Red Hat OpenShift Dev Spaces 3.14

管理指南

管理 Red Hat OpenShift Dev Spaces 3.14

Last Updated: 2024-06-11

Red Hat OpenShift Dev Spaces 3.14 管理指南

管理 Red Hat OpenShift Dev Spaces 3.14

Jana Vrbkova
jvrbkova@redhat.com

法律通告

摘要

管理员操作 Red Hat OpenShift Dev Spaces 的信息。

目录

第 1 章 准备安装

要准备 OpenShift Dev Spaces 安装，了解 OpenShift Dev Spaces 生态系统和部署限制：

- 第 1.1 节 “支持的平台”
- 第 1.2 节 “安装 dsc 管理工具”
- 第 1.3 节 “架构”
- 第 1.4 节 “计算 Dev Spaces 资源要求”
- 第 3.1 节 “了解 **CheCluster** 自定义资源”

1.1. 支持的平台

OpenShift Dev Spaces 在以下 CPU 架构上的 OpenShift 4.12-4.16 上运行：

- AMD64 和 Intel 64 (**x**86_64)
- IBM Power (**ppc64le**) 和 IBM Z (**s390x**)

其他资源

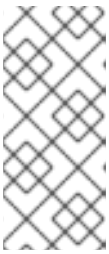
- OpenShift 文档

1.2. 安装 DSC 管理工具

您可以在 Microsoft Windows、Apple MacOS 和 Linux 上安装 **dsc**、Red Hat OpenShift Dev Spaces 命令行管理工具。使用 **dsc**，您可以对 OpenShift Dev Spaces 服务器执行操作，如启动、停止、更新和删除服务器。

先决条件

- Linux 或 macOS。



注意

有关在 Windows 上安装 **dsc**，请查看以下页面：

- https://developers.redhat.com/products/openshift-dev-spaces/download
- https://github.com/redhat-developer/devspaces-chectl

流程

1. 将存档从 https://developers.redhat.com/products/openshift-dev-spaces/download 下载到目录，如 **$HOME**。
2. 在存档上运行 **tar xvzf** 以提取 **/dsc** 目录。
3. 将提取的 **/dsc/bin** 子目录添加到 **$PATH**。

验证

- 运行 **dsc** 查看有关它的信息。

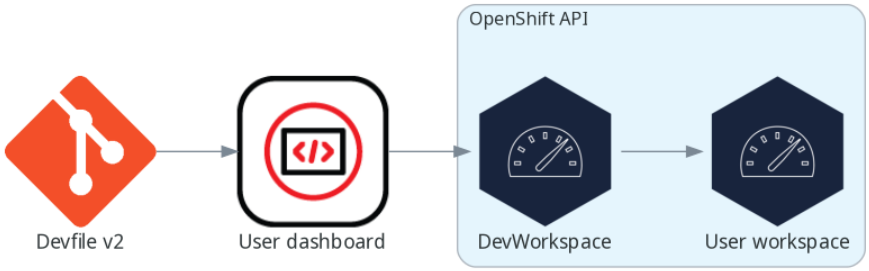
```
$ dsc
```

其他资源

- "DDS 参考文档"

1.3. 架构

图 1.1. 使用 Dev Workspace Operator 的高级别 OpenShift Dev Spaces 架构



OpenShift Dev Spaces 在三个组件组上运行：

OpenShift Dev Spaces 服务器组件
管理用户项目和工作区。主组件是 User 仪表板，用户从中控制其工作区。

dev Workspace operator
创建和控制运行用户工作区所需的 OpenShift 对象。包括 **Pod**、**服务** 和 **PersistentVolume**。

用户工作区
基于容器的开发环境，包含 IDE。

这些 OpenShift 功能的角色是中心的：

dev Workspace 自定义资源
代表用户工作区并由 OpenShift Dev Spaces 操作的有效 OpenShift 对象。它是三个组件组的通信频道。

OpenShift 基于角色的访问控制(RBAC)
控制所有资源的访问。

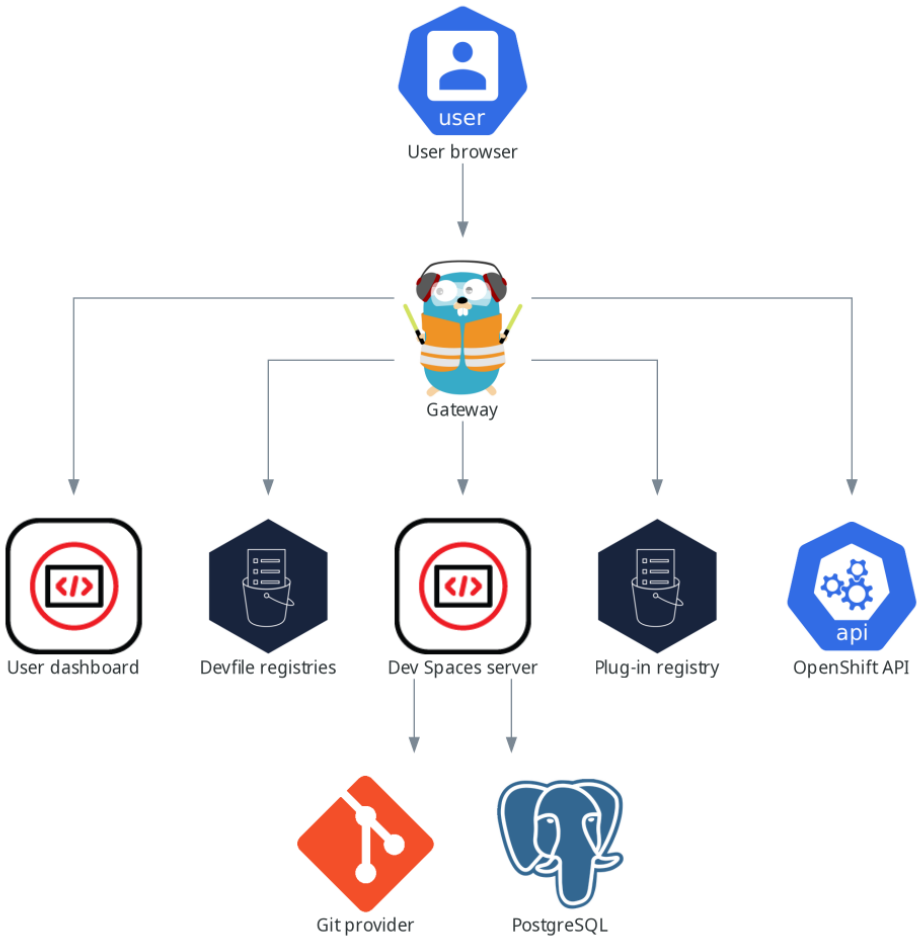
其他资源

- 第 1.3.1 节 “服务器组件”
- 第 1.3.1.2 节 “dev Workspace operator”
- 第 1.3.2 节 “用户工作区”
- dev Workspace Operator 存储库
- Kubernetes 文档 - 自定义资源

1.3.1. 服务器组件

OpenShift Dev Spaces 服务器组件确保多租户和工作区管理。

图 1.2. OpenShift Dev Spaces 服务器组件与 Dev Workspace operator 交互



其他资源

- 第 1.3.1.1 节 “dev Spaces operator”
- 第 1.3.1.2 节 “dev Workspace operator”
- 第 1.3.1.3 节 “gateway”
- 第 1.3.1.4 节 “用户仪表板”
- 第 1.3.1.5 节 “devfile registry”
- 第 1.3.1.6 节 “dev Spaces 服务器”
- 第 1.3.1.7 节 “插件 registry”

1.3.1.1. dev Spaces operator

OpenShift Dev Spaces operator 确保 OpenShift Dev Spaces 服务器组件的完整生命周期管理。它引入了：

CheCluster 自定义资源定义(CRD)

定义 **CheCluster** OpenShift 对象。

OpenShift Dev Spaces 控制器

创建和控制运行 OpenShift Dev Spaces 实例所需的 OpenShift 对象，如 pod、服务和持久性卷。

CheCluster 自定义资源(CR)

在使用 OpenShift Dev Spaces operator 的集群中，可以创建 **CheCluster** 自定义资源(CR)。OpenShift Dev Spaces operator 在此 OpenShift Dev Spaces 实例中确保对 OpenShift Dev Spaces 服务器组件的完整生命周期管理：

- 第 1.3.1.2 节 “dev Workspace operator”
- 第 1.3.1.3 节 “gateway”
- 第 1.3.1.4 节 “用户仪表板”
- 第 1.3.1.5 节 “devfile registry”
- 第 1.3.1.6 节 “dev Spaces 服务器”
- 第 1.3.1.7 节 “插件 registry”

其他资源

- 第 3.1 节 “了解 **CheCluster** 自定义资源”
- 第 2 章 *安装 Dev Spaces*

1.3.1.2. dev Workspace operator

Dev Workspace operator 扩展 OpenShift 以提供 Dev Workspace 支持。它引入了：

dev Workspace 自定义资源定义

定义 Devfile v2 规范中的 Dev Workspace OpenShift 对象。

dev Workspace 控制器

创建和控制运行 Dev Workspace 所需的 OpenShift 对象，如 Pod、服务和持久卷。

dev Workspace 自定义资源

在使用 Dev Workspace operator 的集群中，可以创建 Dev Workspace 自定义资源(CR)。Dev Workspace CR 是 Devfile 的 OpenShift 表示。它在 OpenShift 集群中定义用户工作区。

其他资源

- Devfile API 存储库

1.3.1.3. gateway

OpenShift Dev Spaces 网关有以下角色：

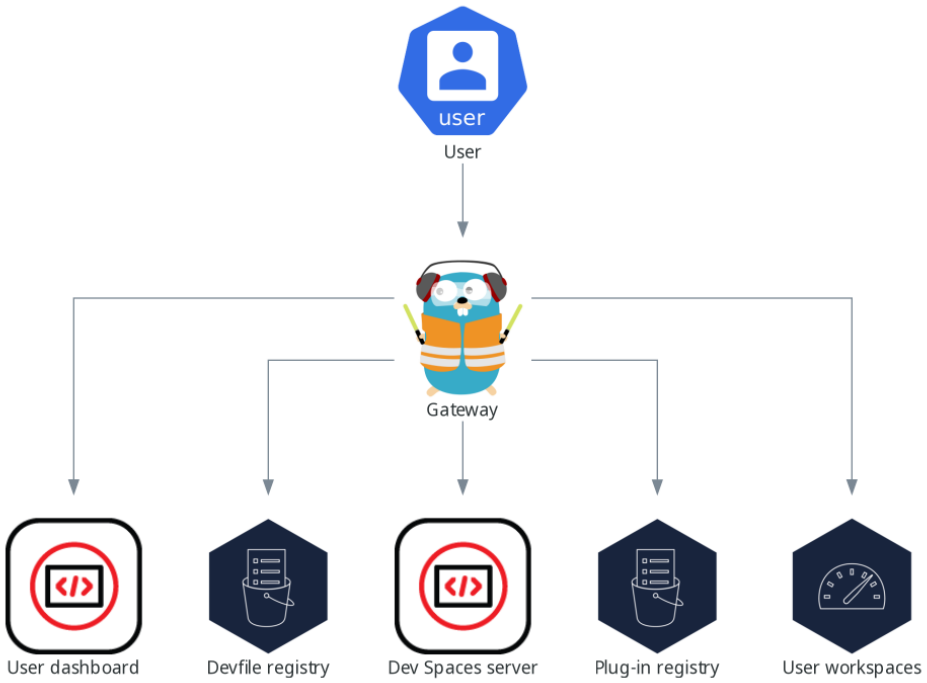
- 路由请求。它使用 Traefik。
- 使用 OpenID Connect (OIDC)验证用户。它使用 OpenShift OAuth2 代理。
- 应用 OpenShift 基于角色的访问控制(RBAC)策略来控制对任何 OpenShift Dev Spaces 资源的访问。它使用 'kube-rbac-proxy'。

OpenShift Dev Spaces operator 会作为 **che-gateway** 部署进行管理。

它控制对以下的访问：

- 第 1.3.1.4 节 “用户仪表板”
- 第 1.3.1.5 节 “devfile registry”
- 第 1.3.1.6 节 “dev Spaces 服务器”
- 第 1.3.1.7 节 “插件 registry”
- 第 1.3.2 节 “用户工作区”

图 1.3. OpenShift Dev Spaces 网关与其他组件交互



其他资源

- 第 3.10 节 “管理身份和授权”

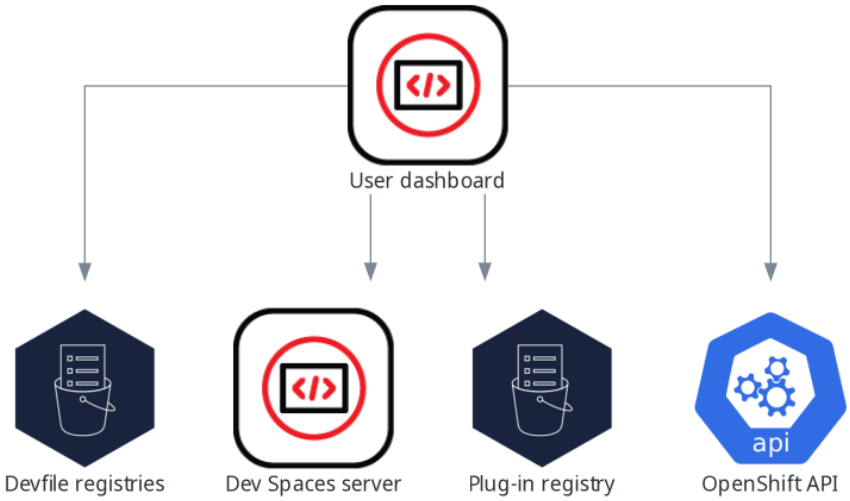
1.3.1.4. 用户仪表板

用户仪表板是 Red Hat OpenShift Dev Spaces 的登录页面。OpenShift Dev Spaces 用户浏览用户仪表板来访问和管理其工作区。它是 React 应用。OpenShift Dev Spaces 部署在 **devspaces-dashboard** Deployment 中启动它。

它需要访问：

- 第 1.3.1.5 节 “devfile registry”
- 第 1.3.1.6 节 “dev Spaces 服务器”
- 第 1.3.1.7 节 “插件 registry”
- OpenShift API

图 1.4. 用户仪表板与其他组件交互



当用户请求用户仪表板来启动工作区时，用户仪表板执行这个操作序列：

1. 当用户从代码示例创建工作区时，从 第 1.3.1.5 节 “devfile registry” 收集 devfile。
2. 当用户从远程 devfile 创建工作区时，请将存储库 URL 发送到 第 1.3.1.6 节 “dev Spaces 服务器”，并需要返回 devfile。
3. 读取描述工作区的 devfile。
4. 从 第 1.3.1.7 节 “插件 registry” 收集附加元数据。
5. 将信息转换为 Dev Workspace 自定义资源。
6. 使用 OpenShift API 在用户项目中创建 Dev Workspace 自定义资源。
7. 监视 Dev Workspace 自定义资源状态。
8. 将用户重定向到正在运行的工作区 IDE。

1.3.1.5. devfile registry

其他资源

OpenShift Dev Spaces devfile registry 是提供示例 devfile 列表的服务，以创建 ready-to-use 工作区。第 1.3.1.4 节 “用户仪表板” 在 **Dashboard → Create Workspace** 页面中显示示例列表。每个示例都包含一个 Devfile v2。OpenShift Dev Spaces 部署在 **devfile-registry** 部署中启动一个 devfile registry 实例。

图 1.5. devfile registry 与其他组件交互

其他资源

- devfile v2 文档
- devfile registry 最新社区版本在线实例
- OpenShift Dev Spaces devfile registry 存储库

1.3.1.6. dev Spaces 服务器

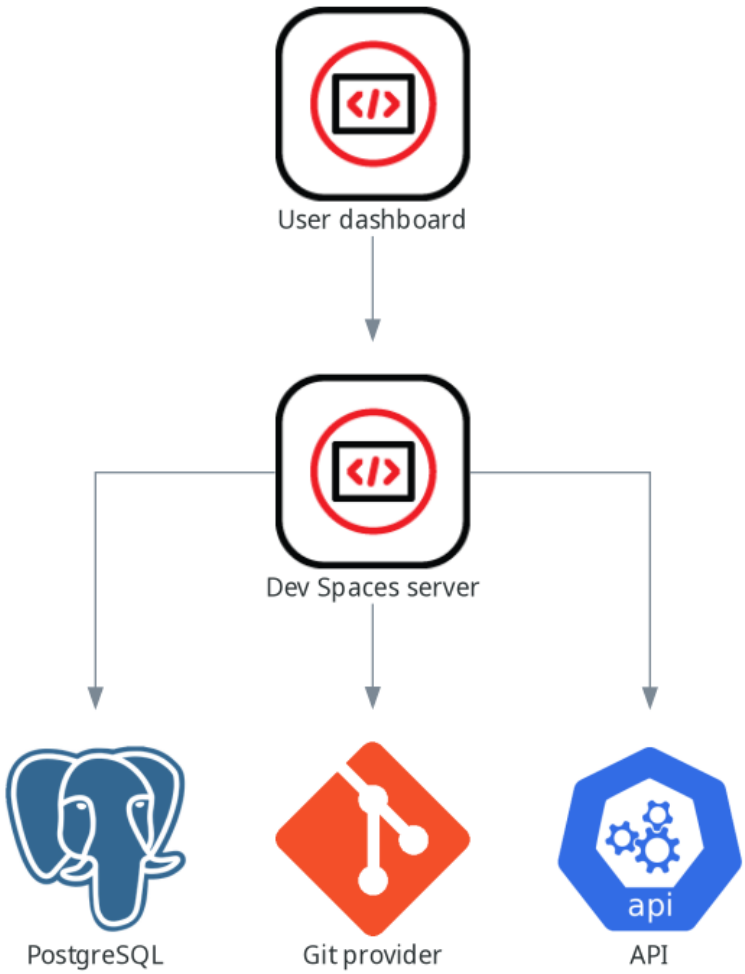
OpenShift Dev Spaces 服务器主要功能有：

- 创建用户命名空间。
- 使用所需的 secret 和配置映射置备用户命名空间。
- 与 Git 服务提供商集成，以获取并验证 devfile 和身份验证。

OpenShift Dev Spaces 服务器是一个 Java Web 服务，用于公开 HTTP REST API 并需要访问：

- Git 服务提供商
- OpenShift API

图 1.6. OpenShift Dev Spaces 服务器与其他组件交互



其他资源

- 第 3.3.2 节 “Dev Spaces 服务器的高级配置选项”

1.3.1.7. 插件 registry

每个 OpenShift Dev Spaces 工作区都以特定的编辑器以及一组关联的扩展开头。OpenShift Dev Spaces 插件 registry 提供了可用编辑器和编辑器扩展的列表。Devfile v2 描述了每个编辑器或扩展。

第 1.3.1.4 节 “用户仪表板” 正在读取 registry 的内容。

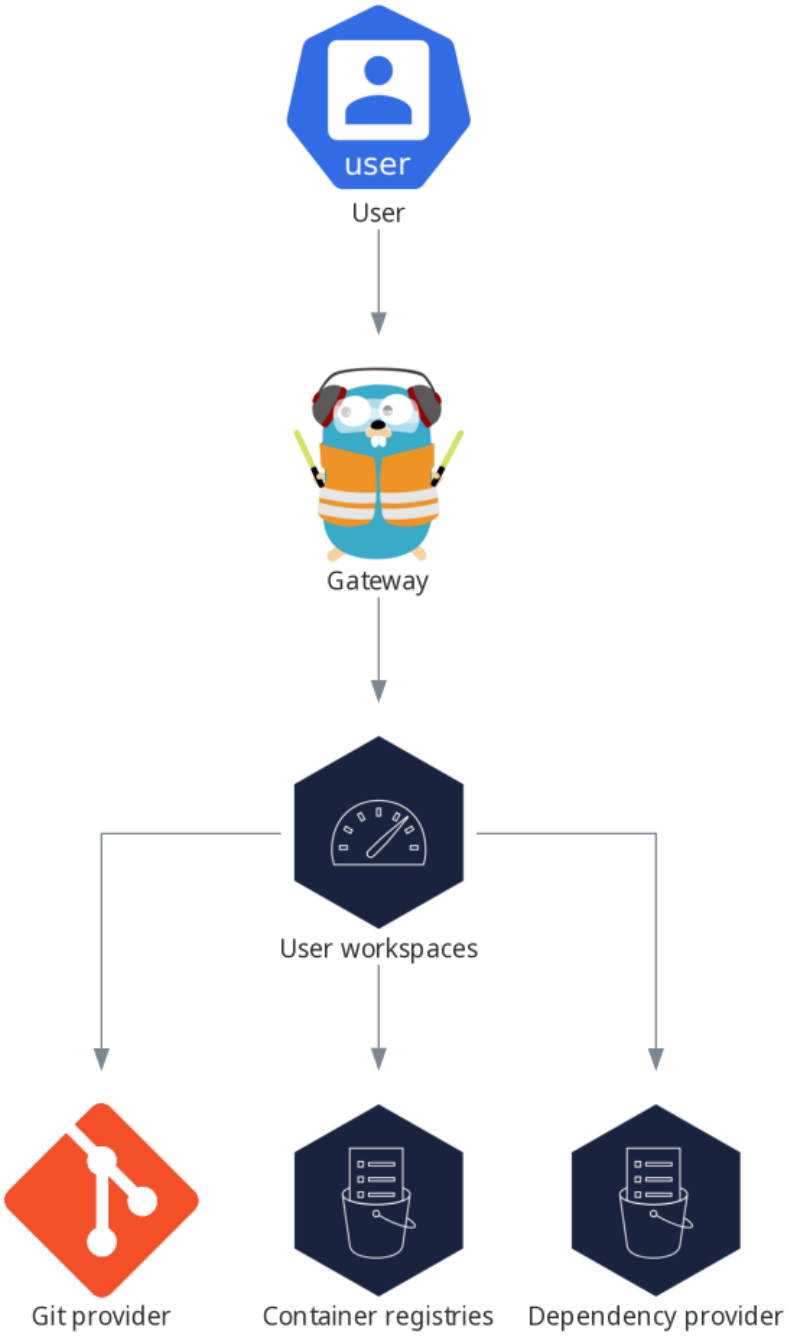
图 1.7. 插件 registry 与其他组件交互

其他资源

- OpenShift Dev Spaces 插件 registry 存储库中的编辑器定义
- 插件 registry 最新社区版本在线实例

1.3.2. 用户工作区

图 1.8. 用户工作区与其他组件交互



用户工作区是在容器中运行的 Web IDE。

用户工作区是一个 Web 应用。它由容器中运行的微服务组成，提供在浏览器中运行的现代 IDE 的所有服务：

- Editor
- 语言自动完成
- 语言服务器
- 调试工具
- 插件
- 应用程序运行时

工作区(workspace)是一个 OpenShift Deployment，其中包含工作区容器并启用的插件，以及相关的 OpenShift 组件：

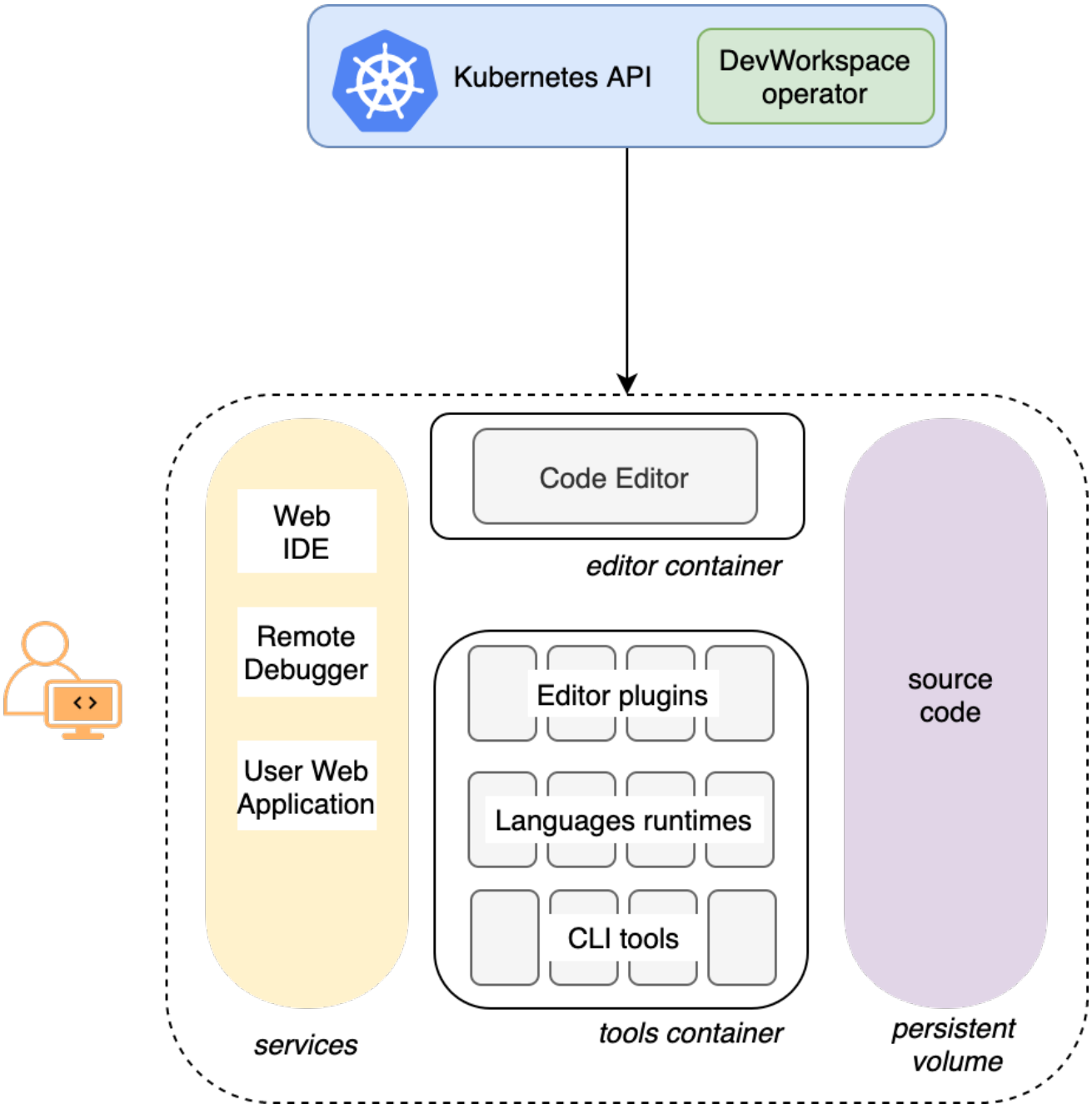
- 容器
- ConfigMaps
- 服务
- Endpoints
- 入口或路由
- Secrets
- 持久性卷(PV)

OpenShift Dev Spaces 工作区包含项目的源代码，并在 OpenShift 持久性卷(PV)中保留。微服务具有对此共享目录的读/写访问权限。

使用 devfile v2 格式指定 OpenShift Dev Spaces 工作区的工具和运行时应用程序。

下图显示了运行 OpenShift Dev Spaces 工作区及其组件。

图 1.9. OpenShift Dev Spaces 工作区组件



在图中，有一个正在运行的工作区。

1.4. 计算 DEV SPACES 资源要求

OpenShift Dev Spaces Operator、Dev Workspace Controller 和用户工作区由一组 pod 组成。pod 有助于 CPU 和内存限值和请求的资源消耗。

