使用 ORACLE 作为系统数据库的操作器安装 3SCALE API 管理

作为红帽 3scale API 管理管理员，您可以使用 Oracle 数据库使用操作器安装 3scale。默认情况下，3scale 2.14 有一个名为 **system** 的组件，该组件将配置数据存储在 MySQL 数据库中。您可以覆盖默认数据库，并将信息存储在外部 Oracle 数据库中。



注意

- 当您只执行 3scale 的 Operator 安装时，OpenShift Container Platform(OCP)版本 4.2 和 4.3 不支持 Oracle 数据库。如需更多信息，请参阅 Red Hat 3scale API 管理支持的配置 页面。
- 在本文档中，**myregistry.example.com** 用作 registry URL 的示例。将它替换为您的 registry URL。
- 免责声明：包括在此处的外部网络链接仅为方便用户而提供。红帽没有审阅链接的内容，并不对其内容负责。包含任何指向外部网站的链接并不表示红帽认可该网站或其实体、产品或服务。您同意红帽对因您使用（或依赖）外部网站或内容而导致的任何损失或费用不承担任何责任。

先决条件

- 一个容器 registry，用于推送容器镜像，该镜像可由安装 3scale 的 OCP 集群访问。
- 3scale operator 的安装。
  - 不要安装 APIManager CR，因为它将在以下步骤中创建。
- 3scale 的 Registry 服务帐户。
- 可以从 OpenShift cluster 访问的一个 Oracle Database 的支持版本。
- 访问 Oracle Database **SYSTEM** 用户以获取安装过程。

要使用 Oracle 作为系统数据库通过操作器安装 3scale，请执行以下步骤：

- 准备 Oracle 数据库
- 构建自定义系统容器镜像
- 使用 Operator 安装 3scale API 管理

2.6.1. 准备 Oracle 数据库

作为 3scale 管理员，当您决定将其用于系统组件时，您必须为 3scale 安装准备 Oracle 数据库。

流程

1. 创建新数据库。
2. 应用以下设置：

```
ALTER SYSTEM SET max_string_size=extended SCOPE=SPFILE;
```

3. 配置数据库用户
   在 3scale 中设置 Oracle 数据库集成有两个选项：带有或不提供 Oracle **SYSTEM** 用户密码。

3scale 仅将 **SYSTEM** 用户用于初始设置，它包含创建常规用户并授予其所需的权限。以下 SQL 命令将设置一个具有适当权限的普通用户。(**{DB_USER}** 和 **{DB_PASSWORD}** 是需要替换为实际值的占位符)：

```
CREATE USER {DB_USER} IDENTIFIED BY {DB_PASSWORD};
GRANT unlimited tablespace TO {DB_USER};
GRANT create session TO {DB_USER};
GRANT create table TO {DB_USER};
GRANT create view TO {DB_USER};
GRANT create sequence TO {DB_USER};
GRANT create trigger TO {DB_USER};
GRANT create procedure TO {DB_USER};
```

a. 使用 **SYSTEM** 用户：

- 在 **system-database** secret 的 **ORACLE_SYSTEM_PASSWORD** 字段中提供 **SYSTEM** 用户密码。
- 安装前，普通用户不需要存在。它将由 3scale 初始化脚本创建。
- 在连接字符串中为常规用户提供所需的用户名和密码，例如：**oracle-enhanced://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}** in the **system-database** secret 的 **URL** 字段中。
- 常规 Oracle 数据库的非系统用户的密码必须是唯一的，且与 **SYSTEM** 用户密码不匹配。
- 如果具有指定用户名的用户已存在，3scale 初始化脚本将使用以下命令更新密码：

```
ALTER USER {DB_USER} IDENTIFIED BY {DB_PASSWORD}
```

如果参数 **PASSWORD_REUSE_TIME** and **PASSWORD_REUSE_MAX** 被设置为限制重新使用相同的密码，则数据库配置可能会阻止这个命令成功完成。

b. 手动设置常规数据库用户：

- 您不需要在 **system-database** secret 中提供 **ORACLE_SYSTEM_PASSWORD**。
- 在 3scale 安装前，在 **system-database** secret 的 **URL** 字段中指定的连接字符串中指定的常规数据库用户（而不是 **SYSTEM**）。
- 用于安装的普通用户必须具有上面列出的所有权限。

其他资源

- 有关创建新数据库的详情，请查看 Oracle 数据库 19c 文档。

2.6.2. 构建自定义系统容器镜像

流程

1. 从 GitHub 存储库下载 3scale OpenShift 模板并提取存档：

```
tar -xzf 3scale-2.14.0-GA.tar.gz
```

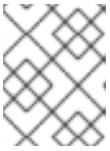
2. *在 Instant Client Downloads 页面中下载：*
   - *客户端：可以是 basic-lite 或 basic。*
   - *ODBC driver。*
   - *Oracle 数据库 19c 的 SDK。*
     - *对于 3scale，使用 Instant Client Downloads for Linux x86-64 (64-bit)*
     - *对于 ppc64le 和 3scale，使用 Oracle Instant Client Downloads for Linux on Power Little Endian (64-bit)*
3. *检查以下 Oracle 软件组件版本的表：*
   - *Oracle Instant Client Package: Basic 或 Basic Light*
   - *Oracle Instant Client Package: SDK*
   - *Oracle Instant Client Package: ODBC*

表 2.4. 3scale 的 Oracle 19c 示例软件包

Oracle 19c 软件包名称	压缩的文件名
基本的	instantclient-basic-linux.x64-19.8.0.0.0dbru.zip
Basic Light	instantclient-basiclite-linux.x64-19.8.0.0.0dbru.zip
SDK	instantclient-sdk-linux.x64-19.8.0.0.0dbru.zip
ODBC	instantclient-odbc-linux.x64-19.8.0.0.0dbru.zip

表 2.5. Oracle 19c 示例软件包用于 ppc64le 和 3scale

Oracle 19c 软件包名称	压缩的文件名
基本的	instantclient-basic-linux.leppc64.c64-19.3.0.0.0dbru.zip
Basic Light	instantclient-basiclite-linux.leppc64.c64-19.3.0.0.0dbru.zip
SDK	instantclient-sdk-linux.leppc64.c64-19.3.0.0.0dbru.zip
ODBC	instantclient-odbc-linux.leppc64.c64-19.3.0.0.0dbru.zip



注意

如果本地下载并存储的客户端软件包版本与 3scale 期望不匹配，3scale 将自动下载并使用以下步骤中的相应版本。

4. *将 Oracle Database Instant Client Package 文件放在 **system-oracle-3scale-2.13.0-GA/oracle-client-files** 目录中。*

5. *使用您在创建 registry 服务帐户中创建的凭证登录到 **registry.redhat.io** 帐户。*

```
$ docker login registry.redhat.io
```

6. *构建基于 Oracle 的自定义系统镜像.镜像标签必须是固定镜像标签，如下例所示：*

```
$ docker build . --tag myregistry.example.com/system-oracle:2.14.0-1
```

7. *将基于 Oracle 的系统镜像推送到 OCP 集群可访问的容器注册表。此容器 registry 是安装 3scale 解决方案的位置：*

```
$ docker push myregistry.example.com/system-oracle:2.14.0-1
```

2.6.3. 使用 Operator 安装 3scale API 管理

流程

1. *通过使用对应字段创建 **system-database** secret，设置 Oracle Database URL 连接字符串和 Oracle Database 系统密码。请参阅为 Oracle 数据库安装外部数据库。*

2. *通过创建 APIManager CR 来安装 3scale 解决方案。按照 使用 operator 部署 3scale API 管理 中的说明进行操作。*

   - *APIManager CR 必须指定设置为您之前构建的基于 Oracle 的系统的 **.spec.system.image** 字段：*

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIManager
metadata:
  name: example-apimanager
spec:
  imagePullSecrets:
  - name: threescale-registry-auth
  - name: custom-registry-auth
  system:
    image: "myregistry.example.com/system-oracle:2.14.0-1"
  externalComponents:
    system:
      database: true
```

2.7. 常见 3SCALE API 管理安装问题的故障排除

这部分包含常见安装问题列表，并为它们解决提供指导。

- *以前的部署导致存在脏的持久性卷声明*

- 经过身份验证的用户 registry 的错误或缺少凭证
- 从 Docker registry 中拉取错误
- 在本地挂载持久性卷时 MySQL 的权限问题
- 无法上传徽标或图像
- 测试在 OpenShift 中无法正常工作的调用
- 来自 3scale API 管理的不同项目上部署 APIcast 失败

2.7.1. 以前的部署导致存在脏的持久性卷声明

问题

之前的部署尝试导致脏的持久性卷声明(PVC)，从而导致 MySQL 容器无法启动。

原因

在 OpenShift 中删除项目不会清理与其关联的 PVC。

解决方案

流程

1. 使用 **oc get pvc** 命令查找包含错误 MySQL 数据的 PVC ：

```
# oc get pvc
NAME                 STATUS    VOLUME    CAPACITY   ACCESSMODES   AGE
backend-redis-storage   Bound     vol003    100Gi      RWO,RWX       4d
mysql-storage           Bound     vol006    100Gi      RWO,RWX       4d
system-redis-storage    Bound     vol008    100Gi      RWO,RWX       4d
system-storage          Bound     vol004    100Gi      RWO,RWX       4d
```

2. 单击 OpenShift Container Platform (OCP)控制台中的**cancel 部署**，以停止 **system-mysql** pod 的部署。
3. 删除 MySQL 路径下的所有内容，以清理卷。
4. 启动新的 **system-mysql** 部署。

2.7.2. 经过身份验证的用户 registry 的错误或缺少凭证

问题

Pod 没有启动。镜像流显示以下错误：

```
! error: Import failed (InternalError): ...unauthorized: Please login to the Red Hat Registry
```

原因

在 OpenShift 4.x 上安装 3scale 时，OpenShift 无法启动 pod，因为 ImageStreams 无法拉取它们引用的镜像。这是因为 pod 无法针对它们所指向的 registry 进行身份验证。

解决方案

流程

1. *键入以下命令以验证容器 registry 身份验证的配置：*

```
$ oc get secret
```

   - *如果 secret 存在，您会在终端中看到以下输出：*

```
threescale-registry-auth        kubernetes.io/dockerconfigjson        1        4m9s
```

   - *但是，如果没有看到输出结果，您必须执行以下操作：*

2. *在创建 registry 服务帐户时，使用之前设置的凭证来创建您的 secret。*

3. *使用在 OpenShift 中配置 registry 身份验证的步骤，替换* ***oc create secret*** *命令中的* ***<your-registry-service-account-username>*** *and* ***<your-registry-service-account-password>***。

4. *在与 APIManager 资源相同的命名空间中生成* ***threescale-registry-auth*** *secret。您必须在* ***<project-name>*** *中运行以下命令：*

```
$ oc project <project-name>
$ oc create secret docker-registry threescale-registry-auth \
  --docker-server=registry.redhat.io \
  --docker-username="<your-registry-service-account-username>" \
  --docker-password="<your-registry-service-account-password>"
  --docker-email="<email-address>"
```

5. *删除并重新创建 APIManager 资源：*

```
$ oc delete -f apimanager.yaml
apimanager.apps.3scale.net "example-apimanager" deleted

$ oc create -f apimanager.yaml
apimanager.apps.3scale.net/example-apimanager created
```

验证

1. *键入以下命令，以确认部署的状态为* ***Starting*** *或* ***Ready*** *：pod 会开始生成：*

```
$ oc describe apimanager
(...)
Status:
  Deployments:
    Ready:
      apicast-staging
      system-memcache
      system-mysql
      system-redis
      zync
      zync-database
      zync-que
    Starting:
```

```
    apicast-production
    backend-cron
    backend-worker
    system-sidekiq
    system-searchd
  Stopped:
    backend-listener
    backend-redis
    system-app
```

2. 键入以下命令以查看每个 pod 的状态：

```
$ oc get pods
NAME                              READY   STATUS      RESTARTS   AGE
3scale-operator-66cc6d857b-sxhgm  1/1     Running     0          17h
apicast-production-1-deploy       1/1     Running     0          17m
apicast-production-1-pxkqm        0/1     Pending     0          17m
apicast-staging-1-dbwcw           1/1     Running     0          17m
apicast-staging-1-deploy          0/1     Completed   0          17m
backend-cron-1-deploy             1/1     Running     0          17m
```

2.7.3. 从 Docker registry 中拉取错误

问题

在安装过程中出现以下错误：

```
svc/system-redis - 1EX.AMP.LE.IP:6379
  dc/system-redis deploys docker.io/rhscl/redis-32-rhel7:3.2-5.3
    deployment #1 failed 13 minutes ago: config change
```

原因

OpenShift 通过发出 **docker** 命令来搜索和调取容器镜像。此命令引用 **docker.io** Docker registry，而不是 **registry.redhat.io** 红帽生态系统目录。

当系统包含 Docker 容器化环境的意外版本时，会出现这种情况。

解决方案

流程

使用适当的 Docker 容器化环境版本。

2.7.4. 在本地挂载持久性卷时 MySQL 的权限问题

问题

system-msql pod 崩溃且不部署，从而导致其他系统依赖它失败部署。pod 日志显示以下错误：

```
[ERROR] Cannot start server : on unix socket: Permission denied
[ERROR] Do you already have another mysqld server running on socket: /var/lib/mysql/mysql.sock ?
[ERROR] Aborting
```

原因

MySQL 进程启动时不正确的用户权限。

解决方案

流程

1. 用于持久性卷的目录 MUST 具有 root 组的写入权限。对 root 用户具有读写权限不够，因为 MySQL 服务以 root 组中的不同用户身份运行。以 root 用户身份执行以下命令：

```
$ chmod -R g+w /path/for/pvs
```

2. 执行以下命令以防止 SElinux 阻止访问：

```
$ chcon -Rt svirt_sandbox_file_t /path/for/pvs
```

2.7.5. 无法上传徽标或图像

问题

无法上传徽标 - **system-app** 日志显示以下错误：