注意

以下到 示例 devfile 的链接是指向来自上游社区的材料的指针。本材料代表了最新可用内容和最新的最佳实践。这些提示尚未被红帽的 QE 部门审查，它们尚未被广泛的用户组验证。请谨慎使用此信息。它最适合用于教育和"开发"目的，而不是"生产"目的。

流程

1. 识别取决于用于定义开发环境的 devfile 的工作区资源要求。这包括识别 devfile 的 **components** 部分明确指定的工作区组件。

   - 以下是具有 以下组件的 devfile 示例 ：

     例 1.1. 工具

     devfile **的工具** 组件定义了以下请求和限值：

     ```
     memoryLimit: 6G
     memoryRequest: 512M
     cpuRequest: 1000m
     cpuLimit: 4000m
     ```

     例 1.2. postgresql

     postgresql 组件没有定义任何请求和限值，因此回退到专用容器的默认值：

     ```
     memoryLimit: 128M
     memoryRequest: 64M
     cpuRequest: 10m
     cpuLimit: 1000m
     ```

   - 在工作区启动过程中，使用以下请求和限制隐式置备内部 **che-gateway** 容器：

     ```
     memoryLimit: 256M
     memoryRequest: 64M
     cpuRequest: 50m
     cpuLimit: 500m
     ```

2. 计算每个工作区所需的资源总和。如果要使用多个 devfile，请为每个预期的 devfile 重复这个计算。

   例 1.3. 上一步中 示例 devfile 的工作区要求

用途	Pod	容器名称	内存限制	内存请求	CPU 限制	CPU 请求
开发人员工具	**workspace**	**工具**	6 GiB	512 MiB	4000 M	1000 M
数据库	**workspace**	**postgresql**	128 MiB	64 MiB	1000 M	10 M
OpenShift Dev Spaces 网关	**workspace**	**che-gateway**	256 MiB	64 MiB	500 M	50 M
		总计	**6.4 GiB**	**640 MiB**	**5500 m**	**1060 m**

3. 将按预期所有用户同时运行的工作区数量乘以每个工作区计算的资源。
4. 计算 OpenShift Dev Spaces Operator、Operands 和 Dev Workspace Controller 的要求总和。

表 1.1. OpenShift Dev Spaces Operator、Operpers 和 Dev Workspace Controller 的默认要求

用途	Pod 名称	容器名称	内存限制	内存请求	CPU 限制	CPU 请求
OpenShift Dev Spaces operator	**devspaces-operator**	**devspaces-operator**	256 MiB	64 MiB	500 M	100 M
OpenShift Dev Spaces Server	**devspaces**	**devspaces-server**	1 GiB	512 MiB	1000 M	100 M
OpenShift Dev Spaces Dashboard	**devspaces-dashboard**	**devspaces-dashboard**	256 MiB	32 MiB	500 M	100 M
OpenShift Dev Spaces 网关	**devspaces-gateway**	**Traefik**	4 GiB	128 MiB	1000 M	100 M
OpenShift Dev Spaces 网关	**devspaces-gateway**	**configbumap**	256 MiB	64 MiB	500 M	50 M
OpenShift Dev Spaces 网关	**devspaces-gateway**	**oauth-proxy**	512 MiB	64 MiB	500 M	100 M
OpenShift Dev Spaces 网关	**devspaces-gateway**	**kube-rbac-proxy**	512 MiB	64 MiB	500 M	100 M
devfile registry	**devfile-registry**	**devfile-registry**	256 MiB	32 MiB	500 M	100 M
插件 registry	**plugin-registry**	**plugin-registry**	256 MiB	32 MiB	500 M	100 M

用途	Pod 名称	容器名称	内存限制	内存请求	CPU 限制	CPU 请求
dev Workspace Controller Manager	**devworkspace-controller-manager**	**devworkspace-controller**	1 GiB	100 MiB	1000 M	250 M
dev Workspace Controller Manager	**devworkspace-controller-manager**	**kube-rbac-proxy**	不适用	不适用	不适用	不适用
dev Workspace webhook 服务器	**devworkspace-webhook-server**	**webhook-server**	300 MiB	20 MiB	200 M	100 M
dev Workspace Operator Catalog	**devworkspace-operator-catalog**	**registry-server**	不适用	50 MiB	不适用	10 M
Dev Workspace Webhook Server	**devworkspace-webhook-server**	**webhook-server**	300 MiB	20 MiB	200 M	100 M
Dev Workspace Webhook Server	**devworkspace-webhook-server**	**kube-rbac-proxy**	不适用	不适用	不适用	不适用
		总计	**9 GiB**	**1.2 GiB**	**6.9**	**1.3**

其他资源

- 什么是 devfile
- devfile 的优点
- devfile 自定义概述

第 2 章 安装 DEV SPACES

本节包含安装 Red Hat OpenShift Dev Spaces 的说明。

每个集群只能部署一个 OpenShift Dev Spaces 实例。

- 第 2.1.2 节 “使用 CLI 在 OpenShift 上安装 Dev Spaces”
- 第 2.1.3 节 “使用 Web 控制台在 OpenShift 上安装 Dev Spaces”
- 第 2.1.4 节 “在受限环境中安装 Dev Spaces”

2.1. 在云中安装 DEV SPACES

在云中部署并运行 Red Hat OpenShift Dev Spaces。

先决条件

- 要在其上部署 OpenShift Dev Spaces 的 OpenShift 集群。
- **DSC**: Red Hat OpenShift Dev Spaces 的命令行工具。请参阅： 第 1.2 节 “安装 dsc 管理工具”。

2.1.1. 在云中部署 OpenShift Dev Spaces

按照下面的说明，使用 **dsc** 工具在云中启动 OpenShift Dev Spaces 服务器。

- 第 2.1.2 节 “使用 CLI 在 OpenShift 上安装 Dev Spaces”
- 第 2.1.3 节 “使用 Web 控制台在 OpenShift 上安装 Dev Spaces”
- 第 2.1.4 节 “在受限环境中安装 Dev Spaces”
- https://access.redhat.com/documentation/zh-cn/red_hat_openshift_dev_spaces/3.14/html-single/user_guide/index#installing-che-on-microsoft-azure

2.1.2. 使用 CLI 在 OpenShift 上安装 Dev Spaces

您可以在 OpenShift 上安装 OpenShift Dev Spaces。

先决条件

- OpenShift Container Platform
- 具有 OpenShift 集群的管理权限的活跃 **oc** 会话。请参阅 OpenShift CLI 入门。
- **DSC**.请参阅： 第 1.2 节 “安装 dsc 管理工具”。

流程

1. 可选： 如果您之前在这个 OpenShift 集群上部署 OpenShift Dev Spaces，请确保删除了以前的 OpenShift Dev Spaces 实例：

```
$ dsc server:delete
```

2. 创建 OpenShift Dev Spaces 实例：

```
$ dsc server:deploy --platform openshift
```

验证步骤

1. 验证 OpenShift Dev Spaces 实例状态：

```
$ dsc server:status
```

2. 进入 OpenShift Dev Spaces 集群实例：

```
$ dsc dashboard:open
```

2.1.3. 使用 Web 控制台在 OpenShift 上安装 Dev Spaces

如果您 在命令行上安装 OpenShift Dev Spaces 时遇到问题，您可以通过 OpenShift Web 控制台安装它。

先决条件

- 集群管理员的 OpenShift Web 控制台会话。请参阅 访问 Web 控制台。
- 具有 OpenShift 集群的管理权限的活跃 **oc** 会话。请参阅 OpenShift CLI 入门。
- 对于在同一 OpenShift 集群上重复安装：您根据 第 8 章 *卸载 Dev Spaces* 卸载以前的 OpenShift Dev Spaces 实例。

流程

1. 在 OpenShift Web 控制台的 **Administrator** 视图中，进入 **Operators → OperatorHub** 并搜索 **Red Hat OpenShift Dev Spaces**。
2. 安装 Red Hat OpenShift Dev Spaces Operator。

提示

请参阅使用 Web 控制台从 OperatorHub 安装。

小心

Red Hat OpenShift Dev Spaces Operator 依赖于 Dev Workspace Operator。如果手动将 Red Hat OpenShift Dev Spaces Operator 安装到非默认命名空间中，请确保也在同一命名空间中安装 Dev Workspace Operator。这是必要的，因为 Operator Lifecycle Manager 将尝试将 Dev Workspace Operator 安装为 Red Hat OpenShift Dev Spaces Operator 命名空间中的依赖项，如果后者安装在其他命名空间中，可能会导致 Dev Workspace Operator 的两个冲突安装。

小心

如果要在集群上加入 Web Terminal Operator，请确保使用与 Red Hat OpenShift Dev Spaces Operator 相同的安装命名空间，因为这两个 Operator 都依赖于 Dev Workspace Operator。Web Terminal Operator、Red Hat OpenShift Dev Spaces Operator 和 Dev Workspace Operator 必须在同一个命名空间中安装。

1. 在 OpenShift 中创建 **openshift-devspaces** 项目，如下所示：

```
oc create namespace openshift-devspaces
```

2. 进入 **Operators → Installed Operators → Red Hat OpenShift Dev Spaces instance Specification → Create CheCluster → YAML view**。
3. 在 **YAML 视图中**，将 **namespace: openshift-operators** 替换为 **namespace: openshift-devspaces**。
4. 选择 **Create**。

提示

请参阅 从已安装的 Operator 创建应用程序。

验证

1. 在 **Red Hat OpenShift Dev Spaces 实例规格**中，进入 **devspaces**，登录 **Details** 选项卡。
1. 在 **Message** 下，检查是否有 **None**，这意味着没有错误。
2. 在 **Red Hat OpenShift Dev Spaces URL**下，等待 OpenShift Dev Spaces 实例的 URL 出现，然后打开 URL 来检查 OpenShift Dev Spaces 仪表板。
3. 在 **Resources** 选项卡中，查看 OpenShift Dev Spaces 部署及其状态的资源。

2.1.4. 在受限环境中安装 Dev Spaces

在受限网络中运行的 OpenShift 集群中，公共资源不可用。

但是，部署 OpenShift Dev Spaces 并运行工作区需要以下公共资源：

- Operator 目录
- 容器镜像
- 项目示例

要使这些资源可用，您可以在 OpenShift 集群可访问的注册表中将其副本替换为其副本。

先决条件

- OpenShift 集群至少有 64 GB 磁盘空间。
- OpenShift 集群已准备好在受限网络上运行，OpenShift control plane 可以访问互联网。请参阅关于断开连接的安装镜像，以及 在受限网络中使用 Operator Lifecycle Manager。
- 具有 OpenShift 集群的管理权限的活跃 **oc** 会话。请参阅 OpenShift CLI 入门。
- 一个到 **registry.redhat.io** 红帽生态系统目录的活跃的 **oc registry** 会话。请参阅： Red Hat Container Registry 身份验证。
- **opm**.请参阅安装 **opm** CLI。

- **jq**.请参阅 下载 **jq**。
- **Podman**.请参阅 Podman 安装说明。
- **Skopeo** 版本 1.6 或更高版本。请参阅 安装 Skopeo。
- 一个活跃的 **skopeo** 会话，用于管理对私有 Docker 注册表的访问。向 registry 进行身份验证，并为断开连接的安装 mirror 镜像。
- **用于** OpenShift Dev Spaces 版本 3.14 的 DSC。请参阅 第 1.2 节 “安装 dsc 管理工具”。

流程

1. 下载并执行镜像脚本，以安装自定义 Operator 目录并镜像相关的镜像： prepare-restricted-environment.sh。

```
$ bash prepare-restricted-environment.sh \
  --devworkspace_operator_index registry.redhat.io/redhat/redhat-operator-index:v4.16\
  --devworkspace_operator_version "v0.28.0" \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.16" \
  --prod_operator_package_name "devspaces" \
  --prod_operator_bundle_name "devspacesoperator" \
  --prod_operator_version "v3.14.0" \
  --my_registry "<my_registry>" (1)
```

(1) 镜像将要镜像的私有 Docker registry

2. 使用上一步中 **che-operator-cr-patch.yaml** 中设置的配置安装 OpenShift Dev Spaces：

```
$ dsc server:deploy \
  --platform=openshift \
  --olm-channel stable \
  --catalog-source-name=devspaces-disconnected-install \
  --catalog-source-namespace=openshift-marketplace \
  --skip-devworkspace-operator \
  --che-operator-cr-patch-yaml=che-operator-cr-patch.yaml
```

3. 允许从 OpenShift Dev Spaces 命名空间到用户项目中的所有 Pod 的传入流量。请参阅：第 3.7.1 节 “配置网络策略”。

其他资源

- 红帽提供的 Operator 目录
- 管理自定义目录

2.1.4.1. 设置 Ansible 示例

按照以下步骤在受限环境中使用 Ansible 示例。

先决条件

- Microsoft Visual Studio Code - 开源 IDE

- 64 位 x86 系统。

流程

1. 镜像以下镜像：

```
ghcr.io/ansible/ansible-workspace-env-reference@sha256:03d7f0fe6caaae62ff2266906b63d67ebd9cf6e4a056c7c0a0c1320e6cfbebce
registry.access.redhat.com/ubi8/python-39@sha256:301fec66443f80c3cc507ccaf72319052db5a1dc56deb55c8f169011d4bbaacb
```

2. 配置集群代理以允许访问以下域：

```
.ansible.com
.ansible-galaxy-ng.s3.dualstack.us-east-1.amazonaws.com
```

注意

计划在以后的发行版本中对以下 IDE 和 CPU 架构的支持：

- IDE
  - JetBrains IntelliJ IDEA 社区版 IDE (技术预览)
- CPU 架构
  - IBM Power (ppc64le)
  - IBM Z (s390x)

2.2. 查找完全限定域名(FQDN)

您可以在命令行中或 OpenShift web 控制台中获取机构 OpenShift Dev Spaces 实例的完全限定域名 (FQDN)。

提示

您可以在 OpenShift Web 控制台的 **Administrator** 视图中找到机构的 OpenShift Dev Spaces 实例的 FQDN，如下所示：进入 **Operators → Installed Operators → Red Hat OpenShift Dev Spaces instance Specification → devspaces → Red Hat OpenShift Dev Spaces URL**。

先决条件

- 具有 OpenShift 集群的管理权限的活跃 **oc** 会话。请参阅 OpenShift CLI 入门。

流程

1. 运行以下命令:

```
oc get checluster devspaces -n openshift-devspaces -o jsonpath='{.status.cheURL}'
```

第 3 章 配置 DEV SPACES

本节论述了 Red Hat OpenShift Dev Spaces 的配置方法和选项。

3.1. 了解 CHECLUSTER 自定义资源

OpenShift Dev Spaces 的默认部署由 Red Hat OpenShift Dev Spaces Operator 的 **CheCluster** 自定义资源参数组成。

CheCluster 自定义资源是一个 Kubernetes 对象。您可以通过编辑 **CheCluster** 自定义资源 YAML 文件来配置它。此文件包含用于配置每个组件的部分：
devWorkspace、**cheServer**、**pluginRegistry**、**devfileRegistry**、仪表板和 **imagePuller**。

Red Hat OpenShift Dev Spaces Operator 将 **CheCluster** 自定义资源转换为 OpenShift Dev Spaces 安装的每个组件可用的配置映射。

OpenShift 平台将配置应用到每个组件，并创建必要的 Pod。当 OpenShift 检测到一个组件的配置中有更改时，它会相应地重启 Pod。

> **例 3.1. 配置 OpenShift Dev Spaces 服务器组件的主要属性**
>
> 1. 在 **cheServer** 组件部分中应用带有适当修改的 **CheCluster** 自定义资源 YAML 文件。
> 2. Operator 生成 **che ConfigMap**。
> 3. OpenShift 会检测 **ConfigMap** 中的更改，并触发 OpenShift Dev Spaces Pod 重启。

其他资源

- 了解 Operator
- "了解自定义资源"

3.1.1. 在安装过程中使用 dsc 配置 CheCluster 自定义资源

要使用适当的配置部署 OpenShift Dev Spaces，请在安装 OpenShift Dev Spaces 时编辑 **CheCluster** 自定义资源 YAML 文件。否则，OpenShift Dev Spaces 部署使用 Operator 的默认配置参数。

先决条件

- 具有 OpenShift 集群的管理权限的活跃 **oc** 会话。请参阅 CLI 入门。
- **DSC**.请参阅： 第 1.2 节 “安装 dsc 管理工具”。

流程

- 创建一个 **che-operator-cr-patch.yaml** YAML 文件，其中包含要配置的 **CheCluster** 自定义资源的子集：

```
spec:
  <component>:
    <property_to_configure>: <value>
```

- 部署 OpenShift Dev Spaces 并应用 **che-operator-cr-patch.yaml** 文件中描述的更改：

```
$ dsc server:deploy \
--che-operator-cr-patch-yaml=che-operator-cr-patch.yaml \
--platform <chosen_platform>
```

验证

1. 验证配置的属性的值：

```
$ oc get configmap che -o jsonpath='{.data.<configured_property>}' \
-n openshift-devspaces
```

其他资源

- 第 3.1.3 节 “**CheCluster** 自定义资源字段参考”.
- 第 3.3.2 节 “Dev Spaces 服务器的高级配置选项”.

3.1.2. 使用 CLI 配置 CheCluster 自定义资源

要配置正在运行的 OpenShift Dev Spaces 实例，请编辑 **CheCluster** 自定义资源 YAML 文件。

先决条件

- OpenShift 上的 OpenShift Dev Spaces 实例。
- 具有对目标 OpenShift 集群的管理权限的活跃 **oc** 会话。请参阅 CLI 入门。

流程

1. 编辑集群中的 CheCluster 自定义资源：

```
$ oc edit checluster/devspaces -n openshift-devspaces
```

2. 保存并关闭该文件以应用更改。

验证

1. 验证配置的属性的值：

```
$ oc get configmap che -o jsonpath='{.data.<configured_property>}' \
-n openshift-devspaces
```

其他资源

- 第 3.1.3 节 “**CheCluster** 自定义资源字段参考”.
- 第 3.3.2 节 “Dev Spaces 服务器的高级配置选项”.

3.1.3. CheCluster 自定义资源字段参考

本节论述了可用于自定义 **CheCluster** 自定义资源的所有字段。

例 3.2. 最小 CheCluster 自定义资源示例。

```
apiVersion: org.eclipse.che/v2
kind: CheCluster
metadata:
  name: devspaces
  namespace: openshift-devspaces
spec:
  components: {}
  devEnvironments: {}
  networking: {}
```

表 3.1. 开发环境配置选项。

属性	描述	default
containerBuildConfiguration	容器构建配置。	
defaultComponents	应用到 DevWorkspace 的默认组件。这些默认组件旨在在 Devfile 中不包含任何组件时使用。	
defaultEditor	要使用以下内容创建工作区创建的默认编辑器。它可以是插件 ID 或 URI。插件 ID 必须具有 **publisher/plugin/version** 格式。URI 必须从 **http://** 或 **https://** 开始。	
defaultNamespace	用户的默认命名空间。	{ "autoProvision": true, "template": "<username>-che"}
defaultPlugins	应用到 DevWorkspace 的默认插件。	
deploymentStrategy	DeploymentStrategy 定义用于将现有工作区 pod 替换为新工作区的部署策略。可用的部署是 **Recreate** 和 **RollingUpdate**。使用 **Recreate** 部署策略时，现有工作区 pod 会在创建新工作区 pod 前被终止。使用 **RollingUpdate** 部署策略时，会创建一个新的工作区 pod，只有在新的工作区 Pod 处于 ready 状态时，现有的工作区 pod 才会被删除。如果没有指定，则使用默认的 **Recreate** 部署策略。	
disableContainerBuildCapabilities	禁用容器构建功能。当设置为 **false** 时（默认值），会忽略 devEnvironments.security.containerSecurityContext 字段，并且应用以下容器 SecurityContext: \n containerSecurityContext: allowPrivilegeEscalation: true capabilities: add: - SETGID - SETUID	
gatewayContainer	GatewayContainer 配置.	
imagePullPolicy	imagePullPolicy 定义 DevWorkspace 中用于容器的 imagePullPolicy。	
maxNumberOfRunningWorkspacesPerUser	每个用户运行工作区的最大数量。值 -1 允许用户运行无限数量的工作区。	
maxNumberOfWorkspacesPerUser	已停止和运行的工作空间总数，用户可以保留该工作区。值 -1 允许用户保留无限数量的工作区。	-1
nodeSelector	节点选择器限制可以运行工作区 pod 的节点。	

属性	描述	default
persistUserHome	PersistUserHome 定义在工作区中保留用户主目录的配置选项。	
podSchedulerName	工作区 pod 的 Pod 调度程序。如果没有指定，pod 调度程序被设置为集群中的默认调度程序。	
projectCloneContainer	项目克隆容器配置。	
secondsOfInactivityBeforeIdling	工作区的空闲超时（以秒为单位）。此超时是没有活动时空闲工作区之后的持续时间。要禁用因为不活跃而空闲的工作区闲置，请将此值设置为 -1。	1800
secondsOfRunBeforeIdling	为工作区运行超时（以秒为单位）。此超时是工作区运行的最长持续时间。要禁用工作区运行超时，请将此值设置为 -1。	-1
安全	工作区安全配置。	
serviceAccount	在启动工作区时，被 DevWorkspace operator 使用的 ServiceAccount。	
serviceAccountTokens	将作为投射卷挂载到工作区 pod 的 ServiceAccount 令牌列表。	
startTimeoutSeconds	StartTimeoutSeconds 确定工作区在自动失败前可以启动的最长时间（以秒为单位）。如果没有指定，则使用默认值 300 秒(5 分钟)。	300
storage	工作区持久性存储。	{ "pvcStrategy": "per-user"}
容限（tolerations）	工作区 pod 的 pod 容限限制了工作区 pod 可以运行的位置。	
trustedCerts	可信证书设置。	
user	用户配置。	

表 3.2. defaultNamespace 选项。

属性	描述	default
autoProvision	指明是否允许自动创建用户命名空间。如果设置为 false，则必须由集群管理员预先创建用户命名空间。	true

属性	描述	default
模板	如果没有提前创建用户命名空间，此字段定义在启动第一个工作区时创建的 Kubernetes 命名空间。您可以使用 < **username>** 和 &**lt;userid&** gt; 占位符，如 che-workspace-<username>。	"<username>-che"

表 3.3. defaultPlugins 选项。

属性	描述	default
editor	用于为其指定默认插件的编辑器 ID。	
plugins	指定编辑器的默认插件 URI。	

表 3.4. gatewayContainer 选项。

属性	描述	default
env	要在容器中设置的环境变量列表。	
image	容器镜像。省略它或将其留空，以使用 Operator 提供的默认容器镜像。	
imagePullPolicy	镜像拉取策略。默认值为 **Always 每天、下一个** 或 **最新的镜像**，其他情况下为 **IfNotPresent**。	
name	容器名称。	
resources	此容器所需的计算资源。	

表 3.5. 存储选项。

属性	描述	default
perUserStrategyPvcConfig	使用 **per-user** PVC 策略时的 PVC 设置。	
perWorkspaceStrategyPvcConfig	使用 **per-workspace** PVC 策略时的 PVC 设置。	
pvcStrategy	OpenShift Dev Spaces 服务器的持久性卷声明策略。支持的策略有：**per-user**（一个卷中的所有工作区 PVC）、**per-workspace**（每个工作区都有自己的独立 PVC）和 **ephemeral**（非持久性存储，当工作区停止时本地的变化会丢失。）	"per-user"

表 3.6. 按用户的 PVC 策略选项。

属性	描述	default
claimSize	持久性卷声明大小。要更新声明大小，置备它的存储类必须支持调整大小。	
storageClass	持久性卷声明的存储类。当忽略或留空时，会使用默认存储类。	

表 3.7. per-workspace PVC 策略选项。

属性	描述	default
claimSize	持久性卷声明大小。要更新声明大小，置备它的存储类必须支持调整大小。	
storageClass	持久性卷声明的存储类。当忽略或留空时，会使用默认存储类。	

表 3.8. trustedCerts 选项。

属性	描述	default
gitTrustedCertsConfigMapName	ConfigMap 包含要传播到 OpenShift Dev Spaces 组件并为 Git 提供特定配置的证书。请参见以下页面：https://www.eclipse.org/che/docs/stable/administration-guide/deploying-che-with-support-for-git-repositories-with-self-signed-certificates/ ConfigMap 必须具有 **app.kubernetes.io/part-of=che.eclipse.org** 标签。	

表 3.9. containerBuildConfiguration 选项。

属性	描述	default
openShiftSecurityContextConstraint	构建容器的 OpenShift 安全性上下文约束。	"container-build"

表 3.10. OpenShift Dev Spaces 组件配置。

属性	描述	default
cheServer	与 OpenShift Dev Spaces 服务器相关的常规配置设置。	{ "debug": false, "logLevel": "INFO"}
dashboard	与 OpenShift Dev Spaces 安装使用的仪表板相关的配置设置。	

属性	描述	default
devWorkspace	DevWorkspace Operator 配置。	
devfileRegistry	与 OpenShift Dev Spaces 安装使用的 devfile registry 相关的配置设置。	
imagePuller	Kubernetes Image Puller 配置。	
metrics	OpenShift Dev Spaces 服务器指标配置。	{ "enable": true}
pluginRegistry	与 OpenShift Dev Spaces 安装使用的插件 registry 相关的配置设置。	

表 3.11. 与 OpenShift Dev Spaces 服务器组件相关的常规配置设置。

属性	描述	default
clusterRoles	分配给 OpenShift Dev Spaces ServiceAccount 的额外 ClusterRole。每个角色都必须有一个 **app.kubernetes.io/part-of=che.eclipse.org** 标签。默认角色为：- < **devspaces-namespace>-cheworkspaces-clusterrole** - **<devspaces-namespace>-cheworkspaces-namespaces-clusterrole - <devspaces-namespace>-cheworkspaces-devworkspace-clusterrole**, 其中 <devspaces-namespace> 是创建 CheCluster CR 的命名空间。OpenShift Dev Spaces Operator 必须已在这些 ClusterRole 中拥有所有权限才能授予它们。	
debug	为 OpenShift Dev Spaces 服务器启用 debug 模式。	false
部署	部署覆盖选项。	
extraProperties	除了从 **CheCluster** 自定义资源(CR)的其他字段外，OpenShift Dev Spaces 服务器还会使用生成的 **che** ConfigMap 中使用的额外环境变量映射。如果 **extraProperties** 字段包含从其他 CR 字段的 **che** ConfigMap 中生成的属性，则改为使用 **extraProperties** 中定义的值。	
logLevel	OpenShift Dev Spaces 服务器的日志级别： **INFO** 或 **DEBUG**。	"INFO"
proxy	Kubernetes 集群的代理服务器设置。OpenShift 集群不需要额外的配置。通过为 OpenShift 集群指定这些设置，您可以覆盖 OpenShift 代理配置。	

表 3.12. 代理 选项。

属性	描述	default
credentialsSecretName	包含代理服务器 **的用户名和密码** 的 secret 名称。secret 必须具有 **app.kubernetes.io/part-of=che.eclipse.org** 标签。	
nonProxyHosts	可直接到达的主机列表，绕过代理。指定通配符域，使用以下形式 **.<DOMAIN&** gt;，例如： - localhost - my.host.com - 123.42.12.32 仅在需要代理配置时才使用。Operator 尊重 OpenShift 集群范围的代理配置，在自定义资源中定义 **nonProxyHosts** 会导致从集群代理配置中合并非proxy 主机列表，以及自定义资源中定义的名称。请参见以下页面 ： https://docs.openshift.com/container-platform/latest/networking/enable-cluster-wide-proxy.html。	
port	代理服务器端口。	
url	代理服务器的 URL （协议+主机名）。仅在需要代理配置时才使用。Operator 尊重 OpenShift 集群范围的代理配置，在自定义资源中定义 **url** 会导致覆盖集群代理配置。请参见以下页面 ： https://docs.openshift.com/container-platform/latest/networking/enable-cluster-wide-proxy.html。	