```
Errno::EACCES (Permission denied @ dir_s_mkdir - /opt/system/public//system/provider-name/2
```

原因

OpenShift 无法写入持久卷。

解决方案

流程

确保您的持久卷可由 OpenShift 写入。它应归 root 组所有，并且可写入组。

2.7.6. 测试在 OpenShift 中无法正常工作的调用

问题

测试调用在创建新服务和 OpenShift 上的路由后无法正常工作。通过 curl 直接调用也失败，带有 **service not available** 信息。

原因

3scale 默认需要 HTTPS 路由，并且 OpenShift 路由不受保护。

解决方案

流程

确保 OpenShift 路由器设置中点击了 **安全路由** 复选框。

2.7.7. 来自 3scale API 管理的不同项目上部署 APIcast 失败

问题

APIcast 部署失败（pod 没有变为蓝色）。您在日志中看到以下错误：

> *update acceptor rejected apicast-3: pods for deployment "apicast-3" took longer than 600 seconds to become ready*

您在 pod 中看到以下错误：

> *Error synching pod, skipping: failed to "StartContainer" for "apicast" with RunContainerError: "GenerateRunContainerOptions: secrets \"apicast-configuration-url-secret\" not found"*

原因

该机密没有正确设置。

解决方案

流程

使用 APIcast v3 创建 secret 时，指定 ***apicast-configuration-url-secret*** *：*

> *$ oc create secret generic apicast-configuration-url-secret --from-literal=password=https://<ACCESS_TOKEN>@<TENANT_NAME>-admin.<WILDCARD_DOMAIN>*

2.8. 其他资源

- *外部组件规格*
- *系统数据库*

第 3 章 安装 APICAST

APIcast 是基于 NGINX 的 API 网关，用于将您的内部和外部 API 服务与红帽 3scale API 管理平台集成。APIcast 利用循环实现负载平衡。

在本指南中，您将了解部署选项、提供的环境以及如何入门。

先决条件

APIcast 不是独立的 API 网关。需要连接到 3scale API Manager。

- 一个有效的 3scale On-Premises 实例。

要安装 APIcast，请执行以下部分中所述的步骤：

- APIcast 部署选项
- APIcast 环境
- 配置集成设置
- 配置您的产品
- 使用操作器部署 APIcast 网关自助管理解决方案

3.1. APICAST 部署选项

您可以使用托管或自我管理的 APIcast。在这两种情况下，APIcast 必须连接到 3scale API 管理平台的其余部分：

- **嵌入式 APIcast**：3scale API 管理安装包含两个默认的 APIcast 网关，即 staging 和 production。这些网关已预先配置，并可供立即使用。
- **自我管理 APIcast**：您可以随时部署 APIcast。以下是部署 APIcast 的建议选项之一：
  - 在 Red Hat OpenShift 上运行 APIcast：在受支持的 OpenShift 版本上运行 APIcast。您可以将自我管理的 APIcasts 连接到 3scale 内部安装或 3scale 托管(SaaS)帐户。为此，请使用 operator 部署 APIcast 网关自助管理解决方案。

3.2. APICAST 环境

默认情况下，当您创建 3scale 帐户时，您将在两个不同的环境中获取嵌入式 APIcast：

- **Stage**：仅在配置和测试 API 集成时使用。当您确认您的设置按预期工作时，您可以选择将其部署到生产环境中。
- **Production**：此环境专用于生产环境。在 OpenShift 模板中为 Production APIcast 设置以下参数：**APICAST_CONFIGURATION_LOADER: boot,APICAST_CONFIGURATION_CACHE: 300**。这意味着，在 APIcast 启动时将完全加载配置，并将缓存 300 秒（5 分钟）。5 分钟后将重新加载配置。这意味着，当您将配置提升到生产环境时，可能需要 5 分钟才能应用，除非您触发新的 APIcast 部署。

3.3. 配置集成设置

作为 3scale 管理员，配置运行 3scale 的环境的集成设置。

先决条件

具有管理员特权的 3scale 帐户。

流程

1. *导航到 **[Your_product_name] > Integration > Settings**。*
2. *在 **Deployment** 中，默认选项如下：*
   - *部署选项：APIcast 3scale 管理*
   - *身份验证模式：API 密钥*
3. *更改到您首选的选项。*
4. *要保存您的更改，请点击 **Update Product**。*

3.4. 配置您的产品

您必须在 Private Base URL 字段中声明您的 API 后端，这是 API 后端的端点主机。在处理了所有身份验证、授权、速率限值和统计数据后，APIcast 将所有流量重定向到您的 API 后端。

本节将指导您配置产品：

- *声明 API 后端*
- *配置身份验证设置*
- *配置 API 测试调用*

3.4.1. 声明 API 后端

*通常，您的 API 的私有基本 URL 将会像 **https://api-backend.yourdomain.com:443** 一样在您管理的域中（**yourdomain.com**）。例如，如果您与 Twitter API 集成，则私有基本 URL 为 **https://api.twitter.com/**。*

*在本例中，您将使用 3scale 托管的 **Echo API**，它是一个简单的 API，接受任何路径并返回有关请求的信息（路径、请求参数、标头等）。其专用基础 URL 是 **https://echo-api.3scale.net:443**。*

流程

- *测试您的私有（非受管）API 是否正常工作。例如，对于 Echo API，您可以使用 **curl** 命令进行以下调用：*

```
$ curl "https://echo-api.3scale.net:443"
```

您将获得以下响应：

```
{
    "method": "GET",
    "path": "/",
    "args": "",
    "body": "",
    "headers": {
```

```
    "HTTP_VERSION": "HTTP/1.1",
    "HTTP_HOST": "echo-api.3scale.net",
    "HTTP_ACCEPT": "*/*",
    "HTTP_USER_AGENT": "curl/7.51.0",
    "HTTP_X_FORWARDED_FOR": "2.139.235.79, 10.0.103.58",
    "HTTP_X_FORWARDED_HOST": "echo-api.3scale.net",
    "HTTP_X_FORWARDED_PORT": "443",
    "HTTP_X_FORWARDED_PROTO": "https",
    "HTTP_FORWARDED": "for=10.0.103.58;host=echo-api.3scale.net;proto=https"
  },
  "uuid": "ee626b70-e928-4cb1-a1a4-348b8e361733"
}
```


3.4.2. 配置身份验证设置

您可以在 [Your_product_name] > Integration > Settings的 AUTHENTICATION 部分中为 API 配置身份验证设置。

表 3.1. 可选的身份验证字段

字段	描述
Auth user key	设置与凭据位置关联的用户密钥。
Credentials location	定义凭据是作为 HTTP 标头、查询参数还是 HTTP 基本身份验证传递。
Host Header	定义自定义主机请求标头。如果您的 API 后端仅接受来自特定主机的流量，则需要此设置。
Secret Token	用于阻止开发人员向 API 后端发出请求。设置此处标头的值，并确保您的后端只允许使用此机密标头调用。

另外，您可以在 [Your_product_name] > Integration > Settings 下配置 GATEWAY RESPONSE 错误代码。为错误定义 Response Code、Content-type 和 Response Body ：身份验证失败、身份验证缺失和不匹配。

表 3.2. 响应代码和默认响应正文

响应代码	响应正文
403	身份验证失败
403	缺少身份验证参数
404	没有匹配的映射规则
429	超过用量限制

3.4.3. 配置 API 测试调用

配置 API 涉及使用产品测试后端，并将 APIcast 配置提升到暂存和生产环境，以根据请求调用进行测试。

对于每个产品，请求会根据路径重定向到对应的后端。当您将后端添加到产品时会配置这个路径。例如，如果您在某个产品中添加了两个后端，每个后端都有自己的路径。

先决条件

- 添加到产品的一个或多个后端。
- 添加到产品的每个后端的映射规则。
- 应用计划定义访问策略。
- 订阅 应用程序 计划的应用程序。

流程

1. 进入到 **[Your_product_name] > Integration > Configuration**，将 APIcast 配置提升到 Staging。
2. 在 APIcast 配置 下，您将看到添加到产品的每个后端的映射规则。点 **Promote v.[n] to Staging APIcast**。
   - **v.[n]** 表示要提升的版本号。
3. 提升到暂存后，您可以将其提升为 Production。在 Staging APIcast 下，点 **Promote v.[n] to Production APIcast**。
   - **v.[n]** 表示要提升的版本号。
4. 要在命令行中测试对您的 API 的请求，请使用用于测试的curl 示例中提供的命令。
   - curl 命令示例基于产品中的第一个映射规则。

在测试对 API 的请求时，您可以通过添加方法和指标来修改映射规则。

每次修改配置时，并在调用 API 之前，请确保提升到 Staging 和 Production 环境。当要提升到 Staging 环境的待处理更改时，您会在管理门户中看到一个声明标记，它位于 **Integration** 菜单项的旁边。

3scale Hosted APIcast 网关进行凭证验证，并应用您为 API 应用计划定义的速率限制。如果您在没有凭证的调用或具有无效凭证的调用，您会看到出错信息，**Authentication failed**。

3.4.4. 在 Podman 上部署 APIcast

这是在 Pod Manager(Podman)容器环境中部署 APIcast 的逐步指南，用作红帽 3scale API 管理 API 网关。

注意

在 Podman 容器环境中部署 APIcast 时，受支持的 Red Hat Enterprise Linux(RHEL)和 Podman 版本如下：

- RHEL 8.x/9.x
- Podman 4.2.0/4.1.1

先决条件

- 您必须根据 第 3 章 安装 APIcast 在 3scale 管理门户中配置 APIcast。
- 访问红帽生态系统目录。
  - 要创建 registry 服务帐户，请参阅 创建和修改 registry 服务帐户。

要在 Podman 容器环境中部署 APIcast，请执行以下部分中所述的步骤：

- 第 3.4.4.1 节 “安装 Podman 容器环境”
- 第 3.4.4.2 节 “运行 Podman 环境”

3.4.4.1. 安装 Podman 容器环境

本指南涵盖了在 RHEL 8.x 上设置 Podman 容器环境的步骤。RHEL 8.x 中不包括 Docker，因此请使用 Podman 来运行容器。

有关 RHEL 8.x 的 Podman 的详情，请参阅容器命令行参考。

流程

- 安装 Podman 容器环境软件包：

```
$ sudo dnf install podman
```

其他资源

对于其他操作系统，请参阅以下 Podman 文档：

- Podman 安装说明

3.4.4.2. 运行 Podman 环境

要运行 Podman 容器环境，请按照以下步骤操作。

流程

1. 从 Red Hat registry 下载就绪的 Podman 容器镜像：

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

2. 在 Podman 中运行 APIcast：

```
$ podman run --name apicast --rm -p 8080:8080 -e
THREESCALE_PORTAL_ENDPOINT=https://<access_token>@<domain>-admin.3scale.net
registry.redhat.io/3scale-amp2/apicast-gateway-rhel8:3scale2.14
```

在这里，**<access_token>** 是 3scale 帐户管理 API 的访问令牌。您可以使用 Provider Key 而不是访问令牌。**<domain>-admin.3scale.net** 是 3scale 管理门户的 URL。

此命令在端口 **8080** 上运行名为 "apicast" 的 Podman 容器引擎，并从 3scale 管理门户获取 JSON 配置文件。有关其他配置选项，请参阅 安装 APIcast。

3.4.4.2.1. 使用 Podman 测试 APIcast

上述步骤确保 Podman 容器引擎使用您自己的配置文件和 3scale registry 中的 Podman 容器镜像来运行。您可以通过 APIcast 在端口 **8080** 上测试调用，并提供正确的身份验证凭据，您可以从 3scale 帐户获得这些凭据。

测试调用不仅验证 APIcast 是否在正确运行，还验证身份验证和报告是否得到成功处理。

注意

确保您用于调用的主机与 **Integration** 页面的 Public Base URL 字段中配置的主机相同。

3.4.4.3. podman 命令选项

您可以在 **podman** 命令中使用以下选项示例：

- **-d** ：在分离模式中运行容器，并打印容器 ID。如果未指定，容器将以前台模式运行，您可以使用 **CTRL + c** 来停止容器。以分离模式启动时，您可以使用 **podman attach** 命令重新附加到容器，例如 **podman attach apicast**。
- **ps** 和 **-a** ：Podman **ps** 用于列出创建和运行容器。将 **-a** 添加到 **ps** 命令将显示所有运行和停止的容器，例如 **podman ps -a**。
- **inspect** 和 **-l**：检查正在运行的容器。例如，使用 **inspect** 查看分配给容器的 ID。使用 **-l** 获取最新容器的详细信息，例如 **podman inspect -l | grep Id\":**。

3.4.4.4. 其他资源

- Red Hat 3scale API Management Platform 支持的配置
- Podman 的基本设置和使用

3.5. 使用操作器部署 APICAST 网关自助管理解决方案

本指南提供了通过 Openshift Container Platform 控制台使用 APIcast operator 部署 APIcast 网关自助管理解决方案的步骤。

部署 APIcast 时，默认设置适用于生产环境。您始终可以调整这些设置来部署 staging 环境。例如，使用以下 **oc** 命令：

```
$ oc patch apicast/{apicast_name} --type=merge -p '{"spec":
{"deploymentEnvironment":"staging","configurationLoadMode":"lazy"}}'
```

如需更多信息，请参阅：APIcast 自定义资源参考

先决条件

- OpenShift 容器平台(OCP)4.x 或更高版本，具有管理员特权。
- * 您遵循在 OpenShift 上安装 APIcast operator 中的步骤。

流程

1. 使用具有管理员特权的帐户登录 OCP 控制台。

2. 点 **Operators > Installed Operators**。
3. 从 Installed Operators 列表中点 APIcast Operator。
4. 点 **APIcast > Create APIcast**。

3.5.1. APIcast 部署和配置选项

您可以使用两种方法部署和配置 APIcast 网关自我管理解决方案：

- 提供 3scale API 管理系统端点
- 提供配置 secret

另请参阅：

- 使用 APIcast operator 注入自定义虚拟环境
- 使用 APIcast operator 注入自定义策略
- 使用 APIcast operator 配置 OpenTracing

3.5.1.1. 提供 3scale API 管理系统端点

流程

1. 创建一个包含 3scale 系统管理门户端点信息的 OpenShift secret：

```
$ oc create secret generic ${SOME_SECRET_NAME} --from-literal=AdminPortalURL=${MY_3SCALE_URL}
```

- **${SOME_SECRET_NAME}** 是 secret 的名称，只要它与现有 secret 不冲突，可以是您想要的任何名称。
- **${MY_3SCALE_URL}** 是包含 3scale 访问令牌和 3scale 系统门户端点的 URI。如需了解更多详细信息，请参阅 **THREESCALE_PORTAL_ENDPOINT**

  Example

```
$ oc create secret generic 3scaleportal --from-literal=AdminPortalURL=https://access-token@account-admin.3scale.net
```

  有关机密内容的更多信息，请参阅 管理门户配置 secret 参考。

2. 为 APIcast 创建 OpenShift 对象

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
spec:
  adminPortalCredentialsRef:
    name: SOME_SECRET_NAME
```

spec.adminPortalCredentialsRef.name 必须是包含 3scale 系统管理门户端点信息的现有 OpenShift secret 的名称。

3. 确认与 APIcast 对象关联的 OpenShift Deployment 的 **readyReplicas** 字段是否为 1，以验证 APIcast 容器集正在运行并已就绪。或者，等待字段设置为：

```
$ echo $(oc get deployment apicast-example-apicast -o jsonpath='{.status.readyReplicas}')
1
```

3.5.1.1.1. 验证 APIcast 网关正在运行且可用

流程

1. 确保 OpenShift Service APIcast 已公开给您的本地计算机，并执行测试请求。通过将 APIcast OpenShift Service 端口转发到 **localhost:8080** 来做到这一点：

```
$ oc port-forward svc/apicast-example-apicast 8080
```

2. 向配置的 3scale 服务发出请求，以验证 HTTP 响应是否成功。使用在服务的 **Staging Public Base URL** 或 **Production Public Base URL** 设置中配置的域名。例如：

```
$ curl 127.0.0.1:8080/test -H "Host: localhost"
{
  "method": "GET",
  "path": "/test",
  "args": "",
  "body": "",
  "headers": {
    "HTTP_VERSION": "HTTP/1.1",
    "HTTP_HOST": "echo-api.3scale.net",
    "HTTP_ACCEPT": "*/*",
    "HTTP_USER_AGENT": "curl/7.65.3",
    "HTTP_X_REAL_IP": "127.0.0.1",
    "HTTP_X_FORWARDED_FOR": ...
    "HTTP_X_FORWARDED_HOST": "echo-api.3scale.net",
    "HTTP_X_FORWARDED_PORT": "80",
    "HTTP_X_FORWARDED_PROTO": "http",
    "HTTP_FORWARDED": "for=10.0.101.216;host=echo-api.3scale.net;proto=http"
  },
  "uuid": "603ba118-8f2e-4991-98c0-a9edd061f0f0"
```