表 3.13. 与 OpenShift Dev Spaces 安装使用的插件 registry 组件相关的配置设置。

属性	描述	default
部署	部署覆盖选项。	
disableInternalRegistry	禁用内部插件 registry。	
externalPluginRegistries	外部插件 registry。	
openVSXURL	打开 VSX 注册表 URL。如果省略了嵌入式实例，则将使用。	

表 3.14. externalPluginRegistries 选项。

属性	描述	default
url	插件 registry 的公共 URL。	

表 3.15. 与 OpenShift Dev Spaces 安装使用的 Devfile registry 组件相关的配置设置。

属性	描述	default
部署	部署覆盖选项。	
disableInternalRegistry	禁用内部 devfile registry。	
externalDevfileRegistries	外部 devfile registry 提供示例 ready-to-use devfile。	

表 3.16. ExternalDevfileRegistries 选项。

属性	描述	default
url	devfile registry 的公共 UR 提供示例 ready-to-use devfile。	

表 3.17. 与 OpenShift Dev Spaces 安装使用的 Dashboard 组件相关的配置设置。

属性	描述	default
品牌	仪表板品牌资源.	
部署	部署覆盖选项。	
headerMessage	仪表板标头消息。	
logLevel	仪表板的日志级别。	"ERROR"

表 3.18. headerMessage 选项。

属性	描述	default
显示	指示仪表板显示消息。	
text	用户仪表板上显示的警告消息。	

表 3.19. Kubernetes Image Puller 组件配置。

属性	描述	default

属性	描述	default
enable	安装和配置社区支持的 Kubernetes Image Puller Operator。当您在不提供任何 spec 的情况下将值设为 **true** 时，它会创建一个由 Operator 管理的默认 Kubernetes Image Puller 对象。当您将值设为 **false** 时，Kubernetes Image Puller 对象会被删除，Operator 会卸载，无论是否提供了 spec。如果将 **spec.images** 字段留空，则会自动检测到一组推荐的与工作区相关的镜像，并在安装后预先拉取。请注意，虽然此 Operator 及其行为受社区支持，但其有效负载可能被商业支持用于拉取商业支持的镜像。	
spec	一个 Kubernetes Image Puller spec，用于在 CheCluster 中配置镜像 puller。	

表 3.20. OpenShift Dev Spaces 服务器指标组件配置。

属性	描述	default
enable	为 OpenShift Dev Spaces 服务器端点启用 **指标**。	true

表 3.21. 允许用户使用远程 Git 存储库的配置设置。

属性	描述	default
azure	允许用户使用托管在 Azure DevOps Service (dev.azure.com)上的存储库。	
Bitbucket	允许用户使用 Bitbucket 上托管的存储库 (bitbucket.org 或 self-hosted)。	
github	允许用户使用托管在 GitHub 上的软件仓库 (github.com 或 GitHub Enterprise)。	
gitlab	允许用户使用 GitLab 上托管的存储库(gitlab.com 或自托管)。	

表 3.22. GitHub 选项。

属性	描述	default
disableSubdomainIsolation	禁用子域隔离。弃用了 **che.eclipse.org/scm-github-disable-subdomain-isolation** 注解。详情请查看以下页面：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-2-for-github/。	

属性	描述	default
端点	GitHub 服务器端点 URL。弃用了 **che.eclipse.org/scm-server-endpoint** 注解。详情请查看以下页面：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-2-for-github/。	
secretName	Kubernetes secret，其中包含 Base64 编码的 GitHub OAuth 客户端 ID 和 GitHub OAuth 客户端 secret。详情请查看以下页面：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-2-for-github/。	

表 3.23. GitLab 选项。

属性	描述	default
端点	GitLab 服务器端点 URL。弃用了 **che.eclipse.org/scm-server-endpoint** 注解。请参见以下页面 ：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-2-for-gitlab/。	
secretName	Kubernetes secret，其中包含 Base64 编码的 GitHub 应用程序 id 和 GitLab Application Client secret。请参见以下页面 ：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-2-for-gitlab/。	

表 3.24. Bitbucket 选项。

属性	描述	default
端点	Bitbucket 服务器端点 URL。弃用了 **che.eclipse.org/scm-server-endpoint** 注解。请参见以下页面 ：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-1-for-a-bitbucket-server/。	
secretName	Kubernetes secret，包含 Base64 编码的 Bitbucket OAuth 1.0 或 OAuth 2.0 数据。详情请查看以下页面：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-1-for-a-bitbucket-server/ 和 https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-2-for-the-bitbucket-cloud/。	

表 3.25. Azure 选项。

属性	描述	default
secretName	Kubernetes secret，其中包含 Base64 编码的 Azure DevOps Service Application ID 和 Client Secret。请参见以下页面：https://www.eclipse.org/che/docs/stable/administration-guide/configuring-oauth-2-for-microsoft-azure-devops-services	

表 3.26. 网络、OpenShift Dev Spaces 身份验证和 TLS 配置。

属性	描述	default
annotations	定义将为 Ingress 设置的注解(OpenShift 平台的路由)。kubernetes 平台的默认值为：kubernetes.io/ingress.class: "nginx" nginx.ingress.kubernetes.io/proxy-read-timeout: "3600", nginx.ingress.kubernetes.io/proxy-connect-timeout: "3600", nginx.ingress.kubernetes.io/ssl-redirect: "true"	
auth	身份验证设置.	{ "gateway": { "configLabels": { "app": "che", "component": "che-gateway-config" } }}
domain	对于 OpenShift 集群，Operator 使用域为路由生成主机名。生成的主机名遵循这个模式：che-<devspaces-namespace>.<domain>。<devspaces-namespace> 是创建 CheCluster CRD 的命名空间。与标签结合使用，它会创建一个由非默认 Ingress 控制器提供服务的路由。对于 Kubernetes 集群，它包含一个全局入口域。没有默认值：您必须指定它们。	
hostname	安装的 OpenShift Dev Spaces 服务器的公共主机名。	
ingressClassName	ingressClassName 是 IngressClass 集群资源的名称。如果在 **IngressClassName** 字段和 **kubernetes.io/ingress.class** 注解中都定义了类名称，**IngressClassName** 字段将具有优先权。	
labels	定义为 Ingress 设置的标签(OpenShift 平台的路由)。	
tlsSecretName	用于设置 Ingress TLS 终止的 secret 名称。如果字段是空字符串，则使用默认集群证书。secret 必须具有 **app.kubernetes.io/part-of=che.eclipse.org** 标签。	

表 3.27. 身份验证选项。

属性	描述	default
advancedAuthorization	提前授权设置。决定允许哪些用户和组访问 Che。如果 he/she 位于 **allowUsers** 列表中，或者不是来自 **allowGroups** 列表中的组的成员，则允许用户访问 OpenShift Dev Spaces，而不是 **denyUsers** 列表，也不是 **denyGroups** 列表的成员。如果 **allowUsers** 和 **allowGroups** 为空，则允许所有用户访问 Che。如果 **denyUsers** 和 **denyGroups** 为空，则不会拒绝用户访问 Che。	
gateway	网关设置。	{ "configLabels": { "app": "che", "component": "che-gateway-config" }}
identityProviderURL	身份提供程序服务器的公共 URL。	
identityToken	要传递给上游的身份令牌。支持两种类型的令牌：**id_token** 和 **access_token**。默认值为 **id_token**。此字段特定于只为 Kubernetes 进行的 OpenShift Dev Spaces 安装，并忽略 OpenShift。	
oAuthAccessTokenInactivityTimeoutSeconds	在 OpenShift **OAuthClient** 资源中设置令牌的不活跃超时，用于在 OpenShift 端设置身份联邦。0 表示此客户端的令牌永远不会超时。	
oAuthAccessTokenMaxAgeSeconds	在 OpenShift **OAuthClient** 资源中设置令牌的访问令牌最大期限，用于在 OpenShift 端设置身份联邦。0 表示没有过期。	
oAuthClientName	用于在 OpenShift 端设置身份联邦的 OpenShift **OAuthClient** 资源的名称。	
oAuthScope	访问令牌范围.此字段特定于只为 Kubernetes 进行的 OpenShift Dev Spaces 安装，并忽略 OpenShift。	
oAuthSecret	OpenShift **OAuthClient** 资源中设置的 secret 名称，用于在 OpenShift 端设置身份联邦。对于 Kubernetes，可以是纯文本 oAuthSecret 值，也可以是包含键 **oAuthSecret** 的 kubernetes secret 的名称，值为 secret。注意：此 secret 必须与 **CheCluster** 资源位于同一个命名空间中，并包含标签 **app.kubernetes.io/part-of=che.eclipse.org**。	

表 3.28. 网关 选项。

属性	描述	default
configLabels	网关配置标签。	{ "app": "che", "component": "che-gateway-config"}
部署	部署覆盖选项。由于网关部署由多个容器组成，它们必须按名称在配置中区分： - **gateway** - **configbump** - **oauth-proxy** - **kube-rbac-proxy**	
kubeRbacProxy	在 OpenShift Dev Spaces 网关 pod 中配置 kube-rbac-proxy。	
oAuthProxy	在 OpenShift Dev Spaces 网关 pod 中配置 oauth-proxy。	
Traefik	在 OpenShift Dev Spaces 网关 pod 中的 Traefik 配置。	

表 3.29. 配置存储 OpenShift Dev Spaces 镜像的替代 registry。

属性	描述	default
hostname	要从中拉取镜像的替代容器 registry 的可选主机名或 URL。这个值会覆盖在 OpenShift Dev Spaces 部署中的所有默认容器镜像中定义的容器 registry 主机名。这在受限环境中安装 OpenShift Dev Spaces 特别有用。	
机构	要从中拉取镜像的替代 registry 的可选存储库名称。这个值会覆盖在 OpenShift Dev Spaces 部署中的所有默认容器镜像中定义的容器 registry 组织。这在受限环境中安装 OpenShift Dev Spaces 特别有用。	

表 3.30. 部署选项。

属性	描述	default
containers	属于 pod 的容器列表。	
securityContext	pod 应该运行的安全选项。	

表 3.31. 容器 选项。

属性	描述	default
env	要在容器中设置的环境变量列表。	

属性	描述	default
image	容器镜像。省略它或将其留空，以使用 Operator 提供的默认容器镜像。	
imagePullPolicy	镜像拉取策略。默认值为 **Always 每天、下一个** 或 **最新的镜像**，其他情况下为 **IfNotPresent**。	
name	容器名称。	
resources	此容器所需的计算资源。	

表 3.32. 容器 选项。

属性	描述	default
limits	描述允许的最大计算资源量。	
request	描述所需的最少计算资源。	

表 3.33. 请求 选项。

属性	描述	default
cpu	CPU，以内核为单位。(500M = .5 个内核)如果没有指定该值，则根据组件设置默认值。如果值为 **0，** 则不会为组件设置值。	
内存	内存，以字节为单位。(500Gi = 500GiB = 500 * 1024 * 1024)如果未指定值，则根据组件设置默认值。如果值为 **0，** 则不会为组件设置值。	

表 3.34. 限制 选项。

属性	描述	default
cpu	CPU，以内核为单位。(500M = .5 个内核)如果没有指定该值，则根据组件设置默认值。如果值为 **0，** 则不会为组件设置值。	
内存	内存，以字节为单位。(500Gi = 500GiB = 500 * 1024 * 1024)如果未指定值，则根据组件设置默认值。如果值为 **0，** 则不会为组件设置值。	

表 3.35. securityContext 选项。

属性	描述	default
fsGroup	适用于 pod 中所有容器的特殊补充组。默认值为 **1724**。	
runAsUser	运行容器进程的入口点的 UID。默认值为 **1724**。	

表 3.36. CheCluster 自定义资源状态定义 OpenShift Dev Spaces 安装的观察状态

属性	描述	default
chePhase	指定 OpenShift Dev Spaces 部署的当前阶段。	
cheURL	OpenShift Dev Spaces 服务器的公共 URL。	
cheVersion	当前安装了 OpenShift Dev Spaces 版本。	
devfileRegistryURL	内部 devfile registry 的公共 URL。	
gatewayPhase	指定网关部署的当前阶段。	
message	人类可读的消息，指示 OpenShift Dev Spaces 部署处于当前阶段的详细信息。	
pluginRegistryURL	内部插件 registry 的公共 URL。	
reason	简短 CamelCase 消息指示 OpenShift Dev Spaces 部署处于当前阶段的详细信息。	
workspaceBaseDomain	解析的工作区基域。这是 spec 中相同名称的显式定义属性的副本，如果它在 spec 中未定义，且在 OpenShift 上运行，则会自动解析路由的 basedomain。	

3.2. 配置项目

对于每个用户，OpenShift Dev Spaces 会隔离项目中的工作区。OpenShift Dev Spaces 通过存在标签和注解来识别用户项目。在启动工作区时，如果所需的项目不存在，OpenShift Dev Spaces 会使用模板名称创建项目。

您可以通过以下方法修改 OpenShift Dev Spaces 行为：

- 第 3.2.1 节 “配置项目名称”
- 第 3.2.2 节 “提前置备项目”

3.2.1. 配置项目名称

您可以在启动工作区时配置 OpenShift Dev Spaces 用来创建所需项目的项目名称模板。

有效的项目名称模板遵循以下约定：

- &lt ;username> 或 <userid > 占位符是必需的。
- 用户名和 ID 无法包含无效字符。如果用户名或 ID 的格式与 OpenShift 对象的命名约定不兼容，OpenShift Dev Spaces 通过将不兼容的字符替换为 - 符号来更改用户名或 ID。
- OpenShift Dev Spaces 将 <userid > 占位符评估为 14 个字符长字符串，并添加一个随机的六个字符长后缀以防止 ID 冲突。结果存储在用户首选项中以便重复使用。
- Kubernetes 将项目名称的长度限制为 63 个字符。
- OpenShift 限制的长度为 49 个字符。

流程

- 配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  components:
    devEnvironments:
      defaultNamespace:
        template: <workspace_namespace_template_>
```

例 3.3. 用户工作区项目名称模板示例

用户工作区项目名称模板	生成的项目示例
<username>-devspaces （默认）	user1-devspaces
<userid>-namespace	**cge1egvsb2nhba-namespace-ul1411**
<userid>-aka-<username>-namespace	**cgezegvsb2nhba-aka-user1-namespace-6m2w2b**

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.2.2. 提前置备项目

您可以提前置备工作区项目，而不依赖于自动置备。为每个用户重复这个过程。

流程

- 使用以下标签和注解为 *<username>* 用户创建 *<project_name>* 项目：

```
kind: Namespace
apiVersion: v1
metadata:
  name: <project_name> 1
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-namespace
  annotations:
    che.eclipse.org/username: <username>
```

1. 使用您选择的项目名称。

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.3. 配置服务器组件

- 第 3.3.1 节 “将 Secret 或 ConfigMap 挂载为文件或环境变量到 Red Hat OpenShift Dev Spaces 容器中”
- 第 3.3.2 节 “Dev Spaces 服务器的高级配置选项”
- 第 3.3.3 节 “为 Red Hat OpenShift Dev Spaces 容器配置副本数”

3.3.1. 将 Secret 或 ConfigMap 挂载为文件或环境变量到 Red Hat OpenShift Dev Spaces 容器中

Secret 是存储敏感数据的 OpenShift 对象，例如：

- 用户名
- 密码
- 身份验证令牌

以加密的形式使用。

用户可以挂载包含敏感数据的 OpenShift Secret 或包含 OpenShift Dev Spaces 管理的容器中的 ConfigMap，如下所示：

- 一个文件

- 环境变量

挂载过程使用标准的 OpenShift 挂载机制，但它需要额外的注释和标记。

3.3.1.1. 将 Secret 或 ConfigMap 挂载为文件到 OpenShift Dev Spaces 容器中

先决条件

- 一个 Red Hat OpenShift Dev Spaces 的运行实例。

流程

1. 在部署了 OpenShift Dev Spaces 的 OpenShift 项目中创建新的 OpenShift Secret 或 ConfigMap。要创建的对象标签必须与一组标签匹配：
   - app.kubernetes.io/part-of: che.eclipse.org
   - app.kubernetes.io/component: <DEPLOYMENT_NAME>-<OBJECT_KIND>
   - & lt;DEPLOYMENT_NAME > 对应于以下部署：
     - devspaces-dashboard
     - devfile-registry
     - plugin-registry
     - devspaces

     和

- <OBJECT_KIND& gt; 是：
  - secret

    或者
  - configmap

例 3.4. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
...
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

1. 配置注释值。注解必须表示，给定的对象作为文件挂载：
   - Che.eclipse.org/mount-as: 文件 - 表示对象挂载为文件。
   - Che.eclipse.org/mount-path: *<TARGET_PATH >* - 提供所需的挂载路径。

例 3.5. Example:

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: custom-data
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
...
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-data
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

OpenShift 对象可以包含多个项目，其名称必须与挂载到容器中的所需文件名匹配。

例 3.6. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <base64 encoded data content here>
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <data content here>
```

这会导致名为 ca.crt 的文件被挂载到 OpenShift Dev Spaces 容器的 /data 路径。

重要

要使 OpenShift Dev Spaces 容器的更改可见，请完全重新创建 Secret 或 ConfigMap 对象。

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.3.1.2. 将 Secret 或 ConfigMap 作为子Path 挂载到 OpenShift Dev Spaces 容器中

先决条件

- 一个 Red Hat OpenShift Dev Spaces 的运行实例。

流程

1. 在部署了 OpenShift Dev Spaces 的 OpenShift 项目中创建新的 OpenShift Secret 或 ConfigMap。要创建的对象标签必须与一组标签匹配：
   - app.kubernetes.io/part-of: che.eclipse.org
   - app.kubernetes.io/component: <DEPLOYMENT_NAME>-<OBJECT_KIND>

- & lt;DEPLOYMENT_NAME > 对应于以下部署：
  - devspaces-dashboard
  - devfile-registry
  - plugin-registry
  - devspaces

    和
- <OBJECT_KIND& gt; 是：
  - secret

    或者
  - configmap

例 3.7. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
...
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

1. 配置注释值。注解必须表示给定对象作为 subPath 挂载：

- Che.eclipse.org/mount-as: subpath - 表示对象挂载为 subPath。
- Che.eclipse.org/mount-path: *<TARGET_PATH >* - 提供所需的挂载路径。

例 3.8. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-data
  annotations:
    che.eclipse.org/mount-as: subpath
    che.eclipse.org/mount-path: /data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
...
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-data
  annotations:
    che.eclipse.org/mount-as: subpath
    che.eclipse.org/mount-path: /data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

OpenShift 对象可以包含多个项目，其名称必须与挂载到容器中的文件名匹配。

例 3.9. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
  annotations:
    che.eclipse.org/mount-as: subpath
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <base64 encoded data content here>
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
  annotations:
    che.eclipse.org/mount-as: subpath
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <data content here>
```

这会导致名为 **ca.crt** 的文件被挂载到 **OpenShift Dev Spaces** 容器的 **/data** 路径。

重要

要使 OpenShift Dev Spaces 容器的更改可见，请完全重新创建 Secret 或 ConfigMap 对象。

其他资源

- **第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”**
- **第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”**

3.3.1.3. 将 Secret 或 ConfigMap 作为环境变量挂载到 OpenShift Dev Spaces 容器中

先决条件

- 一个 Red Hat OpenShift Dev Spaces 的运行实例。

流程

1. 在部署了 OpenShift Dev Spaces 的 OpenShift 项目中创建新的 OpenShift Secret 或 ConfigMap。要创建的对象标签必须与一组标签匹配：
   - app.kubernetes.io/part-of: che.eclipse.org
   - app.kubernetes.io/component: <DEPLOYMENT_NAME>-<OBJECT_KIND>
   - & lt;DEPLOYMENT_NAME > 对应于以下部署：
     - devspaces-dashboard
     - devfile-registry
     - plugin-registry
     - devspaces

     和
   - <OBJECT_KIND& gt; 是：
     - secret

     或者

- configmap

例 3.10. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
...
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

1. 配置注释值。注解必须表示，给定的对象作为环境变量挂载：

   - Che.eclipse.org/mount-as: env - 表示对象作为环境变量挂载
   - Che.eclipse.org/env-name *: <FOO_ENV* > - 提供一个环境变量名称，这是挂载对象键值所必需的

例 3.11. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
```

```
    app.kubernetes.io/component: devspaces-secret
data:
  mykey: myvalue
```

或者

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
data:
  mykey: myvalue
```

这会生成两个环境变量：

- FOO_ENV
- myValue

被置备到一个 OpenShift Dev Spaces 容器中。

如果对象提供多个数据项，则必须为每个数据键提供环境变量名称，如下所示：

例 3.12. Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/mount-as: env
    che.eclipse.org/mykey_env-name: FOO_ENV
    che.eclipse.org/otherkey_env-name: OTHER_ENV
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
```

```
stringData:
  mykey: <data_content_here>
  otherkey: <data_content_here>
```

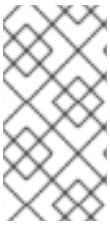
或者

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/mount-as: env
    che.eclipse.org/mykey_env-name: FOO_ENV
    che.eclipse.org/otherkey_env-name: OTHER_ENV
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
data:
  mykey: <data content here>
  otherkey: <data content here>
```

这会生成两个环境变量：

- FOO_ENV
- OTHER_ENV

被置备到一个 OpenShift Dev Spaces 容器中。



注意

OpenShift 对象中注解名称的最大长度为 63 个字符，其中 9 个字符作为前缀被保留，以 / 结尾。这可作为可用于对象的密钥的最大长度的限制。

重要

要使 OpenShift Dev Spaces 容器的更改可见，请完全重新创建 Secret 或 ConfigMap 对象。

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.3.2. Dev Spaces 服务器的高级配置选项

以下章节描述了 OpenShift Dev Spaces 服务器组件的高级部署和配置方法。

3.3.2.1. 了解 OpenShift Dev Spaces 服务器高级配置

以下章节描述了部署的 OpenShift Dev Spaces 服务器组件高级配置方法。

高级配置需要：

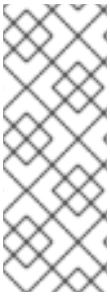
- 添加 Operator 从标准 CheCluster 自定义资源字段自动生成的环境变量。
- 从标准 CheCluster 自定义资源字段覆盖 Operator 自动生成的属性。

customCheProperties 字段是 CheCluster 自定义资源 服务器设置 的一部分，包含要应用到 OpenShift Dev Spaces 服务器组件的额外环境变量映射。

例 3.13. 覆盖工作区的默认内存限值

- 配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
apiVersion: org.eclipse.che/v2
kind: CheCluster
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LOGS_APPENDERS_IMPL: json
```



注意

OpenShift Dev Spaces Operator 的早期版本具有名为 custom 的 ConfigMap，以满足此角色。如果 OpenShift Dev Spaces Operator 找到了名为 custom 的 configMap，它会将包含的数据添加到 customCheProperties 字段中，重新部署 OpenShift Dev Spaces，并删除 自定义 configMap。

其他资源

- 第 3.1.3 节 “CheCluster 自定义资源字段参考”.

3.3.3. 为 Red Hat OpenShift Dev Spaces 容器配置副本数

要使用 Kubernetes HorizontalPodAutoscaler (HPA)为 OpenShift Dev Spaces 操作对象配置副本数，您可以为部署定义 HPA 资源。HPA 根据指定指标动态调整副本数。

流程

1. 为部署创建 HPA 资源，指定目标指标和所需的副本数。

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: scaler
  namespace: openshift-devspaces
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: <deployment_name> 1
  ...
```

1. & *lt;deployment_name* > 对应于以下部署：
   - devspaces
   - che-gateway

- devspaces-dashboard
- plugin-registry
- devfile-registry

例 3.14. 为 devspaces 部署创建一个 HorizontalPodAutoscaler ：

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: devspaces-scaler
  namespace: openshift-devspaces
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: devspaces
  minReplicas: 2
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 75
```

在本例中，HPA 将名为 devspaces 的 Deployment 为目标，最小为 2 个副本，最大 5 个副本，根据 CPU 使用率进行扩展。

其他资源

- Pod 横向自动扩展

3.4. 全局配置工作区

本节论述了管理员如何全局配置工作区。

- 第 3.4.1 节 “限制用户可以保留的工作区数量”
- 第 3.4.2 节 “允许用户同时运行多个工作区”
- 第 3.4.3 节 “带有自签名证书的 Git”
- 第 3.4.4 节 “配置工作区 nodeSelector”

3.4.1. 限制用户可以保留的工作区数量

默认情况下，用户可以在仪表板中保留无限数量的工作区，但您可以限制这个数量来减少集群的要求。

此配置是 CheCluster 自定义资源的一部分：

```
spec:
  devEnvironments:
    maxNumberOfWorkspacesPerUser: <kept_workspaces_limit> (1)
```

(1) 设置每个用户的最大工作区数。默认值 -1 允许用户保留无限数量的工作区。使用正整数设置每个用户的最大工作区数。

流程

1. 获取 OpenShift Dev Spaces 命名空间的名称。默认值为 openshift-devspaces。

```
$ oc get checluster --all-namespaces \
  -o=jsonpath="{.items[*].metadata.namespace}"
```

2. 配置 maxNumberOfWorkspacesPerUser ：

```
$ oc patch checluster/devspaces -n openshift-devspaces \ (1)
--type='merge' -p \
'{"spec":{"devEnvironments":{"maxNumberOfWorkspacesPerUser":
<kept_workspaces_limit>}}}' (2)
```

-

1

您在第 1 步中获取的 OpenShift Dev Spaces 命名空间。

2

选择 < *kept_workspaces_limit*> 值。

其他资源

- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.4.2. 允许用户同时运行多个工作区

默认情况下，一个用户一次只能运行一个工作区。您可以启用用户同时运行多个工作区。

注意

如果使用默认存储方法，如果 pod 在多节点集群中的节点间分布，则用户可能会遇到同时运行工作区时可能会出现问题。从每个用户的 通用 存储策略切换到 per-workspace 存储策略，或 使用临时存储 类型可以避免或解决这些问题。

此配置是 CheCluster 自定义资源的一部分：

```
spec:
  devEnvironments:
    maxNumberOfRunningWorkspacesPerUser: <running_workspaces_limit>
```

1

设置每个用户同时运行工作区的最大数量。-1 值允许用户运行无限数量的工作区。默认值为 1。

流程

1. 获取 OpenShift Dev Spaces 命名空间的名称。默认值为 openshift-devspaces。

```
$ oc get checluster --all-namespaces \
  -o=jsonpath="{.items[*].metadata.namespace}"
```

2. 配置 **maxNumberOfRunningWorkspacesPerUser** ：

```
$ oc patch checluster/devspaces -n openshift-devspaces \ (1)
--type='merge' -p \
'{"spec":{"devEnvironments":{"maxNumberOfRunningWorkspacesPerUser":
<running_workspaces_limit>}}}' (2)
```

(1) 您在第 1 步中获取的 OpenShift Dev Spaces 命名空间。

(2) 选择 **< *running_workspaces_limit*>** 值。

其他资源

- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.4.3. 带有自签名证书的 Git

您可以配置 OpenShift Dev Spaces，以支持使用自签名证书的 Git 供应商上的操作。

先决条件

- 具有 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 OpenShift CLI 入门。
- Git 版本 2 或更高版本