3.5.1.1.2. 通过 Kubernetes 入口公开 APIcast

要通过 Kubernetes Ingress 向外部公开 APIcast，请设置并配置 **exposeHost** 部分。当设置 **exposeHost** 部分中的 **host** 字段时，这会创建一个 Kubernetes Ingress 对象。然后，以前安装的和现有的 Kubernetes Ingress Controller 可以使用 Kubernetes Ingress 对象，使 APIcast 从外部访问。

要了解 Ingress Controller 可用于使 APIcast 外部访问，以及如何配置它们，请参阅 Kubernetes Ingress Controller 文档。

以下示例使用主机名 **myhostname.com** 公开 APIcast：

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
```

```
metadata:
  name: example-apicast
spec:
  ...
  exposedHost:
    host: "myhostname.com"
  ...
```

这个示例使用 HTTP 在端口 80 上创建一个 Kubernetes Ingress 对象。当 APIcast 部署位于 OpenShift 环境中时，OpenShift 默认 Ingress Controller 将使用 Ingress 对象 APIcast 创建 Route 对象，以允许从外部访问 APIcast 安装。

您也可以为 **exposeHost** 部分配置 TLS。下表中可用字段的详情：

表 3.3. APIcastExposedHost 参考表

json/yaml 项	类型	必填	默认值	描述
主机	string	是	N/A	路由到网关的域名
tls	[]networkv1.IngressTLS	否	N/A	ingress TLS 对象的数组。请参阅 TLS 的更多信息。

3.5.1.2. 提供配置 secret

流程

1. 使用配置文件创建 secret：

```
$ curl https://raw.githubusercontent.com/3scale/APIcast/master/examples/configuration/echo.json -o $PWD/config.json

$ oc create secret generic apicast-echo-api-conf-secret --from-file=$PWD/config.json
```

配置文件必须名为 **config.json**。这是一个 APIcast CRD 引用 要求。

有关机密内容的更多信息，请参阅 管理门户配置 secret 参考。

2. 创建 APIcast 自定义资源：

```
$ cat my-echo-apicast.yaml
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: my-echo-apicast
spec:
  exposedHost:
    host: YOUR DOMAIN
  embeddedConfigurationSecretRef:
    name: apicast-echo-api-conf-secret

$ oc apply -f my-echo-apicast.yaml
```

a. *以下是嵌入式配置 secret 的示例：*

```
apiVersion: v1
kind: Secret
metadata:
  name: SOME_SECRET_NAME
type: Opaque
stringData:
  config.json: |
    {
      "services": [
       {
         "proxy": {
          "policy_chain": [
           { "name": "apicast.policy.upstream",
             "configuration": {
              "rules": [{
                "regex": "/",
                "url": "http://echo-api.3scale.net"
              }]
             }
           }
          ]
         }
       }
      ]
    }
```

3. *在创建 APIcast 对象时设置以下内容：*

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: example-apicast
spec:
  embeddedConfigurationSecretRef:
    name: SOME_SECRET_NAME
```

***spec.embeddedConfigurationSecretRef.name** 必须是包含网关配置的现有 OpenShift secret 的名称。*

4. *确认与 APIcast 对象关联的 OpenShift Deployment 的 **readyReplicas** 字段是否为 1，以验证 APIcast 容器集正在运行并已就绪。或者，等待字段设置为：*

```
$ echo $(oc get deployment apicast-example-apicast -o jsonpath='{.status.readyReplicas}')
1
```

3.5.1.2.1. 验证 APIcast 网关正在运行且可用

流程

1. *确保 OpenShift Service APIcast 已公开给您的本地计算机，并执行测试请求。通过将 APIcast OpenShift Service 端口转发到 **localhost:8080** 来做到这一点：*

```
$ oc port-forward svc/apicast-example-apicast 8080
```

a. 接下来：向配置的 3scale 服务发出请求，并验证 HTTP 响应是否成功。

3.5.1.3. 使用 APIcast operator 注入自定义虚拟环境

在使用自我管理的 APIcast 的 3scale 安装中，您可以使用 **APIcast** operator 注入自定义环境。自定义环境定义 APIcast 适用于网关服务的所有上游 API 的行为。要创建自定义环境，请在 Lua 代码中定义全局配置。

您可以注入自定义环境，作为 APIcast 安装的一部分或安装后。在注入自定义环境后，您可以将其删除，**APIcast** operator 会协调更改。

先决条件

- 已安装 APIcast operator。

流程

1. 编写用于定义您要注入的自定义环境的 Lua 代码。例如，以下 **env1.lua** 文件显示 **APIcast** operator 为所有服务加载的自定义日志策略。

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

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

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

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

2. 从定义自定义环境的 Lua 文件创建 secret。例如：

```
$ oc create secret generic custom-env-1 --from-file=./env1.lua
```

secret 可以包含多个自定义虚拟环境。为定义自定义环境的每个文件指定 **-from-file** 选项。Operator 会加载每个自定义环境。

3. 定义一个 **APIcast** 自定义资源，用于引用您刚才创建的 secret。以下示例显示了相对于引用定义自定义环境的 secret 的内容。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
```

```
spec:
  customEnvironments:
    - secretRef:
        name: custom-env-1
```

APIcast 自定义资源可以引用定义自定义虚拟环境的多个 secret。Operator 会加载每个自定义环境。

4. 创建 **APIcast** 自定义资源来添加自定义虚拟环境。例如，如果您在 **apicast.yaml** 文件中保存 **APIcast** 自定义资源，请运行以下命令：

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

后续步骤

如果更新您的自定义环境，请确保重新创建其 secret，以便 secret 包含更新。**APIcast** operator 会监视是否有更新，并在找到更新时自动重新部署。

3.5.1.4. 使用 APIcast operator 注入自定义策略

在使用自助管理的 APIcast 的 3scale 安装中，您可以使用 **APIcast** operator 注入自定义策略。注入自定义策略会将策略代码添加到 APIcast。然后，您可以使用以下任一策略将自定义策略添加到 API 产品策略链中：

- 3scale API
- **产品**自定义资源

要使用 3scale 管理门户将自定义策略添加到产品策略链中，还必须使用 **CustomPolicyDefinition** 自定义资源注册自定义策略的 schema。自定义策略注册是只有在您要使用管理门户配置产品策略链时才需要。

您可以注入自定义策略，作为 APIcast 安装的一部分或之后。注入自定义策略后，您可以将其删除，并且 **APIcast** 运算符协调更改。

先决条件

- APIcast operator 已安装，或正在安装它。
- 您已定义一个自定义策略，如写您自己的策略中所述。即，已创建了定义自定义策略的 **my-first-custom-policy.lua**、**apicast-policy.json** 和 **init.lua** 文件，

流程

1. 从定义一个自定义策略的文件创建 secret。例如：

```
$ oc create secret generic my-first-custom-policy-secret \
 --from-file=./apicast-policy.json \
 --from-file=./init.lua \
 --from-file=./my-first-custom-policy.lua
```

如果您有多个自定义策略，请为每个自定义策略创建一个 secret。secret 只能包含一个自定义策略。

2. 定义一个 **APIcast** 自定义资源，用于引用您刚才创建的 secret。以下示例显示有关引用定义自定义策略的 secret 的内容。

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  customPolicies:
    - name: my-first-custom-policy
      version: "0.1"
      secretRef:
        name: my-first-custom-policy-secret
```

APIcast 自定义资源可以引用定义自定义策略的多个 secret。Operator 加载每个自定义策略。

3. 创建 **APIcast** 自定义资源，以添加自定义策略。例如，如果您在 **apicast.yaml** 文件中保存 **APIcast** 自定义资源，请运行以下命令：

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

后续步骤

如果更新自定义策略，请确保重新创建其 secret，以便 secret 包含更新。**APIcast** operator 会监视是否有更新，并在找到更新时自动重新部署。

其他资源

- APIcast 自定义资源定义

3.5.1.5. 使用 APIcast operator 配置 OpenTracing

在使用自助管理的 APIcast 的 3scale 安装中，您可以使用 **APIcast** operator 来配置 OpenTracing。通过启用 OpenTracing，您可以在 APIcast 实例上获取更多见解和更好的可观察性。

先决条件

- **APIcast** operator 已安装，或正在安装它。

流程

1. 在 **stringData.config** 中包含您的 OpenTracing 配置详情的 secret。这是包含您的 OpenTracing 配置详细信息的属性的唯一有效值。任何其他规格都阻止 APIcast 收到您的 OpenTracing 配置详细信息。以下示例显示了一个有效的 secret 定义：

```
apiVersion: v1
kind: Secret
metadata:
  name: myjaeger
stringData:
  config: |-
      {
      "service_name": "apicast",
      "disabled": false,
      "sampler": {
        "type": "const",
        "param": 1
      },
```

```
    "reporter": {
      "queueSize": 100,
      "bufferFlushInterval": 10,
      "logSpans": false,
      "localAgentHostPort": "jaeger-all-in-one-inmemory-agent:6831"
    },
    "headers": {
      "jaegerDebugHeader": "debug-id",
      "jaegerBaggageHeader": "baggage",
      "TraceContextHeaderName": "uber-trace-id",
      "traceBaggageHeaderPrefix": "testctx-"
    },
    "baggage_restrictions": {
        "denyBaggageOnInitializationFailure": false,
        "hostPort": "127.0.0.1:5778",
        "refreshInterval": 60
    }
    }
type: Opaque
```

2. *创建 secret.例如，如果您将以前的 secret 定义保存到* ***myjaeger.yaml*** *文件中，您将运行以下命令：*

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

3. *定义一个* ***APIcast*** *自定义资源，指定* ***OpenTracing*** *属性。在 CR 定义中，将* ***spec.tracingConfigSecretRef.name*** *属性设置为包含您的 OpenTracing 配置详情的 secret 的名称。下例仅显示了与配置 OpenTracing 相关的内容。*

```
apiVersion: apps.3scale.net/v1alpha1
kind: APIcast
metadata:
  name: apicast1
spec:
  ...
  openTracing:
    enabled: true
    tracingConfigSecretRef:
      name: myjaeger
    tracingLibrary: jaeger
...
```

4. *创建配置 OpenTracing 的* ***APIManager*** *自定义资源。例如，如果您将* ***APIcast*** *自定义资源保存在* ***apicast1.yaml*** *文件中，您将运行以下命令：*

```
$ oc apply -f apicast1.yaml
```

后续步骤

根据 OpenTracing 的安装方式，您应该在 Jaeger 服务用户界面中看到 trace。

其他资源

- *APIcast 自定义资源定义*

3.6. 其他资源

要获取有关最新发布的及受支持 APIcast 版本的信息，请参阅文章：

- Red Hat 3scale API Management Platform 支持的配置
- 红帽 3scale API 管理 - 组件详情。

第 4 章 用于 3SCALE API 管理中的高可用性支持的外部 REDIS 数据库配置

重要

红帽支持使用外部 Redis 数据库的 3scale 配置。但是，不支持为零停机时间设置 Redis、为 3scale 配置后端组件或 Redis 数据库复制和分片。其内容仅供参考。另外，3scale 不支持 Redis ***集群模式***。

OpenShift 容器平台(OCP)为大多数组件提供高可用性(HA)。

注意

从 Red Hat 3scale API 管理部署外部化数据库时，这意味着与应用程序分离，并对在数据库一级的服务中断有一定的弹性。服务中断的恢复取决于您托管数据库的基础架构或平台供应商提供的服务级别协议 (SLA)。3scale 不提供此功能。有关您选择的部署提供的数据库外部化的详情，请查看相关的文档。

红帽 3scale API 管理中的 HA 数据库组件包括：

- ***backend-redis*** *：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 *和* ***system-dis*** *都可以与 Redis Sentinel 和 Redis Enterprise 支持的 Redis 高可用性变体一起工作。*

如果 Redis pod 进入停止状态，或者 OpenShift Container Platform 停止它，则会自动创建新的 pod。持久存储将恢复数据，以便 pod 继续工作。在这些情况下，新容器集启动时会出现少量停机时间。这是因为 Redis 中不支持多主设置的限制。您可以通过在已部署 Redis 的所有节点上预安装 Redis 镜像来缩短停机时间。这将加快 pod 重启时间。

为零停机时间设置 Redis，并为 3scale 配置后端组件：

- *为零停机时间设置 Redis*
- *为 3scale API 管理配置后端组件*
- *重新删除数据库分片和复制*

先决条件

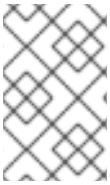
- *具有管理员角色的 3scale 帐户。*

4.1. 为零停机时间设置 REDIS

以 3scale 管理员身份，如果您不需要停机，请在 OCP 外部配置 Redis。使用 3scale pod 的配置选项进行设置的方法有几种：

- *设置您自己的自助管理的 Redis*
- *使用 Redis Sentinel：参考 Redis Sentinel 文档*

- redis 作为服务提供：
  例如：
  - Amazon ElastiCache
  - redis Labs

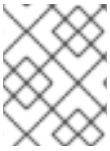


注意

红帽不支持上述服务。提及任何此类服务并不意味着红帽认可这些产品或服务。您同意，由于您使用（或依赖）任何外部内容而可能导致的任何损失或费用，红帽不承担任何责任。

4.2. 为 3SCALE API 管理配置后端组件

作为 3scale 管理员，在以下部署配置中为 **后端**组件环境变量配置 Redis HA(failover)：**backend-cron**、**backend-listener** 和 **backend-worker**。这些配置是 3scale 中红帽 HA 所必需的。



注意

如果要将 Redis 与 sentinels 搭配使用，您必须在 **backend-redis**、**system-redis** 或两个 secret 中提供 sentinel 配置。

4.2.1. 创建 backend-redis 和 system-redis secret

按照以下步骤相应地创建 **backend-redis** 和 **system-redis** secret：

- 为 HA 部署 3scale API 管理的新安装
- 将 3scale API 管理的非 HA 部署迁移到 HA

4.2.2. 为 HA 部署 3scale API 管理的新安装

流程

1. 使用以下字段创建 **backend-redis** 和 **system-redis** secret：

   backend-redis

   ```
   REDIS_QUEUES_SENTINEL_HOSTS
   REDIS_QUEUES_SENTINEL_ROLE
   REDIS_QUEUES_URL
   REDIS_STORAGE_SENTINEL_HOSTS
   REDIS_STORAGE_SENTINEL_ROLE
   REDIS_STORAGE_URL
   ```

   system-redis

   ```
   NAMESPACE
   SENTINEL_HOSTS
   SENTINEL_ROLE
   URL
   ```

- 当使用 sendinels 为 Redis 配置时，**backend-redis** 和 **system-redis** 中的对应 **URL** 字段以 **redis://[:redis-password@]redis-group[/db]** 格式引用 Redis 组，其中 [x] 表示可选元素 **x**，**redis-password**、**redis-group** 和 **db** 是变量：

  Example

  ```
  redis://:redispwd@mymaster/5
  ```

- **SENTINEL_HOSTS** 字段采用以下格式以逗号分隔：

  ```
  redis://:sentinel-password@sentinel-hostname-or-ip:port
  ```

  - 对于列表的每个元素，[x] 表示可选元素 **x**，**sendinel-password**、**finel-hostname-or-ip**，**port** 是相应地替换的变量：

    Example

    ```
    :sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722
    ```

- **SENTINEL_ROLE** 字段为 **master** 或 **slave** 字段。

2. 按照使用 operator 部署 3scale API 管理中所述 部署 3scale。

   a. 忽略因为 **backend-redis** 和 **system-redis** 已存在的错误。

4.2.3. 将 3scale API 管理的非 HA 部署迁移到 HA

1. 使用所有字段编辑 **backend-redis** 和 **system-redis** secret，如 部署 3scale API Management for HA 所示。