流程

1. 创建一个新的 ConfigMap，其中包含有关 Git 服务器的详情：

```
$ oc create configmap che-git-self-signed-cert \
  --from-file=ca.crt=<path_to_certificate> \ (1)
  --from-literal=githost=<git_server_url> -n openshift-devspaces (2)
```

(1)

2

指定 Git 服务器 URL 的可选参数，例如 https://git.example.com:8443。在省略时，通过 HTTPS 为所有存储库使用自签名证书。

注意

- 证书文件通常存储为 Base64 ASCII 文件，例如：.pem,.crt,.ca-bundle.保存证书文件的所有 ConfigMap 都应该使用 Base64 ASCII 证书，而不是二进制数据证书。
- 需要信任证书链。如果 ca.crt 由证书颁发机构(CA)签名，则必须将 CA 证书包含在 ca.crt 文件中。

2. 在 ConfigMap 中添加所需的标签：

```
$ oc label configmap che-git-self-signed-cert \
  app.kubernetes.io/part-of=che.eclipse.org -n openshift-devspaces
```

3. 配置 OpenShift Dev Spaces 操作对象，以将自签名证书用于 Git 存储库。请参阅第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  devEnvironments:
    trustedCerts:
      gitTrustedCertsConfigMapName: che-git-self-signed-cert
```

验证步骤

- 创建并启动新的工作区。工作区使用的每个容器都挂载一个包含自签名证书的文件的特殊卷。容器的 /etc/gitconfig 文件包含与 Git 服务器主机(its URL)和 http 部分中证书的路径相关的信息（请参阅有关 git-config的 Git 文档）。

例 3.15. /etc/gitconfig 文件的内容

```
[http "https://10.33.177.118:3000"]
sslCAInfo = /etc/config/che-git-tls-creds/certificate
```

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”
- 第 3.7.3 节 “将不受信任的 TLS 证书导入到 Dev Spaces”.

3.4.4. 配置工作区 nodeSelector

本节论述了如何为 OpenShift Dev Spaces 工作区的 Pod 配置 nodeSelector。

流程

a. 使用 NodeSelector

OpenShift Dev Spaces 使用 CheCluster 自定义资源来配置 nodeSelector ：

```
spec:
  devEnvironments:
    nodeSelector:
      key: value
```

本节必须为每个节点标签包含一组 key=value 对，才能组成 nodeSelector 规则。https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels

b. 使用 Taints 和 Tolerations

这的工作方式与 nodeSelector 相反。不指定 Pod 将要调度到哪些节点，而是指定 Pod 无法调度到哪些节点。如需更多信息，请参阅 ：https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration。

OpenShift Dev Spaces 使用 CheCluster 自定义资源来配置 容限 ：

```
spec:
  devEnvironments:
    tolerations:
      - effect: NoSchedule
```

```
    key: key
    value: value
    operator: Equal
```

重要

nodeSelector 必须在 OpenShift Dev Spaces 安装过程中配置。这可防止现有工作区因为卷关联性冲突而无法运行，因为现有工作区 PVC 和 Pod 调度到不同的区中。

为了避免 Pod 和 PVC 调度到大型的多区集群中的不同区中，请创建额外的 StorageClass 对象（请注意 allowedTopologies 字段），这将协调 PVC 创建过程。

通过 CheCluster 自定义资源将这个新创建的 StorageClass 的名称传递给 OpenShift Dev Spaces。如需更多信息，请参阅： 第 3.8.1 节 “配置存储类”。

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.4.5. 打开 VSX registry URL

要搜索和安装扩展，Microsoft Visual Studio Code - 开源编辑器使用嵌入式 Open VSX registry 实例。您还可以将 OpenShift Dev Spaces 配置为使用另一个 Open VSX registry 实例，而不是嵌入的 registry 实例。

流程

- 在 CheCluster 自定义资源 spec.components.pluginRegistry.openVSXURL 字段中设置 Open VSX registry 实例的 URL。

```
spec:
  components:
# [...]
    pluginRegistry:
      openVSXURL: <your_open_vsx_registy>
# [...]
```

其他资源

- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

- 打开 VSX registry

3.4.6. 配置用户命名空间

此流程指导您使用 OpenShift Dev Spaces 将 ConfigMap、Secret 和 PersistentVolumeClaim 从 openshift-devspaces 命名空间复制到大量特定于用户的命名空间。OpenShift Dev Spaces 会自动同步重要配置数据，如共享凭证、配置文件和证书到用户命名空间。

如果您对 openshift-devspaces 命名空间中的 Kubernetes 资源进行更改，OpenShift Dev Spaces 将立即在所有用户命名空间中复制更改。相反，如果在用户命名空间中修改了 Kubernetes 资源，OpenShift Dev Spaces 将立即恢复更改。

流程

1. 创建以下 ConfigMap 以将其复制到每个用户命名空间中。要增强可配置性，您可以通过添加额外的标签和注解来自定义 ConfigMap。有关其他可能的标签和注解，请参阅 自动挂载卷、configmap 和 secret。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: user-configmap
  namespace: openshift-devspaces
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-config
data:
  ...
```

例 3.16. 将 settings.xml 文件挂载到用户工作区：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: user-settings-xml
  namespace: openshift-devspaces
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-config
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /home/user/.m2
data:
```

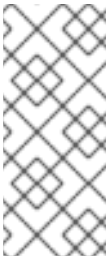
```
  settings.xml: |
    <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
https://maven.apache.org/xsd/settings-1.0.0.xsd">
      <localRepository>/home/user/.m2/repository</localRepository>
      <interactiveMode>true</interactiveMode>
      <offline>false</offline>
    </settings>
```

2. 创建以下 Secret 以将其复制到每个用户命名空间中。要增强可配置性，您可以通过添加额外的标签和注解来自定义 Secret。有关其他可能的标签和注解，请参阅 自动挂载卷、configmap 和 secret。

```
kind: Secret
apiVersion: v1
metadata:
  name: user-secret
  namespace: openshift-devspaces
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-config
data:
  ...
```

例 3.17. 将证书挂载到用户工作区：

```
kind: Secret
apiVersion: v1
metadata:
  name: user-certificates
  namespace: openshift-devspaces
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-config
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /etc/pki/ca-trust/source/anchors
stringData:
  trusted-certificates.crt: |
    ...
```



注意

在工作区启动时运行 update-ca-trust 命令以导入证书。它可以手动实现，也可以将此命令添加到 devfile 中的 postStart 事件中。请参阅在 devfile 中添加事件绑定。

例 3.18. 将环境变量挂载到用户工作区：

```
kind: Secret
apiVersion: v1
metadata:
  name: user-env
  namespace: openshift-devspaces
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-config
  annotations:
    controller.devfile.io/mount-as: env
stringData:
  ENV_VAR_1: value_1
  ENV_VAR_2: value_2
```

3. 创建以下 PersistentVolumeClaim，将其复制到每个用户命名空间中。

要增强可配置性，您可以通过添加额外的标签和注解来自定义 PersistentVolumeClaim。有关其他可能的标签和注解，请参阅 自动挂载卷、configmap 和 secret。

要修改 'PersistentVolumeClaim'，请删除它并在 openshift-devspaces 命名空间中创建一个新。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: user-pvc
  namespace: openshift-devspaces
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-config
spec:
  ...
```

例 3.19. 将 PersistentVolumeClaim 挂载到用户工作区：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: user-pvc
  namespace: openshift-devspaces
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-config
    controller.devfile.io/mount-to-devworkspace: 'true'
  annotations:
    controller.devfile.io/mount-path: /home/user/data
```

```
    controller.devfile.io/read-only: 'true'
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  volumeMode: Filesystem
```

其他资源

- https://access.redhat.com/documentation/zh-cn/red_hat_openshift_dev_spaces/3.14/html-single/user_guide/index#end-user-guide:mounting-configmaps
- https://access.redhat.com/documentation/zh-cn/red_hat_openshift_dev_spaces/3.14/html-single/user_guide/index#end-user-guide:mounting-secrets
- https://access.redhat.com/documentation/zh-cn/red_hat_openshift_dev_spaces/3.14/html-single/user_guide/index#end-user-guide:requesting-persistent-storage-for-workspaces
- 自动挂载卷、configmap 和 secret

3.5. 缓存镜像以便更快地启动工作区

要提高 OpenShift Dev Spaces 工作区的开始时间性能，请使用 Image Puller，它是一个 OpenShift Dev Spaces-agnostic 组件，可用于预拉取 OpenShift 集群的镜像。

Image Puller 是一个额外的 OpenShift 部署，它会创建一个 *DaemonSet*，它可以配置为在每个节点上预拉取相关的 OpenShift Dev Spaces 工作区镜像。当 OpenShift Dev Spaces 工作区启动时，这些镜像将可用，从而改进了工作区启动时间。

- https://access.redhat.com/documentation/zh-cn/red_hat_openshift_dev_spaces/3.14/html-single/user_guide/index#installing-image-puller-on-kubernetes-by-using-cli

- 第 3.5.2 节 “使用 Web 控制台在 OpenShift 上安装 Image Puller”
- 第 3.5.1 节 “使用 CLI 在 OpenShift 上安装 Image Puller”
- 第 3.5.3 节 “将 Image Puller 配置为预拉取默认 Dev Spaces 镜像”
- 第 3.5.4 节 “将 Image Puller 配置为预拉取自定义镜像”
- 第 3.5.5 节 “配置 Image Puller 以预拉取额外镜像”

其他资源

- Kubernetes Image Puller 源代码存储库

3.5.1. 使用 CLI 在 OpenShift 上安装 Image Puller

您可以使用 OpenShift oc 管理工具在 OpenShift 上安装 Kubernetes Image Puller。

先决条件

- 具有 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 OpenShift CLI 入门。

流程

1. 进入链接来收集要拉取的相关容器镜像列表：
   - https://*<openshift_dev_spaces_fqdn>*/plugin-registry/v3/external_images.txt
   - https://*<openshift_dev_spaces_fqdn>*/devfile-registry/devfiles/external_images.txt
2. 定义内存请求和限值参数，以确保拉取的容器和平台有足够的内存来运行。

在为 **CACHING_MEMORY_REQUEST 或 CACHING_MEMORY_LIMIT 定义最小值时，请考虑运行每个容器镜像进行拉取所需的内存量。**

当为 **CACHING_MEMORY_REQUEST 或 CACHING_MEMORY_LIMIT 定义 maximal 值时，请考虑分配给集群中的 DaemonSet Pod 的总内存：**

```
(memory limit) * (number of images) * (number of nodes in the cluster)
```

在 20 个节点上拉取 5 个镜像，容器内存限值为 20Mi 需要 2000Mi 内存。

3. **克隆 Image Puller 存储库，并在包含 OpenShift 模板的目录中：**

```
$ git clone https://github.com/che-incubator/kubernetes-image-puller
$ cd kubernetes-image-puller/deploy/openshift
```

4. **使用以下参数配置 app.yaml、configmap.yaml 和 serviceaccount.yaml OpenShift 模板：**

表 3.37. app.yaml中的 Image Puller OpenShift 模板参数

值	使用方法	默认值
DEPLOYMENT_NAME	ConfigMap 中的 **DEPLOYMENT_NAME** 的值	**kubernetes-image-puller**
IMAGE	用于 **kubernetes-image-puller** 部署的镜像	**registry.redhat.io/devspaces/imagepuller-rhel8**
IMAGE_TAG	要拉取的镜像标签	**latest**
SERVICEACCOUNT_NAME	部署创建和使用的 ServiceAccount 的名称	**kubernetes-image-puller**

表 3.38. configmap.yaml 中的 Image Puller OpenShift 模板参数

值	使用方法	默认值
CACHING_CPU_LIMIT	ConfigMap 中的 **CACHING_CPU_LIMIT** 的值	**.2**

值	使用方法	默认值
CACHING_CPU_REQUEST	ConfigMap 中的 **CACHING_CPU_REQUEST** 的值	**.05**
CACHE_INTERVAL_HOURS	ConfigMap 中的 **CACHING_INTERVAL_HOURS** 的值	**"1"**
CACHING_MEMORY_LIMIT	ConfigMap 中的 **CACHING_MEMORY_LIMIT** 的值	**"20Mi"**
CACHING_MEMORY_REQUEST	ConfigMap 中的 **CACHING_MEMORY_REQUEST** 的值	**"10Mi"**
DAEMONSET_NAME	ConfigMap 中的 **DAEMONSET_NAME** 的值	**kubernetes-image-puller**
DEPLOYMENT_NAME	ConfigMap 中的 **DEPLOYMENT_NAME** 的值	**kubernetes-image-puller**
镜像	ConfigMap 中的 **IMAGES** 值	**{}**
NAMESPACE	ConfigMap 中的 **NAMESPACE** 值	**k8s-image-puller**
NODE_SELECTOR	ConfigMap 中的 **NODE_SELECTOR** 的值	**"{}"**

表 3.39. serviceaccount.yaml 中的 Image Puller OpenShift 模板参数

值	使用方法	默认值
SERVICEACCOUNT_NAME	部署创建和使用的 ServiceAccount 的名称	**kubernetes-image-puller**

5. 创建 **OpenShift** 项目来托管 **Image Puller**：

```
$ oc new-project <k8s-image-puller>
```

6. 处理并应用模板来安装拉取程序：

```
$ oc process -f serviceaccount.yaml | oc apply -f -
$ oc process -f configmap.yaml | oc apply -f -
$ oc process -f app.yaml | oc apply -f -
```

验证步骤

1. 验证是否存在 < ***kubernetes-image-puller*** > ***部署和*** < ***kubernetes-image-puller*** > DaemonSet。DaemonSet 需要为集群中的每个节点有一个 Pod：

```
$ oc get deployment,daemonset,pod --namespace <k8s-image-puller>
```

2. 验证 < ***kubernetes-image-puller*** > ConfigMap 的值。

```
$ oc get configmap <kubernetes-image-puller> --output yaml
```

3.5.2. 使用 Web 控制台在 OpenShift 上安装 Image Puller

您可以使用 OpenShift Web 控制台在 OpenShift 上安装支持的 Kubernetes Image Puller Operator。

先决条件

- 集群管理员的 OpenShift Web 控制台会话。请参阅 访问 Web 控制台。

流程

1. 安装社区支持的 Kubernetes Image Puller Operator。请参阅使用 Web 控制台从 OperatorHub 安装。

2. 从社区支持的 Kubernetes Image Puller Operator 创建一个 kubernetes-image-puller KubernetesImagePuller 操作对象。请参阅 从已安装的 Operator 创建应用程序。

3.5.3. 将 Image Puller 配置为预拉取默认 Dev Spaces 镜像

您可以将 Kubernetes Image Puller 配置为预拉取默认的 OpenShift Dev Spaces 镜像。Red Hat OpenShift Dev Spaces operator 将控制镜像列表到预拉取(pull)并在 OpenShift Dev Spaces 升级时自动更新它们。

先决条件

- 您的机构 OpenShift Dev Spaces 实例已安装并在 Kubernetes 集群上运行。
- Image Puller 在 Kubernetes 集群上安装。
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 将 Image Puller 配置为预拉取 OpenShift Dev Spaces 镜像。

```
oc patch checluster/devspaces \
  --namespace openshift-devspaces \
  --type='merge' \
  --patch '{
        "spec": {
          "components": {
            "imagePuller": {
              "enable": true
            }
          }
        }
      }'
```

3.5.4. 将 Image Puller 配置为预拉取自定义镜像

您可以将 Kubernetes Image Puller 配置为预拉取自定义镜像。

先决条件

- 您的机构 OpenShift Dev Spaces 实例已安装并在 Kubernetes 集群上运行。
- Image Puller 在 Kubernetes 集群上安装。
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 将 Image Puller 配置为预拉取自定义镜像。

```
oc patch checluster/devspaces \
  --namespace openshift-devspaces \
  --type='merge' \
  --patch '{
        "spec": {
          "components": {
            "imagePuller": {
              "enable": true,
              "spec": {
                "images": "NAME-1=IMAGE-1;NAME-2=IMAGE-2" (1)
              }
            }
          }
        }
      }'
```

(1) 分号分隔的镜像列表

3.5.5. 配置 Image Puller 以预拉取额外镜像

您可以将 Kubernetes Image Puller 配置为预拉取额外的 OpenShift Dev Spaces 镜像。

先决条件

- 您的机构 OpenShift Dev Spaces 实例已安装并在 Kubernetes 集群上运行。
- Image Puller 在 Kubernetes 集群上安装。
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 创建 k8s-image-puller 命名空间：

```
oc create namespace k8s-image-puller
```

2. 创建 KubernetesImagePuller 自定义资源：

```
oc apply -f - <<EOF
apiVersion: che.eclipse.org/v1alpha1
kind: KubernetesImagePuller
metadata:
  name: k8s-image-puller-images
  namespace: k8s-image-puller
spec:
  images: "__NAME-1__=__IMAGE-1__;__NAME-2__=__IMAGE-2__" (1)
EOF
```

(1) 分号分隔的镜像列表

添加资源

- Kubernetes Image Puller 源代码存储库
- 社区支持的 Kubernetes Image Puller Operator 源代码存储库

3.6. 配置可观察性

要配置 OpenShift Dev Spaces 可观察功能，请参阅：

- 第 3.6.2.14 节 “配置服务器日志记录”
- 第 3.6.2.15 节 “使用 dsc 收集日志”
- 第 3.6.3 节 “监控 Dev Workspace Operator”
- 第 3.6.4 节 “监控 Dev Spaces Server”

3.6.1. Woopra 遥测插件

Woopra Telemetry 插件 构建了一个插件，用于将遥测从 Red Hat OpenShift Dev Spaces 安装发送到 Segment 和 Woopra。此插件 供红帽托管的 Eclipse Che 使用，但任何 Red Hat OpenShift Dev Spaces 部署都可以利用此插件。没有有效的 Woopra 域和 Segment Write 密钥以外的依赖项。插件的 devfile v2 ， plugin.yaml，有四个可传递给插件的环境变量：

- WOOPRA_DOMAIN - 将事件发送到的 Woopra 域。
- SEGMENT_WRITE_KEY - 将事件发送到 Segment 和 Woopra 的写入密钥。
- WOOPRA_DOMAIN_ENDPOINT - 如果您不希望直接传递 Woopra 域，则该插件将从返回 Woopra 域的提供 HTTP 端点中获取。
- SEGMENT_WRITE_KEY_ENDPOINT - 如果您不希望直接传递 Segment 写入密钥，则该插件将从返回 Segment 写入密钥的所提供的 HTTP 端点中获取它。

在 Red Hat OpenShift Dev Spaces 安装中启用 Woopra 插件：

流程

- 将 plugin.yaml devfile v2 文件部署到 HTTP 服务器，并正确设置环境变量。

1. 配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  devEnvironments:
    defaultPlugins:
    - editor: eclipse/che-theia/next    1
      plugins:                          2
      - 'https://your-web-server/plugin.yaml'
```

1. 用于为其设置 telemetry 插件的 editorId。
2. 遥测插件 devfile v2 定义的 URL。

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.6.2. 创建 telemetry 插件

本节演示了如何创建扩展 AbstractAnalyticsManager 的 AnalyticsManager 类并实现以下方法：

- isEnabled （） - 决定遥测后端是否正常工作。这可能意味着始终返回 true，或者有更复杂的检查，例如在缺少连接属性时返回 false。
- destroy （） - 在关闭遥测后端前运行的清理方法。此方法发送 WORKSPACE_STOPPED 事件。
- onActivity （） - 通知给定用户仍发生一些活动。这主要用于发送 WORKSPACE_INACTIVE 事件。
- onEvent （） - 将遥测事件提交到遥测服务器，如 WORKSPACE_USED 或 WORKSPACE_STARTED。
- increaseDuration （） - 增加当前事件的持续时间，而不是在小时间段内发送多个事件。

以下部分涵盖了：

- 创建遥测服务器以将事件回显到标准输出.
- 扩展 OpenShift Dev Spaces 遥测客户端并实施用户的自定义后端。
- 创建一个 plugin.yaml 文件，代表自定义后端的 Dev Workspace 插件。
-

通过设置 CheCluster 自定义资源中的 workspacesDefaultPlugins 属性，指定自定义插件到 OpenShift Dev Spaces 的位置。

3.6.2.1. 开始使用

本文档描述了扩展 OpenShift Dev Spaces 遥测系统以与自定义后端通信所需的步骤：

1. 创建接收事件的服务器进程
2. 扩展 OpenShift Dev Spaces 库以创建向服务器发送事件的后端
3. 在容器中打包遥测后端并将其部署到镜像 registry
4. 为后端添加插件，并指示 OpenShift Dev Spaces 在 Dev Workspaces 中加载插件

此处 提供了遥测后端的完整示例。

创建接收事件的服务器

出于演示目的，本例演示了如何创建从遥测插件接收事件并将其写入标准输出的服务器。

对于生产环境，请考虑与第三方遥测系统（如 Segment、Woopra）集成，而不是创建自己的遥测服务器。在这种情况下，使用您的供应商的 API 将事件从自定义后端发送到其系统。

以下 Go 代码在端口 8080 上启动服务器，并将事件写入标准输出：

例 3.20. main.go

```
package main

import (
 "io/ioutil"
 "net/http"

 "go.uber.org/zap"
)
```

```
var logger *zap.SugaredLogger

func event(w http.ResponseWriter, req *http.Request) {
 switch req.Method {
 case "GET":
  logger.Info("GET /event")
 case "POST":
  logger.Info("POST /event")
 }
 body, err := req.GetBody()
 if err != nil {
  logger.With("err", err).Info("error getting body")
  return
 }
 responseBody, err := ioutil.ReadAll(body)
 if err != nil {
  logger.With("error", err).Info("error reading response body")
  return
 }
 logger.With("body", string(responseBody)).Info("got event")
}

func activity(w http.ResponseWriter, req *http.Request) {
 switch req.Method {
 case "GET":
  logger.Info("GET /activity, doing nothing")
 case "POST":
  logger.Info("POST /activity")
  body, err := req.GetBody()
  if err != nil {
   logger.With("error", err).Info("error getting body")
   return
  }
  responseBody, err := ioutil.ReadAll(body)
  if err != nil {
   logger.With("error", err).Info("error reading response body")
   return
  }
  logger.With("body", string(responseBody)).Info("got activity")
 }
}

func main() {

 log, _ := zap.NewProduction()
 logger = log.Sugar()

 http.HandleFunc("/event", event)
 http.HandleFunc("/activity", activity)
 logger.Info("Added Handlers")

 logger.Info("Starting to serve")
 http.ListenAndServe(":8080", nil)
}
```

基于此代码创建容器镜像，并在 openshift-devspaces 项目中将其公开为 OpenShift 中的部署。示例遥测服务器的代码位于 telemetry-server-example 中。要部署遥测服务器，请克隆存储库并构建容器：

```
$ git clone https://github.com/che-incubator/telemetry-server-example
$ cd telemetry-server-example
$ podman build -t registry/organization/telemetry-server-example:latest .
$ podman push registry/organization/telemetry-server-example:latest
```

manifest_with_ingress.yaml 和 manifest_with_route 都包含 Deployment 和 Service 的定义。前者还定义了 Kubernetes Ingress，后者则定义了 OpenShift Route。

在清单文件中，替换 image 和 host 字段以匹配您推送的镜像和 OpenShift 集群的公共主机名。然后运行：

```
$ kubectl apply -f manifest_with_[ingress|route].yaml -n openshift-devspaces
```

3.6.2.2. 创建后端项目

注意

为了在开发时快速反馈，建议在 Dev Workspace 中进行开发。这样，您可以在集群中运行应用程序，并从前端 telemetry 插件接收事件。

1. Maven Quarkus 项目构建：

```
mvn io.quarkus:quarkus-maven-plugin:2.7.1.Final:create \
    -DprojectGroupId=mygroup -DprojectArtifactId=devworkspace-telemetry-example-plugin \
-DprojectVersion=1.0.0-SNAPSHOT
```

2. 删除 src/main/java/mygroup 和 src/test/java/mygroup 下的文件。

3. 如需最新的版本，请参阅 GitHub 软件包，以及 backend-base 的 Maven 协调。

4. 将以下依赖项添加到 pom.xml 中：

例 3.21. pom.xml

```
<!-- Required -->
<dependency>
  <groupId>org.eclipse.che.incubator.workspace-telemetry</groupId>
  <artifactId>backend-base</artifactId>
  <version>LATEST VERSION FROM PREVIOUS STEP</version>
</dependency>


<!-- Used to make http requests to the telemetry server -->
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-rest-client</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-rest-client-jackson</artifactId>
</dependency>
```

5. 创建一个具有 **read:packages** 权限的个人访问令牌，以从 **GitHub** 软件包下载 **org.eclipse.che.incubator.workspace-telemetry:backend-base** 依赖项。

6. 在 **~/.m2/settings.xml** 文件中添加您的 **GitHub** 用户名、个人访问令牌和 **che-incubator** 存储库详情：

例 3.22. settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>che-incubator</id>
      <username>YOUR GITHUB USERNAME</username>
      <password>YOUR GITHUB TOKEN</password>
    </server>
  </servers>

  <profiles>
    <profile>
      <id>github</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <repositories>
        <repository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>false</enabled></snapshots>
          </repository>
```

```
            <repository>
            <id>che-incubator</id>
            <url>https://maven.pkg.github.com/che-incubator/che-workspace-
telemetry-client</url>
          </repository>
        </repositories>
      </profile>
    </profiles>
</settings>
```

3.6.2.3. 创建分析管理器的具体实施并添加专用逻辑

在项目的 src/main/java/mygroup 下创建两个文件：

- MainConfiguration.java - 包含提供给 分析管理器 的配置。
- AnalyticsManager.java - 包含特定于遥测系统的逻辑。

例 3.23. MainConfiguration.java

```
package org.my.group;

import java.util.Optional;

import javax.enterprise.context.Dependent;
import javax.enterprise.inject.Alternative;

import org.eclipse.che.incubator.workspace.telemetry.base.BaseConfiguration;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Dependent
@Alternative
public class MainConfiguration extends BaseConfiguration {
  @ConfigProperty(name = "welcome.message")  1
  Optional<String> welcomeMessage;           2
}
```

1. MicroProfile 配置注释用于注入 welcome.message 配置。

有关如何设置特定于后端的配置属性的详情，请参阅 Quarkus 配置参考指南。

例 3.24. AnalyticsManager.java

```
package org.my.group;

import java.util.HashMap;
import java.util.Map;

import javax.enterprise.context.Dependent;
import javax.enterprise.inject.Alternative;
import javax.inject.Inject;

import org.eclipse.che.incubator.workspace.telemetry.base.AbstractAnalyticsManager;
import org.eclipse.che.incubator.workspace.telemetry.base.AnalyticsEvent;
import org.eclipse.che.incubator.workspace.telemetry.finder.DevWorkspaceFinder;
import org.eclipse.che.incubator.workspace.telemetry.finder.UsernameFinder;
import org.eclipse.microprofile.rest.client.inject.RestClient;
import org.slf4j.Logger;

import static org.slf4j.LoggerFactory.getLogger;

@Dependent
@Alternative
public class AnalyticsManager extends AbstractAnalyticsManager {

  private static final Logger LOG = getLogger(AbstractAnalyticsManager.class);

  public AnalyticsManager(MainConfiguration mainConfiguration, DevWorkspaceFinder
devworkspaceFinder, UsernameFinder usernameFinder) {
    super(mainConfiguration, devworkspaceFinder, usernameFinder);

    mainConfiguration.welcomeMessage.ifPresentOrElse(     1
      (str) -> LOG.info("The welcome message is: {}", str),
      () -> LOG.info("No welcome message provided")
    );
  }

  @Override
  public boolean isEnabled() {
    return true;
  }

  @Override
  public void destroy() {}

  @Override
  public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent,
String resolution, Map<String, Object> properties) {
    LOG.info("The received event is: {}", event);         2
  }

  @Override
  public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) { }
```

```
    @Override
    public void onActivity() {}
}
```

1 如果提供，请记录欢迎消息。

2 记录从前端插件接收的事件。

由于 org.my.group.AnalyticsManager 和 org.my.group.MainConfiguration 是替代 Bean，使用 src/main/resources/application.properties 中的 quarkus.arc.selected-alternatives 属性来指定它们。

例 3.25. application.properties

```
quarkus.arc.selected-alternatives=MainConfiguration,AnalyticsManager
```

3.6.2.4. 在 Dev Workspace 中运行应用程序

1. 在 Dev Workspace 中设置 DEVWORKSPACE_TELEMETRY_BACKEND_PORT 环境变量。在这里，值设为 4167。

```
spec:
  template:
    attributes:
      workspaceEnv:
        - name: DEVWORKSPACE_TELEMETRY_BACKEND_PORT
          value: '4167'
```

2. 从 Red Hat OpenShift Dev Spaces 仪表板中重启 Dev Workspace。

3. 在 Dev Workspace 的终端窗口中运行以下命令以启动应用程序。使用 --settings 标志指定包含 GitHub 访问令牌的 settings.xml 文件的位置路径。

```
$ mvn --settings=settings.xml quarkus:dev -
Dquarkus.http.port=${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}
```

现在，应用程序通过前端插件的端口 4167 接收遥测事件。

验证步骤

1. 验证是否记录以下输出：

```
INFO  [org.ecl.che.inc.AnalyticsManager] (Quarkus Main Thread) No welcome message provided
INFO  [io.quarkus] (Quarkus Main Thread) devworkspace-telemetry-example-plugin 1.0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.2.Final) started in 0.323s. Listening on: http://localhost:4167
INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, kubernetes-client, rest-client, rest-client-jackson, resteasy, resteasy-jsonb, smallrye-context-propagation, smallrye-openapi, swagger-ui, vertx]
```

2. 要验证 AnalyticsManager 的 onEvent （） 方法是否从前端插件接收事件，请按 l 键来禁用 Quarkus 实时编码并编辑 IDE 中的任何文件。应记录以下输出：

```
INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor] (Aesh InputStream Reader) Live reload disabled
INFO  [org.ecl.che.inc.AnalyticsManager] (executor-thread-2) The received event is: Edit Workspace File in Che
```

3.6.2.5. 实施 isEnabled （）

在示例中，此方法始终会在调用时返回 true。

例 3.26. AnalyticsManager.java

```
@Override
public boolean isEnabled() {
   return true;
}
```

可以在 isEnabled （） 中放置更复杂的逻辑。例如，托管的 OpenShift Dev Spaces Woopra 后端在确定是否启用了后端前检查配置属性是否存在。

3.6.2.6. 实施 onEvent （）

onEvent （） 将后端接收的事件发送到遥测系统。对于 example 应用，它将 HTTP POST 有效负载发送到遥测服务器的 /event 端点。

3.6.2.6.1. 向示例 telemetry 服务器发送 POST 请求

在以下示例中，遥测服务器应用通过以下 URL 部署到 OpenShift：其中 apps-crc.testing 是 OpenShift 集群的入口域名。

1. 通过创建 TelemetryService.java来设置 RESTEasy REST 客户端

例 3.27. TelemetryService.java

```
package org.my.group;

import java.util.Map;

import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

@RegisterRestClient
public interface TelemetryService {
   @POST
   @Path("/event") 1
   @Consumes(MediaType.APPLICATION_JSON)
   Response sendEvent(Map<String, Object> payload);
}
```

1 将 POST 请求发送到的端点。

2. 在 src/main/resources/application.properties 文件中指定 TelemetryService 的基本 URL：

例 3.28. application.properties

```
org.my.group.TelemetryService/mp-rest/url=http://little-telemetry-server-che.apps-crc.testing
```

3. 将 TelemetryService 注入 AnalyticsManager，并在 onEvent （）中发送 POST 请求

例 3.29. AnalyticsManager.java

```
@Dependent
@Alternative
public class AnalyticsManager extends AbstractAnalyticsManager {
    @Inject
    @RestClient
    TelemetryService telemetryService;

...