2. 确保为后端容器集定义了以下 **backend-redis** 环境变量。

```
name: BACKEND_REDIS_SENTINEL_HOSTS
  valueFrom:
    secretKeyRef:
      key: REDIS_STORAGE_SENTINEL_HOSTS
      name: backend-redis
name: BACKEND_REDIS_SENTINEL_ROLE
  valueFrom:
    secretKeyRef:
      key: REDIS_STORAGE_SENTINEL_ROLE
      name: backend-redis
```

3. 确保为 **system-(app|sidekiq)** pod 定义了以下 **system-redis** 环境变量。

```
name: REDIS_SENTINEL_HOSTS
  valueFrom:
    secretKeyRef:
      key: SENTINEL_HOSTS
      name: system-redis
name: REDIS_SENTINEL_ROLE
  valueFrom:
```

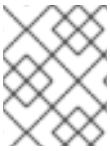
```
secretKeyRef:
  key: SENTINEL_ROLE
  name: system-redis
```

4. 继续执行说明以使用模板升级 3scale。

4.2.3.1. 使用 Redis Enterprise

1. 在 OpenShift 中使用 Redis Enterprise，具有三个不同的 **redis-enterprise** 实例：
   a. 编辑 **system-redis** secret:
      i. 将 **system-redis** 中的系统 redis 数据库设置为 **URL**。
   b. 将 **backend-redis** 中的后端数据库设置为 **REDIS_QUEUES_URL**。
   c. 将第三个数据库设置为 **backend-redis** 的 **REDIS_STORAGE_URL**。

4.2.3.2. 使用 Redis Sentinel



注意

您可以选择将 Redis Sentinels 应用到任何数据库。但是，红帽建议将 Redis Sentinels 应用到所有系统以实现 HA。

1. 用于统计信息的后端 redis：更新 **backend-redis** secret 并为其提供值：
   - ***REDIS_STORAGE_URL***
   - ***REDIS_STORAGE_SENTINEL_ROLE***
   - ***REDIS_STORAGE_SENTINEL_HOSTS***
     将 **REDis_STORAGE_SENTINEL_ROLE** 设置为一个以逗号分隔的主机和端口列表，例如：
     :sentinelpwd@123.45.67.009:2711,:sentelpwd@other-sentinel:2722
2. 用于队列的后端 redis：更新 **backend-redis** secret 并为其提供值：
   - ***REDIS_QUEUES_URL***
   - ***REDIS_QUEUES_SENTINEL_ROLE***
   - ***REDIS_QUEUES_SENTINEL_HOSTS***
     将 **REDIS_QUEUES_SENTINEL_ROLE** 设置为一个以逗号分隔的主机和端口列表，例如：
     :sentinelpwd@123.45.67.009:2711,:sentelpwd@other-sentinel:2722
3. 使用 Redis 数据库的 Redis Sentinel：
4. 用于数据的 redis：更新 **system-redis** secret 并为其提供值：

注意

*编辑 **system-redis** secret: **URL***

- ***SENTINEL_ROLE***

- ***NAMESPACE***
- ***URL***
- ***SENTINEL_HOSTS***
  将 ***SENTINEL_HOSTS*** *设置为以逗号分隔的主机和端口列表，例如：*
  :sentinelpwd@123.45.67.009:2711,:sentinelpwd@other-sentinel:2722

备注

- *system-app 和 system-sidekiq 组件直接连接到* ***后端*** *Redis 以检索统计信息。*
  - *从 3scale 2.7 开始，使用发送时这些系统组件也可以连接到* ***后端*** *Redis（存储）。*
- *system-app 和 system-sidekiq 组件* ***仅使用 backend-redis*** *存储，而不使用* ***backend-redis*** *队列。*
  - *对系统组件所做的更改支持带有 sendinels 的* ***backend-redis*** *存储。*

4.3. 重新删除数据库分片和复制

分片有时称为分区，可将大型数据库隔离为称为分片的较小数据库。通过复制，您的数据库使用托管在独立计算机上的同一数据集的副本进行设置。

分片

分片有助于添加更多领导实例，当您有如此多的数据并不适用于单个数据库中，或者 CPU 负载接近 100% 时，这也会很有用。

使用 Red Hat is HA for 3scale 时，以下两个原因是分片非常重要的原因：

- *拆分和扩展大量数据，并调整特定索引的分片数量，以帮助避免瓶颈。*
- *跨不同节点分布操作，因此可以提高性能，例如当多台机器处理同一个查询时。*

禁用集群模式的 Redis 数据库分片的三个主要解决方案是：

- *Amazon ElastiCache*
- *通过 Redis 发送的标准 Redis*
- *redis Enterprise*

复制

redis 数据库复制通过在不同机器之间复制您的数据集来确保冗余性。通过利用复制功能，您可以在领导机停机时让 Redis 保持工作。然后，从单个实例（领导）中拉取数据，以确保高可用性。

通过适用于 3scale 的 HA，数据库复制可确保主分片的高可用性副本。操作原则包括：

- *当主分片失败时，副本分片将自动提升到新的主分片。*
- *恢复原始主分片后，它会自动成为新主分片的副本分片。*

Redis 数据库复制的三个主要解决方案是：

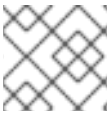
- *redis Enterprise*

- Amazon ElastiCache
- 通过 Redis 发送的标准 Redis

使用 **twemproxy** 进行分片

对于 Amazon ElastiCache 和 Standard Redis，分片涉及基于密钥分割数据。您需要给定特定密钥的代理组件知道要查找的分片，如 **twemproxy**。**twemproxy** 也称为 nutcracker，它是一个适用于 Redis 协议的轻量级代理解决方案，它根据特定的密钥或服务器映射找到分配给它们的分片。使用 **twemproxy** 在 Amazon ElastiCache 或 Standard Redis 实例中添加分片功能具有以下优点：

- 在多个服务器间自动分片数据的功能。
- 支持多种哈希模式，支持一致的散列和发行版。
- 在多个实例中运行的能力，允许客户端连接到第一个可用的代理服务器。
- 减少与后端上缓存名称服务器的连接数量。



注意

redis Enterprise 使用自己的代理，因此不需要 **twemproxy**。

其他资源

- redis Sentinel 文档。
- **twemproxy**。

4.4. 附加信息

- 有关 3scale 和 Red Hatis 数据库支持的更多信息，请参阅 Red Hat 3scale API 管理支持的配置 。
- 有关 Redis 的 Amazon ElastiCache 的更多信息，请参阅官方 Amazon ElastiCache 文档。
- 有关 Redis Enterprise 的详情请参考最新的文档。

第 5 章 配置外部 MYSQL 数据库

重要

从 Red Hat 3scale API 管理部署外部化数据库时，这意味着与应用程序分离，并对在数据库一级的服务中断有一定的弹性。服务中断的恢复取决于您托管数据库的基础架构或平台供应商提供的服务级别协议 (SLA)。3scale 不提供此功能。有关您选择的部署提供的数据库外部化的详情，请查看相关的文档。

红帽支持使用外部 MySQL 数据库的 3scale 配置。但是，数据库本身不在支持范围之内。

本指南提供有关外部化 MySQL 数据库的信息。当使用默认的 ***system-mysql*** *容器集存在多个基础架构问题（如网络或文件系统）时，这非常有用。*

先决条件

- *使用具有管理员特权的帐户访问 OpenShift Container Platform 4.x 集群。*
- *在 OpenShift 集群上安装 3scale 实例。请参阅在 OpenShift 上安装 3scale API 管理 。*

要配置外部 MySQL 数据库，请执行以下部分中所述的步骤：

- *第 5.1 节 “外部 MySQL 数据库限制”*
- *第 5.2 节 “外部化 MySQL 数据库”*
- *第 5.3 节 “回滚”*

5.1. 外部 MYSQL 数据库限制

外部化 MySQL 数据库的过程存在一些限制：

3scale 内部内部版本

它仅在 3scale 的 2.5 个内部版本和 2.6 内部部署版本中进行测试并验证。

MySQL 数据库用户

URL 必须为以下格式：

<database_scheme>://<admin_user>:<admin_password>@<database_host>/<database_name>