@Override
public void onEvent(AnalyticsEvent event, String ownerId, String ip, String
userAgent, String resolution, Map<String, Object> properties) {
    Map<String, Object> payload = new HashMap<String, Object>(properties);
    payload.put("event", event);
    telemetryService.sendEvent(payload);
}
```

这会将 HTTP 请求发送到遥测服务器，并在少量时间内自动延迟相同的事件。默认持续时间为 1500 毫秒。

3.6.2.7. 实施 increaseDuration （）

许多遥测系统都识别事件持续时间。AbstractAnalyticsManager 将相同时间段内发生的类似事件合并到一个事件中。这个 increaseDuration （） 的实现是一个 no-op。此方法使用用户遥测供应商的 API 更改事件或事件属性，以反映事件的持续时间增加。

例 3.30. AnalyticsManager.java

```
@Override
public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) {}
```

3.6.2.8. 实施 onActivity （）

设置不活跃超时限制，并使用 onActivity （） 发送 WORKSPACE_INACTIVE 事件（如果最后一次事件时间超过超时）。

例 3.31. AnalyticsManager.java

```
public class AnalyticsManager extends AbstractAnalyticsManager {

  ...

  private long inactiveTimeLimit = 60000 * 3;
```

```
    ...

    @Override
    public void onActivity() {
        if (System.currentTimeMillis() - lastEventTime >= inactiveTimeLimit) {
            onEvent(WORKSPACE_INACTIVE, lastOwnerId, lastIp, lastUserAgent,
lastResolution, commonProperties);
        }
    }
```

3.6.2.9. 实施 destroy （）

当调用 destroy （） 时，发送 WORKSPACE_STOPPED 事件并关闭连接池等任何资源。

例 3.32. AnalyticsManager.java

```
@Override
public void destroy() {
    onEvent(WORKSPACE_STOPPED, lastOwnerId, lastIp, lastUserAgent, lastResolution,
commonProperties);
}
```

运行 mvn quarkus:dev，如 第 3.6.2.4 节 “在 Dev Workspace 中运行应用程序” 所述，并使用 Ctrl+C 终止应用程序向服务器发送 WORKSPACE_STOPPED 事件。

3.6.2.10. 打包 Quarkus 应用程序

有关在容器中打包应用程序的最佳说明，请参阅 Quarkus 文档。构建容器并将其推送到您选择的容器注册中心。

3.6.2.10.1. 用于构建使用 JVM 运行的 Quarkus 镜像的 Dockerfile 示例

例 3.33. Dockerfile.jvm

```
FROM registry.access.redhat.com/ubi8/openjdk-11:1.11

ENV LANG='en_US.UTF-8' LANGUAGE='en_US:en'

COPY --chown=185 target/quarkus-app/lib/ /deployments/lib/
COPY --chown=185 target/quarkus-app/*.jar /deployments/
COPY --chown=185 target/quarkus-app/app/ /deployments/app/
COPY --chown=185 target/quarkus-app/quarkus/ /deployments/quarkus/

EXPOSE 8080
```

```
USER 185

ENTRYPOINT ["java", "-Dquarkus.http.host=0.0.0.0", "-
Djava.util.logging.manager=org.jboss.logmanager.LogManager", "-
Dquarkus.http.port=${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}", "-jar",
"/deployments/quarkus-run.jar"]
```

要构建镜像，请运行：

```
mvn package && \
podman build -f src/main/docker/Dockerfile.jvm -t image:tag .
```

3.6.2.10.2. 用于构建 Quarkus 原生镜像的 Dockerfile 示例

例 3.34. Dockerfile.native

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.5
WORKDIR /work/
RUN chown 1001 /work \
    && chmod "g+rwX" /work \
    && chown 1001:root /work
COPY --chown=1001:root target/*-runner /work/application

EXPOSE 8080
USER 1001

CMD ["./application", "-Dquarkus.http.host=0.0.0.0", "-
Dquarkus.http.port=$DEVWORKSPACE_TELEMETRY_BACKEND_PORT}"]
```

要构建镜像，请运行：

```
mvn package -Pnative -Dquarkus.native.container-build=true && \
podman build -f src/main/docker/Dockerfile.native -t image:tag .
```

3.6.2.11. 为插件创建 plugin.yaml

创建一个 plugin.yaml devfile v2 文件，该文件代表 Dev Workspace 插件，该文件在 Dev Workspace Pod 中运行自定义后端。有关 devfile v2 的更多信息，请参阅 Devfile v2 文档

例 3.35. plugin.yaml

```
schemaVersion: 2.1.0
metadata:
  name: devworkspace-telemetry-backend-plugin
```

```
  version: 0.0.1
  description: A Demo telemetry backend
  displayName: Devworkspace Telemetry Backend
components:
  - name: devworkspace-telemetry-backend-plugin
    attributes:
      workspaceEnv:
        - name: DEVWORKSPACE_TELEMETRY_BACKEND_PORT
          value: '4167'
    container:
      image: YOUR IMAGE          1
      env:
        - name: WELCOME_MESSAGE    2
          value: 'hello world!'
```

1 指定从 第 3.6.2.10 节 “打包 Quarkus 应用程序” 构建的容器镜像。

2 为 Example 4 中的 welcome.message 可选配置属性设置值。

通常，用户将此文件部署到企业 Web 服务器。本指南演示了如何在 OpenShift 上创建 Apache Web 服务器，并在其中托管插件。

创建引用新 plugin.yaml 文件的 ConfigMap 对象。

```
$ oc create configmap --from-file=plugin.yaml -n openshift-devspaces telemetry-plugin-yaml
```

创建部署、服务和路由，以公开 Web 服务器。部署引用此 ConfigMap 对象并将其放在 /var/www/html 目录中。

例 3.36. manifest.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apache
  template:
```

```
    metadata:
      labels:
        app: apache
    spec:
      volumes:
        - name: plugin-yaml
          configMap:
            name: telemetry-plugin-yaml
            defaultMode: 420
      containers:
        - name: apache
          image: 'registry.redhat.io/rhscl/httpd-24-rhel7:latest'
          ports:
            - containerPort: 8080
              protocol: TCP
          resources: {}
          volumeMounts:
            - name: plugin-yaml
              mountPath: /var/www/html
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
---
kind: Service
apiVersion: v1
metadata:
  name: apache
spec:
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  selector:
    app: apache
  type: ClusterIP
---
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: apache
spec:
  host: apache-che.apps-crc.testing
  to:
    kind: Service
    name: apache
    weight: 100
  port:
    targetPort: 8080
  wildcardPolicy: None
```

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

验证步骤

- 部署启动后，确认 web 服务器中的 plugin.yaml 可用：

```
$ curl apache-che.apps-crc.testing/plugin.yaml
```

3.6.2.12. 在 Dev Workspace 中指定 telemetry 插件

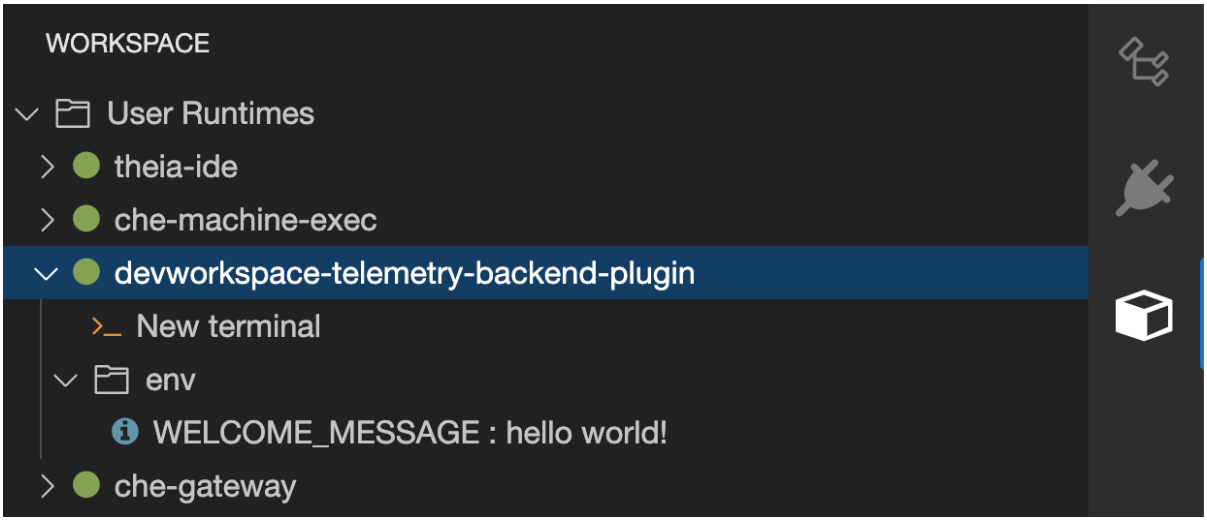
1. 将以下内容添加到现有 Dev Workspace 的 components 字段中：

```
components:
  ...
  - name: telemetry-plugin
    plugin:
      uri: http://apache-che.apps-crc.testing/plugin.yaml
```

2. 从 OpenShift Dev Spaces 仪表板启动 Dev Workspace。

验证步骤

1. 验证 telemetry 插件容器是否在 Dev Workspace pod 中运行。在这里，这可以通过检查编辑器中的 Workspace 视图进行验证。



2. 在编辑器中编辑文件，并在示例遥测服务器日志中观察其事件。

3.6.2.13. 为所有 Dev Workspaces 应用 telemetry 插件

将 telemetry 插件设置为默认插件。默认插件应用于新的和现有的 Dev Workspaces 的 Dev Workspace 启动。

- 配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  devEnvironments:
    defaultPlugins:
    - editor: eclipse/che-theia/next      1
      plugins:                            2
      - 'http://apache-che.apps-crc.testing/plugin.yaml'
```

1. 用于为其设置默认插件的编辑器标识。

2. devfile v2 插件的 URL 列表。

其他资源

- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”.

验证步骤

1. 从 Red Hat OpenShift Dev Spaces 仪表板中启动新的或现有的 Dev Workspace。

2. 按照 第 3.6.2.12 节 “在 Dev Workspace 中指定 telemetry 插件” 的验证步骤，验证 telemetry 插件是否正常工作。

3.6.2.14. 配置服务器日志记录

可以在 OpenShift Dev Spaces 服务器中微调各个日志记录器的日志级别。

整个 OpenShift Dev Spaces 服务器的日志级别使用 Operator 的 cheLogLevel 配置属性进行全局配置。请参阅 第 3.1.3 节 “CheCluster 自定义资源字段参考”。要在不由 Operator 管理的安装中设置全局

日志级别，请在 che ConfigMap 中指定 CHE_LOG_LEVEL 环境变量。

可以使用 CHE_LOGGER_CONFIG 环境变量在 OpenShift Dev Spaces 服务器中配置各个日志记录器的日志级别。

3.6.2.14.1. 配置日志级别

流程

- 配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LOGGER_CONFIG: "<key1=value1,key2=value2>" (1)
```

(1) 以逗号分隔的键值对列表，其中键是 OpenShift Dev Spaces 服务器日志输出中所示的日志记录器名称，值是所需的日志级别。

例 3.37. 为 WorkspaceManager配置调试模式

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LOGGER_CONFIG:
"org.eclipse.che.api.workspace.server.WorkspaceManager=DEBUG"
```

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.6.2.14.2. 日志记录器命名

日志记录器的名称遵循使用这些日志记录器的内部服务器类的类名称。

3.6.2.14.3. 日志记录 HTTP 流量

流程

- 要在 OpenShift Dev Spaces 服务器和 Kubernetes 或 OpenShift 集群的 API 服务器之间记录 HTTP 流量，请配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LOGGER_CONFIG: "che.infra.request-logging=TRACE"
```

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.6.2.15. 使用 dsc 收集日志

Red Hat OpenShift Dev Spaces 安装由 OpenShift 集群中运行的几个容器组成。虽然可以从每个正在运行的容器中手动收集日志，但 dsc 提供了可自动化进程的命令。

以下命令可以使用 dsc 工具从 OpenShift 集群收集 Red Hat OpenShift Dev Spaces 日志：

DSC 服务器：logs

收集现有的 Red Hat OpenShift Dev Spaces 服务器日志，并将其存储在本地计算机上的目录中。默认情况下，日志下载到机器上的临时目录中。但是，这可以通过指定 -d 参数来覆盖。例如，要将 OpenShift Dev Spaces 日志下载到 /home/user/che-logs/ 目录中，请使用 命令

```
dsc server:logs -d /home/user/che-logs/
```

运行时，dsc server:logs 在控制台中打印一条消息，指定将存储日志文件的目录：

```
Red Hat OpenShift Dev Spaces logs will be available in '/tmp/chectl-logs/1648575098344'
```

如果在非默认项目中安装了 Red Hat OpenShift Dev Spaces，则 dsc server:logs 需要 -n <NAMESPACE> paremeter，其中 & lt;NAMESPACE > 是安装 Red Hat OpenShift Dev Spaces 的 OpenShift 项目。例如，若要从 my-namespace 项目中的 OpenShift Dev Spaces 获取日志，请使用 命令

```
dsc server:logs -n my-namespace
```

dsc server:deploy

使用 dsc 安装时，会在 OpenShift Dev Spaces 安装过程中自动收集日志。与 dsc server:logs 一样，可以使用 -d 参数指定目录日志存储在 中。

其他资源

- "DDS 参考文档"

3.6.3. 监控 Dev Workspace Operator

您可以配置 OpenShift in-cluster 监控堆栈来提取 Dev Workspace Operator 公开的指标。

3.6.3.1. 收集 Dev Workspace Operator 指标

使用 in-cluster Prometheus 实例来收集、存储和查询 Dev Workspace Operator 的指标：

先决条件

- 您的机构 OpenShift Dev Spaces 实例已安装并在 Red Hat OpenShift 中运行。
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。
- devworkspace-controller-metrics 服务在端口 8443 上公开指标。默认情况下，这是预配置的。

流程

1. 创建用于检测 Dev Workspace Operator 指标服务的 ServiceMonitor。

例 3.38. ServiceMonitor

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: devworkspace-controller
  namespace: openshift-devspaces (1)
spec:
  endpoints:
    - bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
      interval: 10s (2)
      port: metrics
      scheme: https
      tlsConfig:
        insecureSkipVerify: true
  namespaceSelector:
    matchNames:
      - openshift-operators
  selector:
    matchLabels:
      app.kubernetes.io/name: devworkspace-controller
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

(2) 提取目标的速度。

2. 允许 in-cluster Prometheus 实例检测 OpenShift Dev Spaces 命名空间中的 ServiceMonitor。默认的 OpenShift Dev Spaces 命名空间为 openshift-devspaces。

```
$ oc label namespace openshift-devspaces openshift.io/cluster-monitoring=true
```

验证

1. 对于 OpenShift Dev Spaces 的新安装，请通过从 Dashboard 创建 OpenShift Dev Spaces 工作区来生成指标。

2. 在 OpenShift Web 控制台的 Administrator 视图中，进入 Observe → Metrics。

3. 运行 PromQL 查询以确认指标可用。例如，输入 devworkspace_started_total 并点 Run query。

   如需了解更多指标，请参阅 第 3.6.3.2 节 “dev Workspace 特定指标”。

提示

要排除缺少的指标，请查看 Prometheus 容器日志以了解与 RBAC 相关的错误：

1. 获取 Prometheus pod 的名称：

```
$ oc get pods -l app.kubernetes.io/name=prometheus -n openshift-monitoring -o=jsonpath='{.items[*].metadata.name}'
```

2. 从上一步中的 Prometheus pod 输出 Prometheus 容器日志的最后 20 行：

```
$ oc logs --tail=20 <prometheus_pod_name> -c prometheus -n openshift-monitoring
```

其他资源

- 查询 Prometheus
- Prometheus 指标类型

3.6.3.2. dev Workspace 特定指标

下表描述了 devworkspace-controller-metrics 服务公开的 Dev Workspace 特定指标。

表 3.40. 指标

Name	类型	描述	标签
devworkspace_started_total	计数	Dev Workspace 启动事件的数量。	源,routingclass
devworkspace_started_success_total	计数	Dev Workspaces 的数量成功进入 Running 阶段。	源,routingclass

Name	类型	描述	标签
devworkspace_fail_total	计数	失败的 Dev Workspaces 数量。	**源**,**reason**
devworkspace_startup_time	Histogram	启动 Dev Workspace 的总时间（以秒为单位）。	**源**,**routingclass**

表 3.41. 标签

Name	描述	值
source	Dev Workspace 的 **controller.devfile.io/devworkspace-source** 标签。	**字符串**
routingclass	Dev Workspace 的 **spec.routingclass**。	**"basic|cluster|cluster-tls|web-terminal"**
reason	工作区启动失败原因。	**"BadRequest|InfrastructureFailure|Unknown"**

表 3.42. 启动失败原因

Name	描述
BadRequest	由于用于创建 Dev Workspace 的无效 devfile 而启动失败。
InfrastructureFailure	由于以下错误导致启动失败：**CreateContainerError**,**RunContainerError**,**FailedScheduling**,**FailedMount**.
Unknown	未知失败原因。

3.6.3.3. 从 OpenShift Web 控制台仪表板查看 Dev Workspace Operator 指标

配置 in-cluster Prometheus 实例来收集 Dev Workspace Operator 指标后，您可以在 OpenShift Web 控制台的 Administrator 视角中在自定义仪表板上查看指标。

先决条件

- **您的机构 OpenShift Dev Spaces 实例已安装并在 Red Hat OpenShift 中运行。**

- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

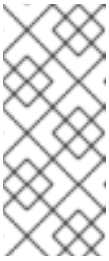
- in-cluster Prometheus 实例正在收集指标。请参阅 第 3.6.3.1 节 “收集 Dev Workspace Operator 指标”。

流程

- 在 openshift-config-managed 项目中为仪表板定义创建 ConfigMap，并应用所需的标签。

a.

```
$ oc create configmap grafana-dashboard-dwo \
  --from-literal=dwo-dashboard.json="$(curl
https://raw.githubusercontent.com/devfile/devworkspace-
operator/main/docs/grafana/openshift-console-dashboard.json)" \
  -n openshift-config-managed
```

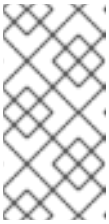


注意

上一命令包含来自上游社区的材料的链接。本材料代表了最新可用内容和最新的最佳实践。这些提示尚未被红帽的 QE 部门审查，它们尚未被广泛的用户组验证。请谨慎使用此信息。

b.

```
$ oc label configmap grafana-dashboard-dwo console.openshift.io/dashboard=true -n
openshift-config-managed
```



注意

仪表板定义基于 Grafana 6.x 仪表板。OpenShift Web 控制台不支持所有 Grafana 6.x 仪表板功能。

验证步骤

1. 在 OpenShift Web 控制台的 Administrator 视图中，进入 Observe → Dashboards。

2. 前往 Dashboard → Che Server JVM，再验证仪表板面板是否包含数据。

3.6.3.4. Dev Workspace Operator 的仪表板

OpenShift Web 控制台自定义仪表板基于 Grafana 6.x，显示 Dev Workspace Operator 中的以下指标。

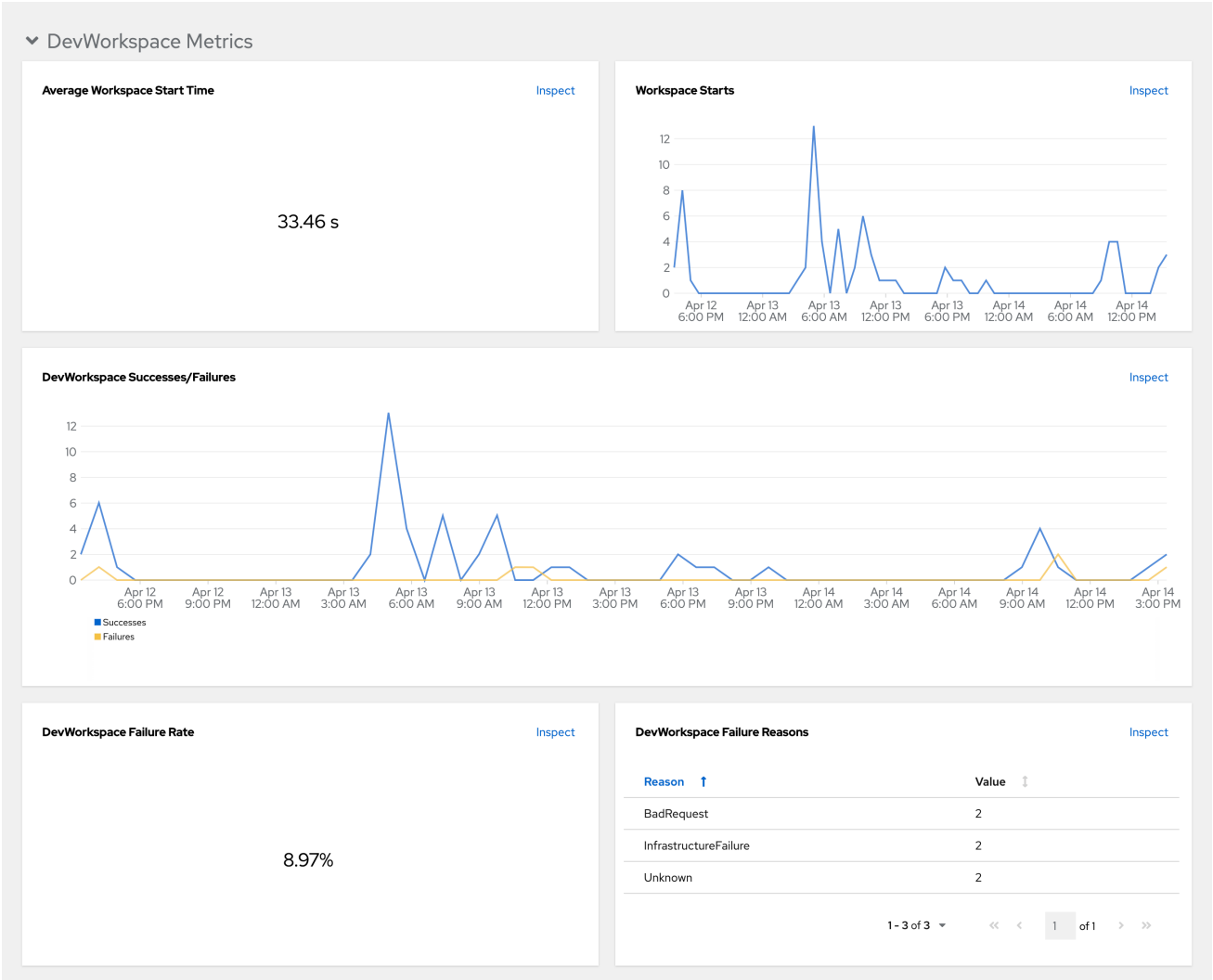
注意

不是 Grafana 6.x 仪表板的所有功能都作为 OpenShift Web 控制台仪表板支持。

3.6.3.4.1. dev Workspace 指标

Dev Workspace 特定的指标显示在 Dev Workspace Metrics 面板中。

图 3.1. Dev Workspace Metrics 面板



平均工作区启动时间

平均工作区启动持续时间。

工作区启动

成功和失败的工作区启动数量。

dev Workspace 成功和失败

成功和失败的 Dev Workspace 启动之间的比较。

dev Workspace 失败率

失败工作区启动数量和工作空间启动总数之间的比率。

dev Workspace 启动失败原因

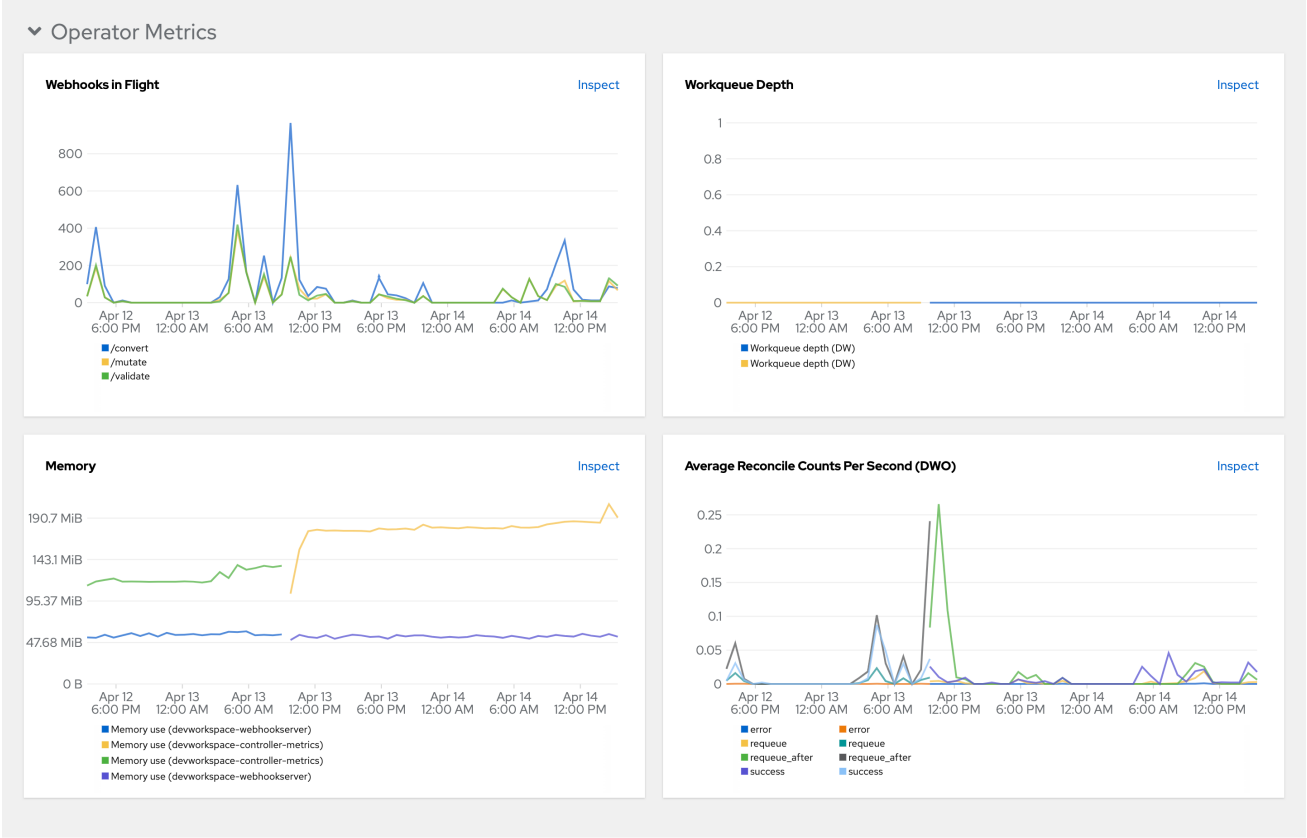
显示工作区启动故障分布的 pie chart：

- **BadRequest**
- **InfrastructureFailure**
- **Unknown**

3.6.3.4.2. Operator 指标

Operator 特定指标显示在 Operator Metrics 面板中。

图 3.2. Operator 指标面板



flight 中的 Webhook

不同 Webhook 请求数量之间的比较。

工作队列深度

工作队列中的协调请求数。

memory

Dev Workspace 控制器和 Dev Workspace Webhook 服务器的内存用量。

平均每秒协调数(DWO)

Dev Workspace 控制器的平均每秒协调数。

3.6.4. 监控 Dev Spaces Server

您可以配置 OpenShift Dev Spaces 以公开 JVM 指标，如 JVM 内存和 OpenShift Dev Spaces 服务器的类加载。

3.6.4.1. 启用和公开 OpenShift Dev Spaces 服务器指标

OpenShift Dev Spaces 在 che-host Service 的端口 8087 上公开 JVM 指标。您可以配置此行为。

流程

- 配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  components:
    metrics:
      enable: <boolean> 1
```