***<admin_user>** 必须是外部数据库中的现有用户，具有 **<database_name>** 逻辑数据库的完整权限。**<database_name>** 必须是外部数据库中已存在的逻辑数据库。*

MySQL 主机

使用来自外部 MySQL 数据库的 IP 地址，而不是 主机名，否则不会解析。例如，使用 1.1.1.1 而不是 mysql.mydomain.com。

5.2. 外部化 MYSQL 数据库

使用以下步骤，将 MySQL 数据库完全外部化。

警告

这将在进程持续期间导致环境中的停机。

流程

1. *登录到托管 3scale On-premises 实例的 OpenShift 节点，并更改到其项目：*

```
$ oc login -u <user> <url>
$ oc project <3scale-project>
```

*将 **<user>**、**<url>** 和 **<3scale-project>** 替换为您自己的凭证和项目名称。*

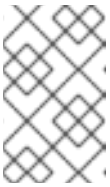
2. *按照下方的步骤，按照所示的顺序缩减所有 pod。这将避免丢失数据。*

停止 3scale 内部部署

从 OpenShift Web 控制台或命令行界面(CLI)，按以下顺序将所有部署配置缩减为零副本：

- ***apicast-wildcard-router** 和 **zync** 适用于 3scale 2.6 之前的版本，**zync-que** 和 **zync** 适用于 3scale 2.6 和以上版本。*
- ***apicast-staging** 和 **apicast-production**。*
- ***system-sidekiq**、**backend-cron** 和 **system-searchd**.*
  - *3scale 2.3 包括 **system-resque**。*
- ***system-app**。*
- ***backend-listener** 和 **backend-worker**。*
- ***backend-redis**、**system-memcache**、**system-mysql**、**system-redis** 和 **zync-database.** 以下示例演示了如何在 CLI 中为 **apicast-wildcard-router** 和 **zync** 执行此操作 ：*

```
$ oc scale dc/apicast-wildcard-router --replicas=0
$ oc scale dc/zync --replicas=0
```



注意

*可以同时缩减每个步骤的部署配置。例如，您可以将 **apicast-wildcard-router** 和 **zync** 一起缩减。但是，最好等待每个步骤中的 pod 终止，然后再缩减后续 pod。3scale 实例将完全无法访问，直到它被完全启动。*

3. *要确认 3scale 项目上没有运行任何 pod，请使用以下命令：*

```
$ oc get pods -n <3scale_namespace>
```

命令应返回 No resources found。

4. *使用以下命令再次扩展数据库级别的 pod：*

```
$ oc scale dc/{backend-redis,system-memcache,system-mysql,system-redis,zync-database}
--replicas=1
```

5. 确保您可以通过 **system-mysql** pod 登录外部 MySQL 数据库，然后继续后续步骤：

```
$ oc rsh system-mysql-<system_mysql_pod_id>
mysql -u root -p -h <host>
```

- <system_mysql_pod_id>: system-mysql pod 的标识符。
- 用户应当始终为 root。如需更多信息，请参阅 外部 MySQL 数据库限制。

    a. CLI 现在将显示 **mysql>**。键入 exit，然后按 返回。在下一提示中再次键入 exit 以 返回到 OpenShift 节点控制台。

6. 使用以下命令执行完整的 MySQL 转储：

```
$ oc rsh system-mysql-<system_mysql_pod_id> /bin/bash -c "mysqldump -u root --single-
transaction --routines --triggers --all-databases" > system-mysql-dump.sql
```

- 将 <system_mysql_pod_id> 替换为您唯一的 **system-mysql** pod ID。
- 验证 **system-mysql-dump.sql** 是否包含有效的 MySQL 级别转储，如下例所示：

```
$ head -n 10 system-mysql-dump.sql
-- MySQL dump 10.13  Distrib 8.0, for Linux (x86_64)
--
-- Host: localhost    Database:
-- ------------------------------------------------------
-- Server version   8.0

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
*/;
/*!40101 SET NAMES utf8 */;
```

7. 缩减 **system-mysql** pod，并保留为 0（零）副本：

```
$ oc scale dc/system-mysql --replicas=0
```

8. 查找与 URL **mysql2://root:<password>@<host>/system** 等效的 base64，相应地替换 <password> 和 <host>：

```
$ echo "mysql2://root:<password>@<host>/system" | base64
```

9. 在远程 MySQL 数据库上创建默认的 'user'@'%'。它只需要具有 SELECT 特权。找到它的 base64 对等信息：

```
$ echo "user" | base64
$ echo "<password>" | base64
```

- 将 <password> 替换为 'user'@'%' 的密码。

10. 执行备份并编辑 OpenShift secret **system-database** ：

```
$ oc get secret system-database -o yaml > system-database-orig.bkp.yml
$ oc edit secret system-database
```

- **URL** ：使用 [step-8] 的值替换它。
- **DB_USER** 和 **DB_PASSWORD** ：使用上一步中的值。

11. 将 **system-mysql-dump.sql** 发送到远程数据库服务器，并将转储导入到其中。使用命令导入它：

12. 使用以下命令将 **system-mysql-dump.sql** 发送到远程数据库服务器，并将转储导入到服务器：

```
mysql -u root -p < system-mysql-dump.sql
```

13. 确保创建了名为 system 的新数据库：

```
mysql -u root -p -se "SHOW DATABASES"
```

14. 使用以下说明启动 3scale 内部部署，以正确顺序扩展所有 pod。

启动 3scale **内部部署**

- **backend-redis**、**system-memcache**、**system-mysql**、**system-redis** 和 **zync-database.**
- **backend-listener** 和 **backend-worker**。
- **system-app**。
- **system-sidekiq**、**backend-cron** 和 **system-searchd**
  - 3scale 2.3 包括 **system-resque**。
- **apicast-staging** 和 **apicast-production**。
- **apicast-wildcard-router** 和 **zync** 适用于 3scale 2.6 之前的版本， **zync-que** 和 **zync** 适用于 3scale 2.6 和以上版本。
  以下示例演示了如何在 CLI 中为 **backend-redis**、**system-memcache**、**system-mysql**、**system-redis** 和 **zync-database** 执行此操作 ：

```
$ oc scale dc/backend-redis --replicas=1
$ oc scale dc/system-memcache --replicas=1
$ oc scale dc/system-mysql --replicas=1
$ oc scale dc/system-redis --replicas=1
$ oc scale dc/zync-database --replicas=1
```

system-app pod 现在应该已启动并运行，且没有任何问题。

15. 验证后，按照所示的顺序扩展其他容器集。

16. 备份 **system-mysql** DeploymentConfig 对象。您可以在几分钟后删除所有内容，确定一切运行正常。如果将来再次执行此步骤，删除 **system-mysql** DeploymentConfig 可避免以后出现混淆。

5.3. 回滚

如果 **system-app** pod 没有完全在线，并且其根本原因在第 14 步 后无法决定或解决，请执行回滚步骤。

1. 使用来自 **system-database-orig.bkp.yml** 的原始值编辑 **system-database**。请参阅 [step-10]:

```
$ oc edit secret system-database
```

将 URL、DB_USER 和 DB_PASSWORD 替换为其原始值。

2. 缩减所有容器集，然后再次向上扩展，包括 **system-mysql**。**system-app** pod 和在启动后启动的其他 pod 应启动并再次运行。运行以下命令确认所有 pod 都已备份并正在运行：

```
$ oc get pods -n <3scale-project>
```

5.4. 附加信息

- 有关 3scale 和 MySQL 数据库支持的更多信息，请参阅 Red Hat 3scale API 管理支持的配置 。