1 true 来启用，false 会禁用。

3.6.4.2. 使用 Prometheus 收集 OpenShift Dev Spaces Server 指标

使用 in-cluster Prometheus 实例为 OpenShift Dev Spaces Server 收集、存储和查询 JVM 指标：

先决条件

- 您的机构 OpenShift Dev Spaces 实例已安装并在 Red Hat OpenShift 中运行。
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。
- OpenShift Dev Spaces 在端口 8087 上公开指标。请参阅启用和公开 OpenShift Dev Spaces 服务器 JVM 指标。

流程

1. 创建用于检测 OpenShift Dev Spaces JVM 指标服务的 ServiceMonitor。

例 3.39. ServiceMonitor

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: che-host
  namespace: openshift-devspaces 1
```

```
spec:
  endpoints:
    - interval: 10s (2)
      port: metrics
      scheme: http
  namespaceSelector:
    matchNames:
      - openshift-devspaces (3)
  selector:
    matchLabels:
      app.kubernetes.io/name: devspaces
```

(1) (3) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

(2) 提取目标的速度。

2. 创建 Role 和 RoleBinding，以允许 Prometheus 查看指标。

例 3.40. 角色

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: prometheus-k8s
  namespace: openshift-devspaces (1)
rules:
  - verbs:
      - get
      - list
      - watch
    apiGroups:
      - ''
    resources:
      - services
      - endpoints
      - pods
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

例 3.41. RoleBinding

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: view-devspaces-openshift-monitoring-prometheus-k8s
  namespace: openshift-devspaces (1)
subjects:
  - kind: ServiceAccount
    name: prometheus-k8s
    namespace: openshift-monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: prometheus-k8s
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

3. 允许 in-cluster Prometheus 实例检测 OpenShift Dev Spaces 命名空间中的 ServiceMonitor。默认的 OpenShift Dev Spaces 命名空间为 openshift-devspaces。

```
$ oc label namespace openshift-devspaces openshift.io/cluster-monitoring=true
```

验证

1. 在 OpenShift Web 控制台的 Administrator 视图中，进入 Observe → Metrics。

2. 运行 PromQL 查询以确认指标可用。例如，输入 process_uptime_seconds{job="che-host"}，然后点 Run query。

提示

要排除缺少的指标，请查看 Prometheus 容器日志以了解与 RBAC 相关的错误：

1. 获取 Prometheus pod 的名称：

```
$ oc get pods -l app.kubernetes.io/name=prometheus -n openshift-monitoring -
o=jsonpath='{.items[*].metadata.name}'
```

2. 从上一步中的 Prometheus pod 输出 Prometheus 容器日志的最后 20 行：

```
$ oc logs --tail=20 <prometheus_pod_name> -c prometheus -n openshift-monitoring
```

其他资源

- 查询 Prometheus
- Prometheus 指标类型

3.6.4.3. 从 OpenShift Web 控制台仪表板查看 OpenShift Dev Spaces Server

配置 in-cluster Prometheus 实例来收集 OpenShift Dev Spaces Server JVM 指标后，您可以在 OpenShift Web 控制台的 Administrator 视角中在自定义仪表板上查看指标。

先决条件

- 您的机构 OpenShift Dev Spaces 实例已安装并在 Red Hat OpenShift 中运行。
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。
- in-cluster Prometheus 实例正在收集指标。请参阅 第 3.6.4.2 节 “使用 Prometheus 收集 OpenShift Dev Spaces Server 指标”。

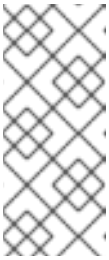
流程

- 在 openshift-config-managed 项目中为仪表板定义创建 ConfigMap，并应用所需的标

签。

a.

```
$ oc create configmap grafana-dashboard-devspaces-server \
  --from-literal=devspaces-server-dashboard.json="$(curl
https://raw.githubusercontent.com/eclipse-che/che-server/main/docs/grafana/openshift-
console-dashboard.json)" \
  -n openshift-config-managed
```



注意

上一命令包含来自上游社区的材料的链接。本材料代表了最新可用内容和最新的最佳实践。这些提示尚未被红帽的 QE 部门审查，它们尚未被广泛的用户组验证。请谨慎使用此信息。

b.

```
$ oc label configmap grafana-dashboard-devspaces-server
console.openshift.io/dashboard=true -n openshift-config-managed
```

注意

仪表板定义基于 **Grafana 6.x** 仪表板。**OpenShift Web** 控制台不支持所有 **Grafana 6.x** 仪表板功能。

验证步骤

1. **在 OpenShift Web 控制台的 Administrator 视图中，进入 Observe → Dashboards。**

2. **进入 Dashboard → Dev Workspace Operator，验证仪表板面板是否包含数据。**

图 3.3. 快速事实

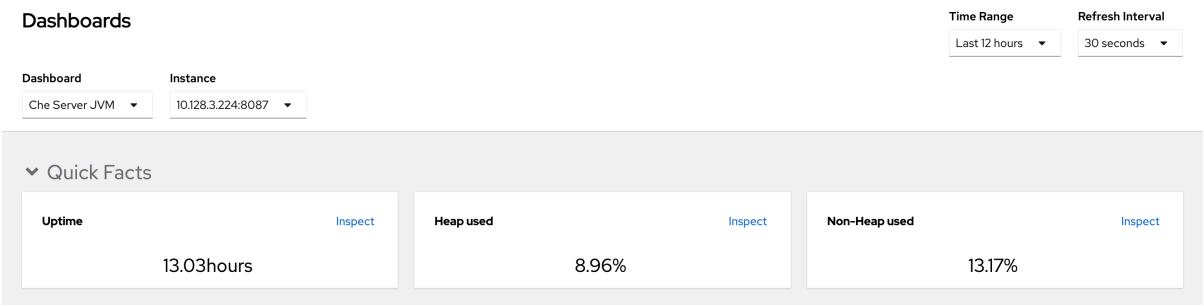


图 3.4. JVM 内存

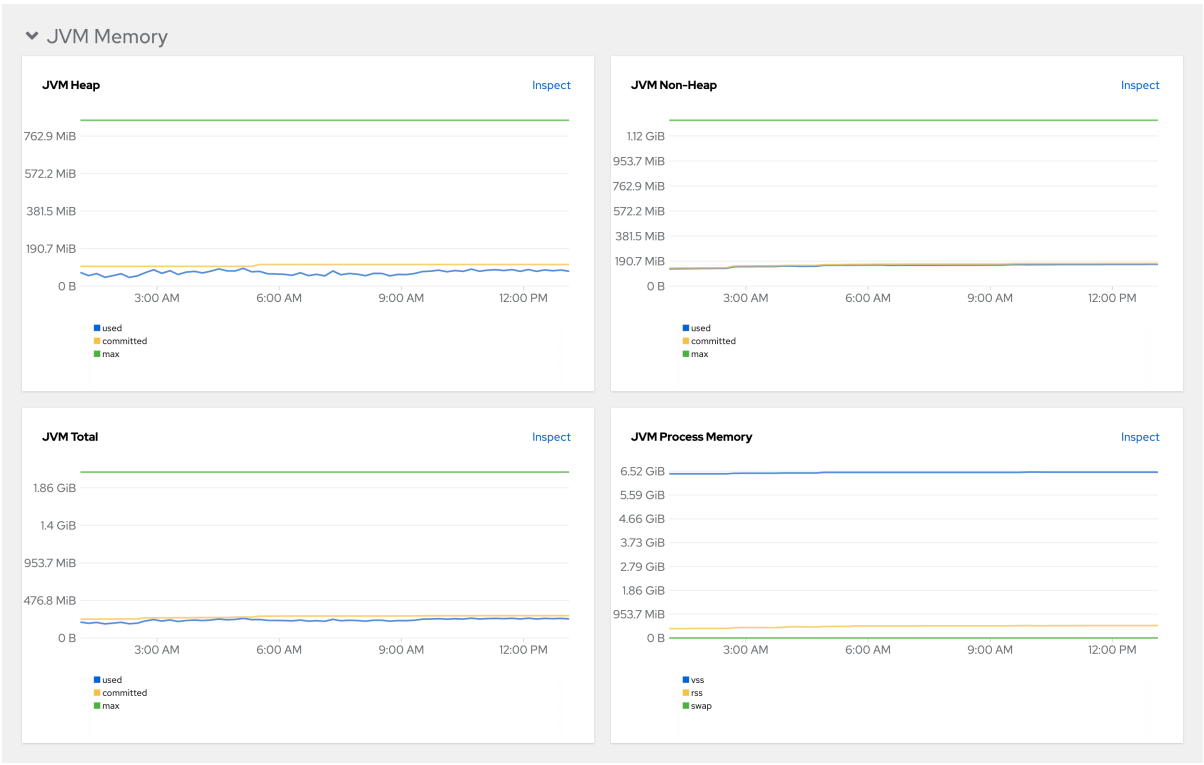


图 3.5. JVM Misc

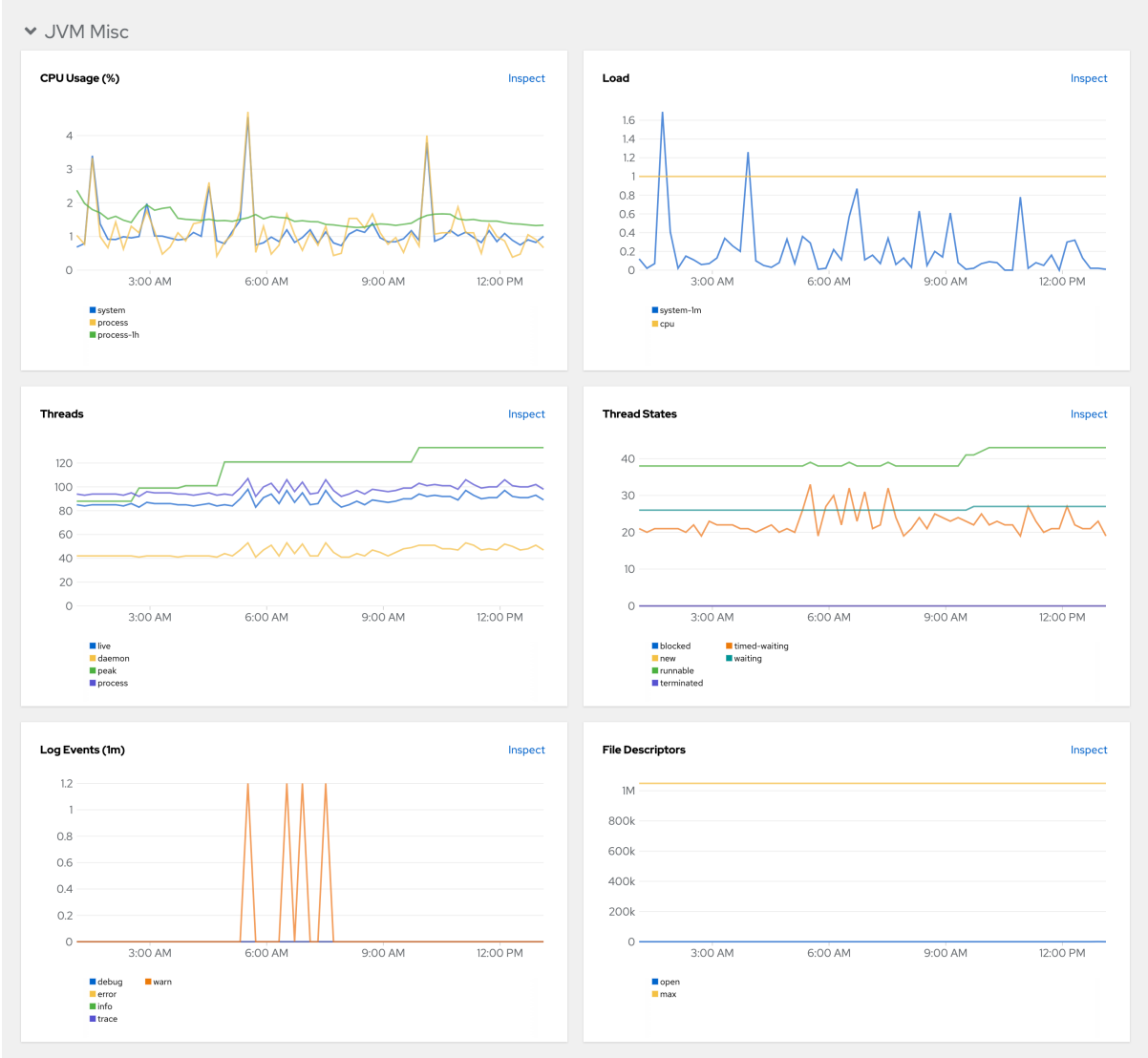


图 3.6. JVM 内存池(heap)

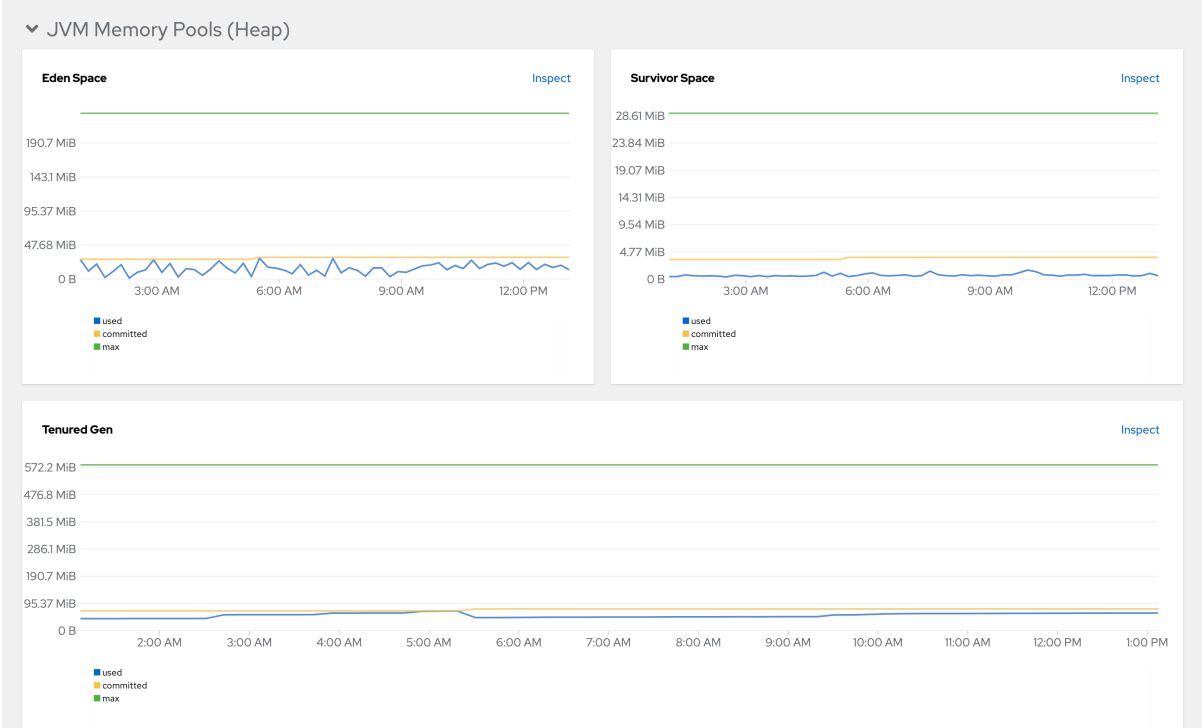


图 3.7. JVM 内存池(Non-Heap)

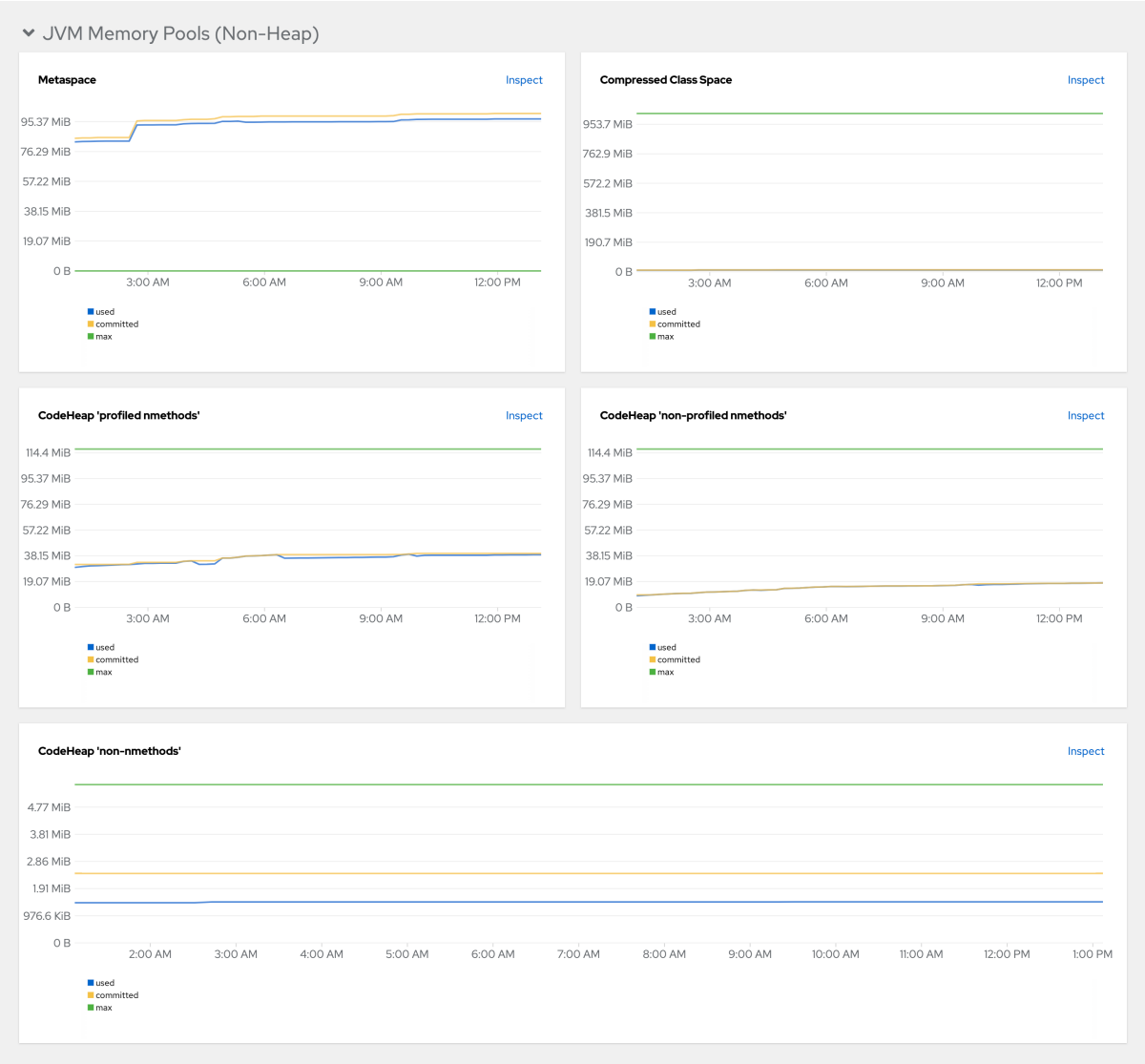


图 3.8. 垃圾回收

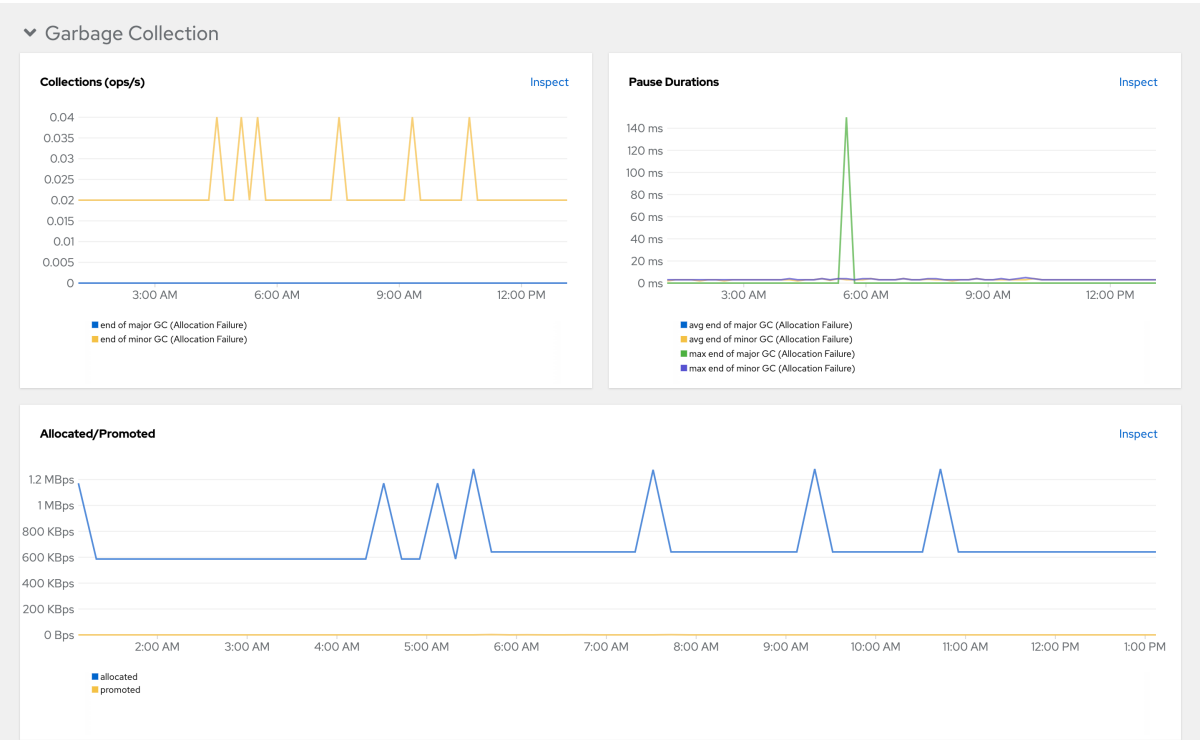


图 3.9. 类加载

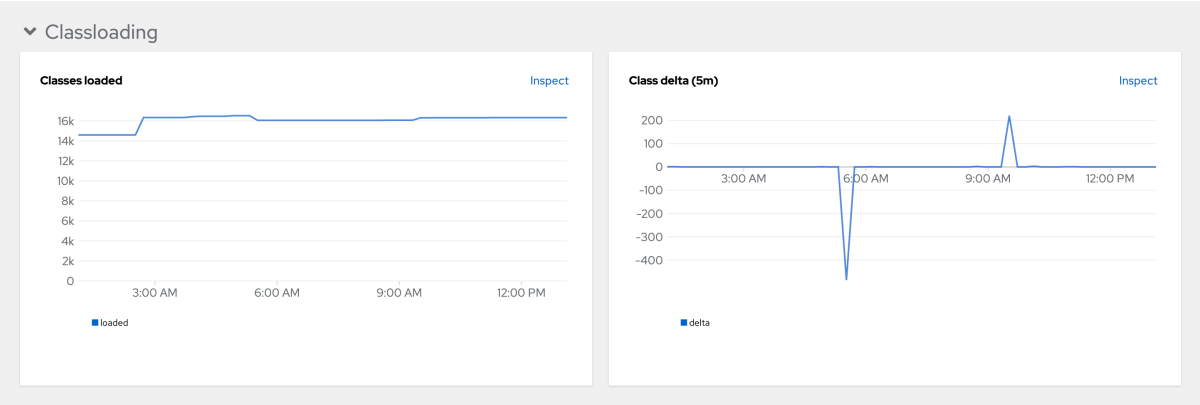
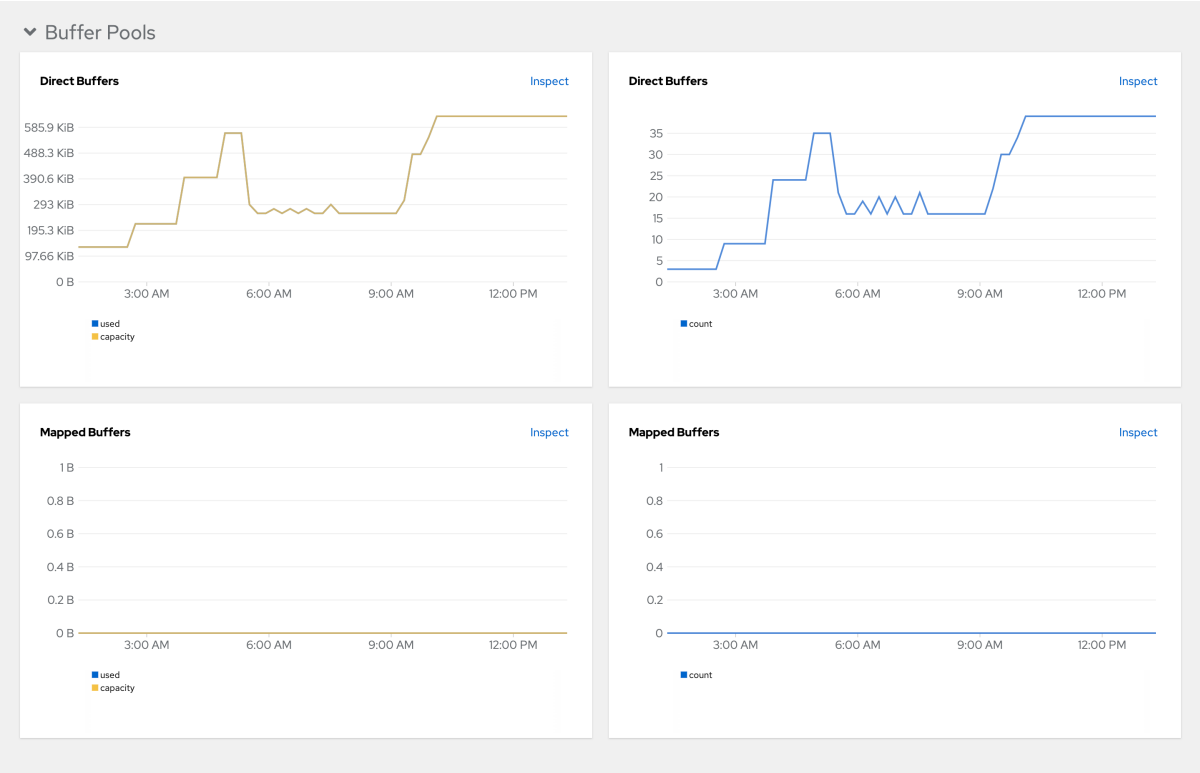


图 3.10. 缓冲池



3.7. 配置网络

- 第 3.7.1 节 “配置网络策略”
- 第 3.7.2 节 “配置 Dev Spaces 主机名”
- 第 3.7.3 节 “将不受信任的 TLS 证书导入到 Dev Spaces”
- 第 3.7.4 节 “添加标签和注解”

3.7.1. 配置网络策略

默认情况下，OpenShift 集群中的所有 Pod 都可以相互通信，即使它们在不同的命名空间中也是如此。在 OpenShift Dev Spaces 中，这使得一个用户项目中的工作区 Pod 可能会向另一个用户项目中的另一个工作区 Pod 发送流量。

为安全起见，可以使用 NetworkPolicy 对象配置多租户隔离，以限制用户项目中的 Pod 的所有传入通信。但是，OpenShift Dev Spaces 项目中的 Pod 必须能够与用户项目中的 Pod 通信。

先决条件

- OpenShift 集群有网络限制，如多租户隔离。

流程

- 将 allow-from-openshift-devspaces NetworkPolicy 应用到每个用户项目。allow-from-openshift-devspaces NetworkPolicy 允许将来自 OpenShift Dev Spaces 命名空间的传入流量到用户项目中的所有 Pod。

例 3.42. allow-from-openshift-devspaces.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-devspaces
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-devspaces  1
  podSelector: {}  2
  policyTypes:
  - Ingress
```

1. OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。
2. 空 podSelector 选择项目中的所有 Pod。

- **OPTIONAL：如果您 使用网络策略配置多租户隔离 时，还必须将 allow-from-openshift-apiserver 和 allow-from-workspaces-namespaces NetworkPolicies 应用到 openshift-devspaces。allow-from-openshift-apiserver NetworkPolicy 允许将 openshift-apiserver 命名空间的传入流量到 devworkspace-webhook-server 启用 Webhook。allow-from-workspaces-namespaces NetworkPolicy 允许每个用户项目的传入流量到 che-gateway pod。**

例 3.43. allow-from-openshift-apiserver.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-apiserver
  namespace: openshift-devspaces 1
spec:
  podSelector:
    matchLabels:
      app.kubernetes.io/name: devworkspace-webhook-server 2
  ingress:
    - from:
        - podSelector: {}
          namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: openshift-apiserver
  policyTypes:
    - Ingress
```

1 OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

2 podSelector 仅选择 devworkspace-webhook-server pod

例 3.44. allow-from-workspaces-namespaces.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-workspaces-namespaces
  namespace: openshift-devspaces 1
spec:
  podSelector:
    matchLabels:
      app.kubernetes.io/component: che-gateway 2
  ingress:
    - from:
        - podSelector: {}
```

```
      namespaceSelector:
        matchLabels:
          app.kubernetes.io/component: workspaces-namespace
  policyTypes:
    - Ingress
```

1

OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

2

podSelector 只选择 che-gateway pod

其他资源

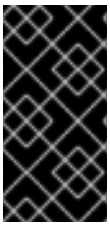
- 第 3.2 节 “配置项目”
- 网络隔离
- 使用网络策略配置多租户隔离

3.7.2. 配置 Dev Spaces 主机名

此流程描述了如何将 OpenShift Dev Spaces 配置为使用自定义主机名。

先决条件

- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。
- 生成证书和私钥文件。



重要

要生成私钥和证书的对，必须使用相同的证书颁发机构(CA)作为其他 OpenShift Dev Spaces 主机。

重要

请求 DNS 供应商将自定义主机名指向集群入口。

流程

1. 为 **OpenShift Dev Spaces** 预先创建项目：

```
$ oc create project openshift-devspaces
```

2. 创建 **TLS secret**：

```
$ oc create secret TLS <tls_secret_name> \ 1
--key <key_file> \ 2
--cert <cert_file> \ 3
-n openshift-devspaces
```

1. **TLS secret** 名称
2. 具有私钥的文件
3. 包含证书的文件

3. 在 **secret** 中添加所需的标签：

```
$ oc label secret <tls_secret_name> \ 1
app.kubernetes.io/part-of=che.eclipse.org -n openshift-devspaces
```

1. **TLS secret** 名称

4. 配置 **CheCluster** 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  networking:
    hostname: <hostname>  (1)
    tlsSecretName: <secret>  (2)
```

(1) 自定义 Red Hat OpenShift Dev Spaces 服务器主机名

(2) TLS secret 名称

5. 如果 OpenShift Dev Spaces 已部署，请等待所有 OpenShift Dev Spaces 组件的推出完成。

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.7.3. 将不受信任的 TLS 证书导入到 Dev Spaces

OpenShift Dev Spaces 组件与外部服务通信通过 TLS 加密。它们需要由可信证书颁发机构(CA)签名的 TLS 证书。因此，您必须导入到 OpenShift Dev Spaces 中，所有不受信任的 CA 链供外部服务使用，例如：

- 代理
- 身份供应商(OIDC)
- 源代码存储库提供程序(Git)

OpenShift Dev Spaces 在 OpenShift Dev Spaces 项目中使用标记的配置映射作为 TLS 证书的源。

配置映射可以具有任意数量的密钥，每个密钥都有随机数量的证书。

注意

当 OpenShift 集群包含通过 cluster- wide-proxy 配置 添加的集群范围可信 CA 证书时，OpenShift Dev Spaces Operator 会检测到它们，并使用 config.openshift.io/inject-trusted-cabundle="true" 标签自动将它们注入到配置映射中。基于此注解，OpenShift 会在配置映射的 ca-bundle.crt 键内自动注入集群范围的可信 CA 证书。

先决条件

- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。
- openshift-devspaces 项目存在。
- 要导入的每个 CA 链：以 PEM 格式的 root CA 和中间证书，使用 ca-cert-for-devspaces-*<count>*.pem 文件。

流程

1. 将要导入的所有 CA 链 PEM 文件串联到 custom-ca-certificates.pem 文件中，并删除与 Java 信任存储不兼容的返回字符。

```
$ cat ca-cert-for-devspaces-*.pem | tr -d '\r' > custom-ca-certificates.pem
```

2. 使用所需的 TLS 证书创建 custom-ca-certificates 配置映射：

```
$ oc create configmap custom-ca-certificates \
    --from-file=custom-ca-certificates.pem \
    --namespace=openshift-devspaces
```

3. 标记 custom-ca-certificates 配置映射：

```
$ oc label configmap custom-ca-certificates \
    app.kubernetes.io/component=ca-bundle \
    app.kubernetes.io/part-of=che.eclipse.org \
    --namespace=openshift-devspaces
```

4. 如果 OpenShift Dev Spaces 之前尚未部署，则部署 OpenShift Dev Spaces。否则，请等待 OpenShift Dev Spaces 组件推出完成。

5. 重启运行工作区以使更改生效。

验证步骤

1. 验证配置映射是否包含您的自定义 CA 证书。此命令以 PEM 格式返回您的自定义 CA 证书：

```
$ oc get configmap \
    --namespace=openshift-devspaces \
    --output='jsonpath={.items[0:].data.custom-ca-certificates\.pem}' \
    --selector=app.kubernetes.io/component=ca-bundle,app.kubernetes.io/part-
of=che.eclipse.org
```

2. 验证 OpenShift Dev Spaces pod 是否包含挂载 ca-certs-merged 配置映射的卷：

```
$ oc get pod \
    --selector=app.kubernetes.io/component=devspaces \
    --output='jsonpath={.items[0].spec.volumes[0:].configMap.name}' \
    --namespace=openshift-devspaces \
    | grep ca-certs-merged
```

3. 验证 OpenShift Dev Spaces 服务器容器是否具有自定义 CA 证书。此命令以 PEM 格式返回您的自定义 CA 证书：

```
$ oc exec -t deploy/devspaces \
    --namespace=openshift-devspaces \
    -- cat /public-certs/custom-ca-certificates.pem
```

4. 在 OpenShift Dev Spaces 服务器日志中验证导入的证书计数是否不是 null：

```
$ oc logs deploy/devspaces --namespace=openshift-devspaces \
    | grep custom-ca-certificates.pem
```

5. 列出证书的 SHA256 指纹：

```
$ for certificate in ca-cert*.pem ;
  do openssl x509 -in $certificate -digest -sha256 -fingerprint -noout | cut -d= -f2;
  done
```

6. 验证 **OpenShift Dev Spaces 服务器 Java 信任存储是否包含具有相同指纹的证书：**

```
$ oc exec -t deploy/devspaces --namespace=openshift-devspaces -- \
    keytool -list -keystore /home/user/cacerts \
    | grep --after-context=1 custom-ca-certificates.pem
```

7. 启动**一个工作区，获取创建它的项目名称：< *workspace_namespace*& gt;，并等待工作区启动。**

8. 验证 **che-trusted-ca-certs 配置映射是否包含您的自定义 CA 证书。此命令以 PEM 格式返回您的自定义 CA 证书：**

```
$ oc get configmap che-trusted-ca-certs \
    --namespace=<workspace_namespace> \
    --output='jsonpath={.data.custom-ca-certificates\.custom-ca-certificates\.pem}'
```

9. 验证**工作区 pod 是否挂载 che-trusted-ca-certs 配置映射：**

```
$ oc get pod \
    --namespace=<workspace_namespace> \
    --selector='controller.devfile.io/devworkspace_name=<workspace_name>' \
    --output='jsonpath={.items[0:].spec.volumes[0:].configMap.name}' \
    | grep che-trusted-ca-certs
```

10. 验证 **universal-developer-image 容器（或工作区 devfile 中定义的容器）是否挂载了 che-trusted-ca-certs 卷：**

```
$ oc get pod \
    --namespace=<workspace_namespace> \
    --selector='controller.devfile.io/devworkspace_name=<workspace_name>' \
    --output='jsonpath={.items[0:].spec.containers[0:]}' \
    | jq 'select (.volumeMounts[].name == "che-trusted-ca-certs") | .name'
```

11. 获取**工作区 pod 名称 < *workspace_pod_name*>:**

```
$ oc get pod \
    --namespace=<workspace_namespace> \
    --selector='controller.devfile.io/devworkspace_name=<workspace_name>' \
    --output='jsonpath={.items[0:].metadata.name}' \
```

12. 验证工作区容器是否具有自定义 CA 证书。此命令以 PEM 格式返回您的自定义 CA 证书：

```
$ oc exec <workspace_pod_name> \
    --namespace=<workspace_namespace> \
    -- cat /public-certs/custom-ca-certificates.custom-ca-certificates.pem
```

其他资源

- 第 3.4.3 节 “带有自签名证书的 Git”.

3.7.4. 添加标签和注解

3.7.4.1. 配置 OpenShift 路由以使用路由器分片

您可以配置标签、注解和域，以用于 路由器分片。

先决条件

- 具有 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 OpenShift CLI 入门。
- DSC.请参阅： 第 1.2 节 “安装 dsc 管理工具”。

流程

- 配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  networking:
    labels: <labels> 1
    domain: <domain> 2
    annotations: <annotations> 3
```

1. 目标入口控制器用来过滤到服务的 Routes 集合的无结构键值映射。
2. 目标入口控制器服务的 DNS 名称。

3

与资源存储的无结构键值映射。

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.8. 配置存储

警告

OpenShift Dev Spaces 不支持网络文件系统(NFS)协议。

- 第 3.8.1 节 “配置存储类”
- 第 3.8.2 节 “配置存储策略”
- 第 3.8.3 节 “配置存储大小”

3.8.1. 配置存储类

要将 OpenShift Dev Spaces 配置为使用配置的基础架构存储，请使用存储类安装 OpenShift Dev Spaces。当您要绑定非默认置备程序提供的持久性卷时，这特别有用。

OpenShift Dev Spaces 有一个组件，需要持久性卷来存储数据：

- OpenShift Dev Spaces 工作区。OpenShift Dev Spaces 工作区使用卷存储源代码，如 /projects 卷。

注意

只有在工作区不是临时时，OpenShift Dev Spaces 工作区源代码才会存储在持久性卷中。

持久性卷声明事实：

- OpenShift Dev Spaces 不会在基础架构中创建持久性卷。
- OpenShift Dev Spaces 使用持久性卷声明(PVC)来挂载持久性卷。
- 第 1.3.1.2 节 “dev Workspace operator” 创建持久性卷声明。

在 OpenShift Dev Spaces 配置中定义存储类名称，以使用 OpenShift Dev Spaces PVC 中的存储类功能。

流程

使用 CheCluster 自定义资源定义来定义存储类：

1. 定义存储类名称：配置 CheCluster 自定义资源，并安装 OpenShift Dev Spaces。请参阅第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”。

```
spec:
  devEnvironments:
    storage:
      perUserStrategyPvcConfig:
        claimSize: <claim_size> (1)
        storageClass: <storage_class_name> (2)
      perWorkspaceStrategyPvcConfig:
        claimSize: <claim_size> (3)
        storageClass: <storage_class_name> (4)
      pvcStrategy: <pvc_strategy> (5)
```

(1) (3) 持久性卷声明大小。

2 4

持久性卷声明的存储类。当忽略或留空时，会使用默认存储类。

5

持久性卷声明策略。支持的策略有：每个用户（一个卷中的所有工作区持久性卷声明）、per-workspace （每个工作区都有自己的独立持久性卷声明）和 ephemeral （非持久性存储，当工作区停止时，本地更改将会丢失。）

其他资源

- 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”
- 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”

3.8.2. 配置存储策略

OpenShift Dev Spaces 可以配置为通过选择存储策略为工作区提供持久性或非持久性存储。默认情况下，所选存储策略将应用到所有新创建的工作区。用户可以通过 URL 参数在 devfile 中为其工作区选择非默认存储策略。

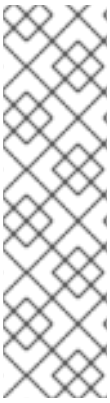
可用的存储策略：

- 每个用户 ：对用户创建的所有工作区使用一个 PVC。
- per-workspace ：每个工作区都给出自己的 PVC。
- 临时: 非持久性存储；当工作空间停止时，任何本地更改都将丢失。

OpenShift Dev Spaces 中使用的默认存储策略是 每个用户。

流程

1. 将 Che Cluster Custom Resource 中的 pvcStrategy 字段设置为 per-user、per-workspace 或 ephemeral。



注意

- 您可在安装时设置此字段。请参阅 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”。
- 您可以在命令行中更新此字段。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  devEnvironments:
    storage:
      pvc:
        pvcStrategy: 'per-user' (1)
```

(1) 可用的存储策略是 每个用户、per- workspace 和 ephemeral。

3.8.3. 配置存储大小

您可以使用 per-user 或 per-workspace 存储策略配置持久性卷声明(PVC)大小。您必须以 Kubernetes 资源数量的形式指定 CheCluster 自定义资源中的 PVC 大小。有关可用存储策略的详情，请查看 此页面。

默认持久性卷声明大小：

-

```
per-user: 10Gi
```

-

```
per-workspace: 5Gi
```

流程

1. 在 Che Cluster 自定义资源中为所需的存储策略设置适当的 claimSize 字段。

注意

- 您可在安装时设置此字段。请参阅 第 3.1.1 节 “在安装过程中使用 dsc 配置 CheCluster 自定义资源”。
- 您可以在命令行中更新此字段。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。

```
spec:
  devEnvironments:
    storage:
      pvc:
        pvcStrategy: '<strategy_name>' (1)
        perUserStrategyPvcConfig: (2)
          claimSize: <resource_quantity> (3)
        perWorkspaceStrategyPvcConfig: (4)
          claimSize: <resource_quantity> (5)
```

(1) 选择存储策略： per-user 或 per-workspace 或 ephemeral。注： 临时存储 策略不使用持久性存储，因此您无法配置其存储大小或其他 PVC 相关的属性。

(2) (4) 在下一行中指定声明大小，或者省略下一行来设置默认的声明大小值。只有在选择此存储策略时，才会使用指定的声明大小。

(3) (5) 声明大小必须指定为 Kubernetes 资源数量。可用的单位包括： Ei、Pi、Ti、Gi、Mi 和 Ki。

3.9. 配置仪表板

- 第 3.9.1 节 “配置入门示例”
- 第 3.9.2 节 “自定义 OpenShift Eclipse Che ConsoleLink 图标”

3.9.1. 配置入门示例

此流程描述了如何配置 OpenShift Dev Spaces Dashboard 来显示自定义示例。

先决条件

- 具有 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 使用示例配置创建 JSON 文件。该文件必须包含一组对象，其中每个对象代表一个示例。

```
cat > my-samples.json <<EOF
[
  {
    "displayName": "<display_name>", (1)
    "description": "<description>", (2)
    "tags": <tags>, (3)
    "url": "<url>", (4)
    "icon": {
      "base64data": "<base64data>", (5)
      "mediatype": "<mediatype>" (6)
    }
  }
]
EOF
```

(1) 示例的显示名称。

(2) 示例的描述。

(3)

4 包含 devfile 的存储库的 URL。

5 图标的 base64 编码数据。

6 图标的介质类型。例如，image/png。

2. 使用示例配置创建 ConfigMap：

```
oc create configmap getting-started-samples --from-file=my-samples.json -n
openshift-devspaces
```

3. 在 ConfigMap 中添加所需的标签：

```
oc label configmap getting-started-samples app.kubernetes.io/part-of=che.eclipse.org
app.kubernetes.io/component=getting-started-samples -n openshift-devspaces
```

4. 刷新 OpenShift Dev Spaces Dashboard 页面，以查看新的示例。

3.9.2. 自定义 OpenShift Eclipse Che ConsoleLink 图标

此流程描述了如何自定义 Red Hat OpenShift Dev Spaces ConsoleLink 图标。

先决条件

- 具有 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 创建 Secret：

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

```
apiVersion: v1
kind: Secret
metadata:
  name: devspaces-dashboard-customization
  namespace: openshift-devspaces
  annotations:
    che.eclipse.org/mount-as: subpath
    che.eclipse.org/mount-path: /public/dashboard/assets/branding
  labels:
    app.kubernetes.io/component: devspaces-dashboard-secret
    app.kubernetes.io/part-of: che.eclipse.org
data:
  loader.svg: <Base64_encoded_content_of_the_image> (1)
type: Opaque
EOF
```

(1) 禁用行嵌套的 Base64 编码。

2. 等待 devspaces-dashboard 的推出完成。

其他资源

- 在 Web 控制台中创建自定义链接

3.10. 管理身份和授权

本节介绍了管理 Red Hat OpenShift Dev Spaces 的身份和授权的不同方面。

3.10.1. 为 Git 供应商配置 OAuth

您可以在 OpenShift Dev Spaces 和 Git 供应商之间配置 OAuth，允许用户使用远程 Git 存储库：

- 第 3.10.1.1 节 “为 GitHub 配置 OAuth 2.0”
- 第 3.10.1.2 节 “为 GitLab 配置 OAuth 2.0”
- 为 Bitbucket 服务器配置 OAuth 2.0，或为 Bitbucket 云配置 OAuth 2.0

- 为 Bitbucket 服务器配置 OAuth 1.0
- 第 3.10.1.6 节 “为 Microsoft Azure DevOps 服务配置 OAuth 2.0”

3.10.1.1. 为 GitHub 配置 OAuth 2.0

允许用户使用 GitHub 上托管的远程 Git 存储库：

1. 设置 GitHub OAuth 应用(OAuth 2.0)。
2. 应用 GitHub OAuth App Secret。

3.10.1.1.1. 设置 GitHub OAuth 应用程序

使用 OAuth 2.0 设置 GitHub OAuth 应用程序。

先决条件

- 已登陆到 GitHub。

流程

1. 转至 https://github.com/settings/applications/new。
2. 输入以下值：
   a. 应用程序名称：<*application name*>
   b. 主页 URL ：https:// &*lt;openshift_dev_spaces_fqdn>*/
   c. 授权回调 URL ：https:// &*lt;openshift_dev_spaces_fqdn&gt*; /api/oauth/callback

3. 点击 Register application。

4. 点 Generate new client secret。

5. 复制并保存 GitHub OAuth 客户端 ID，以便在应用 GitHub OAuth App Secret 时使用。

6. 复制并保存 GitHub OAuth 客户端 Secret，以便在应用 GitHub OAuth App Secret 时使用。

其他资源

- GitHub 文档：创建 OAuth 应用程序

3.10.1.1.2. 应用 GitHub OAuth 应用程序 Secret

准备并应用 GitHub OAuth App Secret。

先决条件

- 设置 GitHub OAuth 应用程序已完成。
- 设置 GitHub OAuth App 时生成的以下值：
  - GitHub OAuth 客户端 ID
  - GitHub OAuth 客户端 Secret
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 准备 Secret：

```
kind: Secret
apiVersion: v1
metadata:
  name: github-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: github
    che.eclipse.org/scm-server-endpoint: <github_server_url> (2)
    che.eclipse.org/scm-github-disable-subdomain-isolation: 'false' (3)
type: Opaque
stringData:
  id: <GitHub_OAuth_Client_ID> (4)
  secret: <GitHub_OAuth_Client_Secret> (5)
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

(2) 这取决于您的机构使用的 GitHub 产品：在 GitHub.com 或 GitHub Enterprise Cloud 上托管软件仓库时，请省略这一行，或者输入默认的 https://github.com。在 GitHub Enterprise 服务器上托管存储库时，请输入 GitHub Enterprise Server URL。

(3) 如果您使用带有禁用 子域隔离 选项的 GitHub Enterprise 服务器，您必须将注解设置为 true，否则您可以省略注解或将其设置为 false。

(4) GitHub OAuth 客户端 ID。

(5) GitHub OAuth 客户端 Secret。

2. 应用 Secret：

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

3.

3. 在输出中验证是否已创建 Secret。

要为另一个 GitHub 供应商配置 OAuth 2.0，您必须重复上述步骤，并创建具有不同名称的第二个 GitHub OAuth Secret。

3.10.1.2. 为 GitLab 配置 OAuth 2.0

允许用户使用 GitLab 实例托管的远程 Git 存储库：

1. 设置 GitLab 授权应用程序(OAuth 2.0)。
2. 应用 GitLab 授权应用程序 Secret。

3.10.1.2.1. 设置 GitLab 授权应用程序

使用 OAuth 2.0 设置 GitLab 授权应用程序。

先决条件

- 您已登录到 GitLab。

流程

1. 点您的 avatar，再前往 Edit profile → Applications。
2. 输入 OpenShift Dev Spaces 作为 Name。
3. 输入 https://*<openshift_dev_spaces_fqdn>* /api/oauth/callback 作为 Redirect URI。
4. 选中 机密和 过期访问令牌 复选框。

-

5. 在 Scopes 下，选中 api、write_repository 和 openid 复选框。

6. 单击 Save application。

7. 复制并保存 GitLab 应用程序 ID，以便在应用 GitLab-authorized 应用 Secret 时使用。

8. 复制并保存 GitLab 客户端 Secret，以便在应用 GitLab-authorized 应用 Secret 时使用。

其他资源

- GitLab Docs: 授权应用程序

3.10.1.2.2. 应用 GitLab-authorized 应用程序 Secret

准备并应用 GitLab-authorized 应用 Secret。

先决条件

- 设置 GitLab 授权应用程序已完成。
- 设置 GitLab 授权应用程序时生成的以下值：
  - GitLab 应用程序 ID
  - GitLab Client Secret
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 准备 Secret：

```
kind: Secret
apiVersion: v1
metadata:
  name: gitlab-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: gitlab
    che.eclipse.org/scm-server-endpoint: <gitlab_server_url> (2)
type: Opaque
stringData:
  id: <GitLab_Application_ID> (3)
  secret: <GitLab_Client_Secret> (4)
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

(2) GitLab 服务器 URL.将 https://gitlab.com 用于 SAAS 版本。

(3) GitLab 应用 ID.

(4) GitLab 客户端 Secret.

2. 应用 Secret：

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

3. 在输出中验证是否已创建 Secret。

3.10.1.3. 为 Bitbucket 服务器配置 OAuth 2.0

您可以使用 OAuth 2.0 使用户处理托管在 Bitbucket 服务器上的远程 Git 存储库：

1. 在 Bitbucket 服务器上设置 OAuth 2.0 应用程序链接。

2. 为 Bitbucket 服务器应用应用链接 Secret。

3.10.1.3.1. 在 Bitbucket 服务器上设置 OAuth 2.0 应用程序链接

在 Bitbucket 服务器上设置 OAuth 2.0 应用程序链接。

先决条件

- 您已登录到 Bitbucket 服务器。

流程

1. 进入 Administration > Applications > Application links。

2. 选择 Create link。

3. 选择 External application 和 Incoming。

4. 在 Redirect URL 字段中输入 https:// *<openshift_dev_spaces_fqdn>* /api/oauth/callback。

5. 选择 应用权限 中的 Admin - Write 复选框。

6. 点击 Save。

7. 复制并保存在应用 Bitbucket 应用程序链接 Secret 时使用的客户端 ID。

8. 复制并保存 Client secret，以便在应用 Bitbucket 应用程序链接 Secret 时使用。

其他资源

- atlassian 文档：配置传入链接

3.10.1.3.2. 为 Bitbucket 服务器应用 OAuth 2.0 应用程序链接 Secret

为 Bitbucket 服务器准备并应用 OAuth 2.0 应用链接 Secret。

先决条件

- 应用程序链接已在 Bitbucket 服务器上设置。
- 设置 Bitbucket 应用程序链接时生成的以下值：
  - Bitbucket 客户端 ID
  - Bitbucket 客户端 secret
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 准备 Secret：

```
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: <bitbucket_server_url> (2)
type: Opaque
stringData:
  id: <Bitbucket_Client_ID> (3)
  secret: <Bitbucket_Client_Secret> (4)
```

(1)

OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

2

Bitbucket 服务器的 URL。

3

Bitbucket 客户端 ID。

4

Bitbucket 客户端机密。

2. 应用 **Secret**：

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

3. **在输出中验证是否已创建 Secret。**

3.10.1.4. 为 Bitbucket 云配置 OAuth 2.0

您可以启用用户处理 Bitbucket 云托管的远程 Git 存储库：

1. **在 Bitbucket 云中设置 OAuth 使用者(OAuth 2.0)。**

2. 为 **Bitbucket 云应用 OAuth 使用者 Secret。**

3.10.1.4.1. 在 Bitbucket 云中设置 OAuth 使用者

在 Bitbucket 云中为 OAuth 2.0 设置 OAuth 使用者。

先决条件

- 您已登录到 Bitbucket 云。

流程

1. 点您的 avatar，再进入 All workspaces 页面。
2. 选择一个工作区并点它。
3. 进入 Settings → OAuth consumers → Add consumer。
4. 输入 OpenShift Dev Spaces 作为 Name。
5. 输入 https://*<openshift_dev_spaces_fqdn*> /api/oauth/callback 作为 Callback URL。
6. 在 Permissions 下，选中所有 帐户 和存储库复选框，然后单击保存。
7. 扩展添加的消费者，然后复制并保存在应用 Bitbucket OAuth 消费者 Secret 时使用的 Key 值：
8. 复制并保存 Secret 值，以便在应用 Bitbucket OAuth 消费者 Secret 时使用。

其他资源

- Bitbucket Docs ：在 Bitbucket 云上使用 OAuth

3.10.1.4.2. 为 Bitbucket 云应用 OAuth 消费者 Secret

为 Bitbucket 云准备并应用 OAuth 使用者 Secret。

先决条件

- OAuth 使用者在 Bitbucket 云中设置。
- 设置 Bitbucket OAuth 消费者时生成的以下值：
  - Bitbucket OAuth 使用者密钥
  - Bitbucket OAuth 消费者 Secret
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 准备 Secret：

```
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
type: Opaque
stringData:
  id: <Bitbucket_Oauth_Consumer_Key> (2)
  secret: <Bitbucket_Oauth_Consumer_Secret> (3)
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

(2) Bitbucket OAuth 使用者密钥。

(3) Bitbucket OAuth 消费者 Secret。

2. 应用 Secret：

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

3. 在输出中验证是否已创建 Secret。

3.10.1.5. 为 Bitbucket 服务器配置 OAuth 1.0

允许用户使用托管在 Bitbucket 服务器上的远程 Git 存储库：

1. 在 Bitbucket 服务器上设置应用程序链接(OAuth 1.0)。

2. 为 Bitbucket 服务器应用应用链接 Secret。

3.10.1.5.1. 在 Bitbucket 服务器上设置应用程序链接

在 Bitbucket 服务器上为 OAuth 1.0 设置应用程序链接。

先决条件

- 您已登录到 Bitbucket 服务器。
- OpenSSL 安装在您正在使用的操作系统中。

流程

1. 在命令行中，运行命令为后续步骤创建必要的文件，并在应用应用程序链接 Secret 时使用：

```
$ openssl genrsa -out private.pem 2048 && \
openssl pkcs8 -topk8 -inform pem -outform pem -nocrypt -in private.pem -out
privatepkcs8.pem && \
cat privatepkcs8.pem | sed 's/-----BEGIN PRIVATE KEY-----//g' | sed 's/-----END PRIVATE
KEY-----//g' | tr -d '\n' > privatepkcs8-stripped.pem && \
openssl rsa -in private.pem -pubout > public.pub && \
```

```
cat public.pub | sed 's/-----BEGIN PUBLIC KEY-----//g' | sed 's/-----END PUBLIC KEY-----//g'
| tr -d '\n' > public-stripped.pub && \
openssl rand -base64 24 > bitbucket-consumer-key && \
openssl rand -base64 24 > bitbucket-shared-secret
```

2. **前往 Administration → Application Links。**

3. **在 URL 字段中输入 https:// *<openshift_dev_spaces_fqdn>* /，点 Create new link。**

4. **在提供的 Application URL 下，选中 Use this URL 复选框并单击 Continue。**

5. **输入 OpenShift Dev Spaces 作为 Application Name。**

6. **选择 Generic Application 作为 Application Type。**

7. **输入 OpenShift Dev Spaces 作为 Service Provider Name。**

8. **将 bitbucket-consumer-key 文件的内容粘贴到 Consumer 键。**

9. **将 bitbucket-shared-secret 文件的内容粘贴到 Shared secret。**

10. **输入 *<bitbucket_server_url>* /plugins/servlet/oauth/request-token 作为 Request Token URL。**

11. **输入 *<bitbucket_server_url>*/plugins/servlet/oauth/access-token 作为 Access token URL。**

12. **输入 *<bitbucket_server_url>* /plugins/servlet/oauth/authorize 作为 Authorize URL。**

13. **选中 Create incoming 链接 复选框，然后单击 Continue。**

14. 将 bitbucket-consumer-key 文件的内容粘贴到 Consumer Key。

15. 输入 OpenShift Dev Spaces 作为 Consumer 名称。

16. 将 public-stripped.pub 文件的内容粘贴到 公钥 中，然后单击 Continue。

其他资源

- atlassian 文档：指向其他应用程序的链接

3.10.1.5.2. 为 Bitbucket 服务器应用应用程序链接 Secret

为 Bitbucket 服务器准备并应用应用程序链接 Secret。

先决条件

- 应用程序链接已在 Bitbucket 服务器上设置。
- 以下在设置应用程序链接时创建的文件已准备好：
  - privatepkcs8-stripped.pem
  - bitbucket-consumer-key
  - bitbucket-shared-secret
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 准备 Secret：

```
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/component: oauth-scm-configuration
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: <bitbucket_server_url> (2)
type: Opaque
stringData:
  private.key: <Content_of_privatepkcs8-stripped.pem> (3)
  consumer.key: <Content_of_bitbucket-consumer-key> (4)
  shared_secret: <Content_of_bitbucket-shared-secret> (5)
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

(2) Bitbucket 服务器的 URL。

(3) privatepkcs8-stripped.pem 文件的内容。

(4) bitbucket-consumer-key 文件的内容。

(5) bitbucket-shared-secret 文件的内容。

2. 应用 Secret：

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

3. 在输出中验证是否已创建 Secret。

3.10.1.6. 为 Microsoft Azure DevOps 服务配置 OAuth 2.0

允许用户使用托管在 Microsoft Azure Repos 上的远程 Git 存储库：

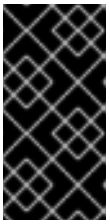
1. 设置 Microsoft Azure DevOps Services OAuth 应用(OAuth 2.0)。
2. 应用 Microsoft Azure DevOps Services OAuth App Secret。

3.10.1.6.1. 设置 Microsoft Azure DevOps Services OAuth 应用程序

使用 OAuth 2.0 设置 Microsoft Azure DevOps Services OAuth 应用程序。

先决条件

- 已登陆到 Microsoft Azure DevOps Services。



重要

为您的机构启用了通过 OAuth 进行第三方应用程序访问。请参阅 为您的机构更改应用程序连接和安全策略。

流程

1. 访问 https://app.vsaex.visualstudio.com/app/register/。
2. 输入以下值：

a. 公司名称 ：OpenShift Dev Spaces

b. 应用程序名称 ：OpenShift Dev Spaces

c. 应用程序网站 ：https:// &*lt;openshift_dev_spaces_fqdn>*/

d. 授权回调 URL ：https:// &*lt;openshift_dev_spaces_fqdn>* /api/oauth/callback

3. 在 Select Authorized scopes 中，选择 Code （读取和写入）。

4. 点 Create application。

5. 复制并保存 App ID，以便在应用 Microsoft Azure DevOps Services OAuth App Secret 时使用。

6. 点 Show 以显示 Client Secret。

7. 复制并保存 Client Secret，以便在应用 Microsoft Azure DevOps Services OAuth App Secret 时使用。

其他资源

- 使用 OAuth 2.0 授权访问 REST API
- 为您的机构更改应用程序连接和安全策略

3.10.1.6.2. 应用 Microsoft Azure DevOps Services OAuth App Secret

准备并应用 Microsoft Azure DevOps Services Secret。

先决条件

- 设置 Microsoft Azure DevOps Services OAuth 应用程序已完成。
- 设置 Microsoft Azure DevOps Services OAuth App 时会生成以下值：
  - 应用程序 ID
  - Client Secret
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 准备 Secret：

```
kind: Secret
apiVersion: v1
metadata:
  name: azure-devops-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: azure-devops
type: Opaque
stringData:
  id: <Microsoft_Azure_DevOps_Services_OAuth_App_ID> (2)
  secret: <Microsoft_Azure_DevOps_Services_OAuth_Client_Secret> (3)
```

(1) OpenShift Dev Spaces 命名空间。默认值为 openshift-devspaces。

(2) Microsoft Azure DevOps 服务 OAuth 应用 ID.

3

Microsoft Azure DevOps Services OAuth 客户端 Secret。

2. 应用 Secret：

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

3. 在输出中验证是否已创建 Secret。

4. 等待 OpenShift Dev Spaces 服务器组件的推出完成。

3.10.2. 为 Dev Spaces 用户配置集群角色

您可以通过向这些用户添加集群角色来授予 OpenShift Dev Spaces 用户更多集群权限。

先决条件

- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1. 定义用户角色名称：

```
$ USER_ROLES=<name> 1
```

1

唯一的资源名称。

2. 找到部署 OpenShift Dev Spaces Operator 的命名空间：

```
$ OPERATOR_NAMESPACE=$(oc get pods -l app.kubernetes.io/component=devspaces-
operator -o jsonpath={".items[0].metadata.namespace"} --all-namespaces)
```

3. **创建所需的角色：**

```
$ kubectl apply -f - <<EOF
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: ${USER_ROLES}
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
rules:
  - verbs:
      - <verbs> (1)
    apiGroups:
      - <apiGroups> (2)
    resources:
      - <resources> (3)
EOF
```

(1) **对于 *<verbs>*，列出适用于此规则中包含的所有 ResourceKinds 和 AttributeRestrictions 的所有 Verbs。您可以使用 * 来代表所有动词。**

(2) **对于 *<apiGroups>*，将包含资源的 APIGroups 命名为。**

(3) **对于 *<resources>*，列出此规则应用到的所有资源。您可以使用 * 来代表所有动词。**

4. **将角色委派给 OpenShift Dev Spaces Operator：**

```
$ kubectl apply -f - <<EOF
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: ${USER_ROLES}
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
subjects:
  - kind: ServiceAccount
    name: devspaces-operator
    namespace: ${OPERATOR_NAMESPACE}
roleRef:
  apiGroup: rbac.authorization.k8s.io
```

```
  kind: ClusterRole
  name: ${USER_ROLES}
EOF
```

5. **配置 OpenShift Dev Spaces Operator，将角色委派给 che 服务帐户：**

```
$ kubectl patch checluster devspaces \
  --patch '{"spec": {"components": {"cheServer": {"clusterRoles": ["'${USER_ROLES}'"]}}}}' \
  --type=merge -n openshift-devspaces
```

6. **配置 OpenShift Dev Spaces 服务器，将角色委派给用户：**

```
$ kubectl patch checluster devspaces \
  --patch '{"spec": {"devEnvironments": {"user": {"clusterRoles": ["'${USER_ROLES}'"]}}}}' \
  --type=merge -n openshift-devspaces
```

7. **等待 OpenShift Dev Spaces 服务器组件的推出完成。**

8. **要求用户退出登录并登录以应用新角色。**

3.10.3. 配置高级授权

您可以确定允许哪些用户和组访问 OpenShift Dev Spaces。

先决条件

- **具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。**

流程

1. **配置 CheCluster 自定义资源。请参阅 第 3.1.2 节 “使用 CLI 配置 CheCluster 自定义资源”。**

```
spec:
  networking:
    auth:
      advancedAuthorization:
        allowUsers:
          - <allow_users> (1)
        allowGroups:
```

```
  - <allow_groups> 2
denyUsers:
  - <deny_users> 3
denyGroups:
  - <deny_groups> 4
```

1

允许访问 Red Hat OpenShift Dev Spaces 的用户列表。

2

允许访问 Red Hat OpenShift Dev Spaces 的用户组列表（仅适用于 OpenShift Container Platform）。

3

拒绝访问 Red Hat OpenShift Dev Spaces 的用户列表。

4

拒绝访问 Red Hat OpenShift Dev Spaces 的用户组列表（仅适用于 OpenShift Container Platform）。

2. 等待 OpenShift Dev Spaces 服务器组件的推出完成。

注意

要允许用户访问 OpenShift Dev Spaces，请将它们添加到 allowUsers 列表中。或者，选择用户所属的组，并将该组添加到 allowGroups 列表中。要拒绝用户访问 OpenShift Dev Spaces，请将它们添加到 denyUsers 列表中。或者，选择用户所属的组，并将该组添加到 denyGroups 列表中。如果用户位于 allow 和 deny 列表中，它们将被拒绝访问 OpenShift Dev Spaces。

如果 allowUsers 和 allowGroups 为空，则允许所有用户访问 OpenShift Dev Spaces，但拒绝 列表中的用户除外。如果 denyUsers 和 denyGroups 为空，则只允许来自 允许列表 的用户访问 OpenShift Dev Spaces。

如果 allow 和 deny 列表都为空，则允许所有用户访问 OpenShift Dev Spaces。

3.10.4. 删除用户数据以遵守 GDPR

您可以删除 OpenShift Container Platform 上的用户数据，以符合 General Data Protection Regulation (GDPR)，它强制个人使用其个人数据擦除。其他 Kubernetes 基础架构的流程可能有所不同。按照用于 Red Hat OpenShift Dev Spaces 安装的供应商管理最佳实践操作。

警告

按如下所示删除用户数据是不可能的！所有删除的数据都将被删除且不可恢复！

先决条件

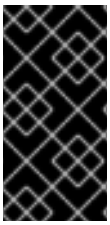
- 具有 OpenShift Container Platform 集群的管理权限的活跃 oc 会话。请参阅 OpenShift CLI 入门。

流程

1. 使用以下命令列出 OpenShift 集群中的所有用户：

```
$ oc get users
```

2. 删除用户条目：



重要

如果用户有任何关联的资源（如项目、角色或服务帐户），则需要先删除这些资源，然后才能删除用户。

```
$ oc delete user <username>
```

其他资源

- 第 6 章 *使用 Dev Spaces 服务器 API*
- 第 3.2.1 节 “配置项目名称”

- 第 8 章 *卸载 Dev Spaces*

3.11. 配置 FUSE-OVERLAYFS

默认情况下，通用基础镜像(UDI)包含 Podman 和 Buildah，可用于在工作区中构建和推送容器镜像。但是，UDI 中的 Podman 和 Buildah 配置为使用不提供写时复制支持的 vfs 存储驱动程序。要更有效地镜像管理，请使用 fuse-overlayfs 存储驱动程序，该驱动程序支持无根环境中的 copy-on-write。

3.11.1. 为 OpenShift 启用 /dev/fuse 的容器访问权限

要使用 fuse-overlayfs，您必须首先让 /dev/fuse 可供工作区容器访问。

注意

OpenShift 版本 4.15 及更新的版本不需要这个过程，因为 /dev/fuse 设备默认可用。请参阅 发行注记。

警告

在 OpenShift 集群中创建 MachineConfig 资源是一个潜在的危险任务，因为您要对集群进行高级的系统级更改。

查看 MachineConfig 文档以了解 更多详细信息和可能的风险。

先决条件

- Butane 工具(但)安装在您正在使用的操作系统中。
- 具有对目标 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 CLI 入门。

流程

1.

根据 OpenShift 集群的类型设置环境变量：单一节点集群，或具有独立 control plane 和 worker 节点的多节点集群。

- 对于单一节点集群，设置：

```
$ NODE_ROLE=master
```

- 对于多节点集群，设置：

```
$ NODE_ROLE=worker
```

2. 为 OpenShift Butane 配置版本设置环境变量。此变量是 OpenShift 集群的主版本和次要版本。例如： 4.12.0、4.13.0 或 4.14.0。

```
$ VERSION=4.12.0
```

3. 创建一个 MachineConfig 资源，在 NODE_ROLE 节点上创建一个名为 99-podman-fuse 的置入 CRI-O 配置文件。此配置文件可以访问某些 pod 的 /dev/fuse 设备。

```
cat << EOF | butane | oc apply -f -
variant: openshift
version: ${VERSION}
metadata:
  labels:
    machineconfiguration.openshift.io/role: ${NODE_ROLE}
  name: 99-podman-dev-fuse-${NODE_ROLE}
storage:
  files:
  - path: /etc/crio/crio.conf.d/99-podman-fuse (1)
    mode: 0644
    overwrite: true
    contents: (2)
      inline: |
        [crio.runtime.workloads.podman-fuse] (3)
        activation_annotation = "io.openshift.podman-fuse" (4)
        allowed_annotations = [
          "io.kubernetes.cri-o.Devices" (5)
        ]
        [crio.runtime]
        allowed_devices = ["/dev/fuse"] (6)
EOF
```

(1)

2

新 drop-in 配置文件的内容。

3

定义 podman-fuse 工作负载。

4

激活 podman-fuse 工作负载设置的 pod 注解。

5

允许处理 podman-fuse 工作负载的注解列表。

6

用户可以使用 io.kubernetes.cri-o.Devices 注解来指定主机上的设备列表。

4. 应用 MachineConfig 资源后，在应用更改时，会临时禁用具有 worker 角色的每个节点调度。查看节点的状态。

```
$ oc get nodes
```

输出示例：

```
NAME                          STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.27.9
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.27.9
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.27.9
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.27.9
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.27.9
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.27.9
```

5. 当所有具有 worker 角色的节点都处于 Ready 状态后，/dev/fuse 将提供给具有以下注解的任何 pod。

```
io.openshift.podman-fuse: ''
io.kubernetes.cri-o.Devices: /dev/fuse
```

验证步骤

1. **获取具有 worker 角色的节点名称：**

```
$ oc get nodes
```

2. **打开到 worker 节点的 oc debug 会话。**

```
$ oc debug node/<nodename>
```

3. 验证名为 **99-podman-fuse 的新 CRI-O 配置文件是否存在。**

```
sh-4.4# stat /host/etc/crio/crio.conf.d/99-podman-fuse
```

3.11.2. 在工作区中使用 fuse-overlayfs 用于 Podman 和 Buildah

用户可以遵循 https://access.redhat.com/documentation/zh-cn/red_hat_openshift_dev_spaces/3.14/html-single/user_guide/index#end-user-guide:using-the-fuse-overlay-storage-driver 来更新现有工作区，以使用 Podman 和 Buildah 的 fuse-overlayfs 存储驱动程序。

第 4 章 管理 IDE 扩展

IDE 使用扩展或插件扩展其功能，管理扩展的机制在 IDE 之间有所不同。

- 第 4.1 节 “Microsoft Visual Studio Code 扩展 - 开源”

4.1. MICROSOFT VISUAL STUDIO CODE 扩展 - 开源

要管理扩展，此 IDE 使用以下 Open VSX registry 实例之一：

- 在 OpenShift Dev Spaces 的 plugin-registry pod 中运行的 Open VSX registry 的嵌入式实例，以支持 air-gapped、offline 和 proxy-restricted 环境。嵌入式 Open VSX 注册表仅包含 open-vsx.org 上发布的扩展的子集。此子集 可自定义。
- 通过互联网访问的公共 open-vsx.org 注册表。
- 单机 Open VSX registry 实例，部署到可从 OpenShift Dev Spaces 工作区 pod 访问的网络中。

默认为 Open VSX 注册表的嵌入式实例。

4.1.1. 选择 Open VSX registry 实例

默认为 Open VSX 注册表的嵌入式实例。

如果默认的 Open VSX registry 实例不是您需要的，您可以选择以下实例之一：

- https://open-vsx.org 上的 Open VSX registry 实例需要访问互联网。
- 单机 Open VSX registry 实例，部署到可从 OpenShift Dev Spaces 工作区 pod 访问的网络中。

流程

- 编辑 CheCluster 自定义资源中的 openVSXURL 值：

```
spec:
 components:
  pluginRegistry:
   openVSXURL: "<url_of_an_open_vsx_registry_instance>" 1
```

1

例如： openVSXURL: "https://open-vsx.org"。

提示

- 要在 plugin-registry pod 中选择嵌入的 Open VSX registry 实例，请使用 openVSXURL: ''。您可以自定义包含的扩展列表。

- 如果可以从组织的集群内部访问其 URL，并将 openVSXURL 指向独立 Open VSX registry 实例的 URL，而不被代理阻止。

4.1.2. 在嵌入式 Open VSX registry 实例中添加或删除扩展

您可以在嵌入式 Open VSX registry 实例中添加或删除扩展。这将生成可在您组织的工作区中使用的 Open VSX 注册表的自定义构建。

提示

要在 OpenShift Dev Spaces 更新后获取最新的安全修复，请基于 latest 标签或 SHA 重建容器。

流程

1. 获取每个所选扩展的发布程序和扩展名称：

   a. 在 Open VSX 注册表网站上 找到扩展，再复制扩展列表页面的 URL。

   b. 从复制的 URL 中提取 *<publisher>* 和名称 *<extension>* ：

   ```
   https://www.open-vsx.org/extension/<publisher>/<extension>
   ```

   提示

   如果扩展仅可从 Microsoft Visual Studio Marketplace 获得，但没有 Open VSX，您可以要求扩展发布者也根据 这些说明 在 open-vsx.org 上发布它，可能使用此 GitHub 操作。

   如果扩展发布者不可用或不希望将扩展发布到 open-vsx.org，且没有与扩展相对应的 Open VSX，考虑使用向 Open VSX 团队报告问题。

2. 下载或分叉并克隆 插件 registry 存储库。

3. 签出与 OpenShift Dev Spaces 版本对应的分支：

   ```
   git checkout devspaces-$PRODUCT_VERSION-rhel-8
   ```

4. 对于需要添加或删除的每个扩展，请编辑 openvsx-sync.json 文件 ：

   - 要添加扩展，请将发布程序和扩展名称添加到 openvsx-sync.json 文件中。
   - 要删除扩展，请从 openvsx-sync.json 文件中删除发布程序和扩展名称。
   - 使用以下 JSON 语法：

```
{
  "id": "<publisher>.<extension>"
}
```

提示

- open-vsx.org 的最新扩展版本是默认的。或者，您可以在新 行中添加 "*<extension_version>* version": " " 以指定版本。

- 如果您的源扩展或只为机构内部使用的扩展开发，您可以使用自定义插件 registry 容器访问的 URL 直接从 .vsix 文件添加扩展：

```
{
  "id": "<publisher>.<extension>",
  "download": "<url_to_download_vsix_file>",
  "version": "<extension_version>"
}
```

- 在使用其资源前，请阅读 Microsoft Visual Studio Marketplace 的 使用条款。

5. 构建插件 registry 容器镜像，并将其发布到 quay.io 等容器注册表：

   a.

```
$ ./build.sh -o <username> -r quay.io -t custom
```

   b.

```
$ podman push quay.io/<username/plugin_registry:custom>
```

6. 编辑机构的集群中的 CheCluster 自定义资源以指向镜像（例如，在 quay.io上），并保存更改：

```
spec:
 components:
  pluginRegistry:
   deployment:
```

```
  containers:
    - image: quay.io/<username/plugin_registry:custom>
openVSXURL: ''
```

验证

1. 检查 plugin-registry pod 是否已重启并在运行。

2. 重启工作区，并检查工作区 IDE 的 Extensions 视图中的可用扩展。

4.2. 为 MICROSOFT VISUAL STUDIO CODE 配置可信扩展

您可以使用 Microsoft Visual Studio Code 的 product.json 文件中的 trustedExtensionAuthAccess 字段指定哪些扩展可被信任来访问身份验证令牌。

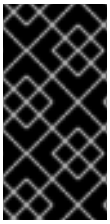
```
"trustedExtensionAuthAccess": [
 "<publisher1>.<extension1>",
 "<publisher2>.<extension2>"
]
```

当您有需要访问 GitHub、Microsoft 或任何需要 OAuth 的其他服务等服务的扩展时，这特别有用。通过在此字段中添加扩展 ID，您要授予它们访问这些令牌的权限。

您可以在 devfile 或 ConfigMap 中定义变量。选择最适合您的需要的选项。使用 ConfigMap 时，变量将在所有工作区上传播，您不需要将变量添加到您要使用的每个 devfile 中。

警告

请谨慎使用 trustedExtensionAuthAccess 字段，因为它可能会在误用时造成安全风险。仅授予可信扩展的访问权限。



流程

由于 Microsoft Visual Studio Code 编辑器捆绑在 che-code 镜像中，因此只能在工作空间启动时更改 product.json 文件。

1. 定义 *VSCODE_TRUSTED_EXTENSIONS* 环境变量。在 devfile.yaml 中定义变量或使用变量挂载 ConfigMap 之间进行选择。

   a. 在 devfile.yaml 中定义 *VSCODE_TRUSTED_EXTENSIONS* 环境变量：

```
env:
  - name: VSCODE_TRUSTED_EXTENSIONS
    value: "<publisher1>.<extension1>,<publisher2>.<extension2>"
```

   b. 使用 *VSCODE_TRUSTED_EXTENSIONS* 环境变量挂载 ConfigMap：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: trusted-extensions
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
  annotations:
    controller.devfile.io/mount-as: env
data:
  VSCODE_TRUSTED_EXTENSIONS: '<publisher1>.<extension1>,<publisher2>.
<extension2>'
```

验证

- 变量的值将在工作区启动时解析，对应的 trustedExtensionAuthAccess 部分将添加到 product.json 中。

第 5 章 配置 VISUAL STUDIO CODE - 开源("代码 - OSS")

了解如何配置 Visual Studio Code - Open Source ("Code - OSS")。

- 第 5.1 节 “配置单一和多根工作区”

5.1. 配置单一和多根工作区

使用多根工作区功能，您可以处理同一工作区中的多个项目文件夹。当您一次性处理多个相关项目，如产品文档和产品代码存储库时，这非常有用。

提示

请参阅什么是 VS Code "workspace" 以更好地了解和编写工作区文件。

注意

工作区默认设置为在多 root 模式中打开。

启动工作区后，会生成 /projects/.code-workspace 工作区文件。工作区文件将包含 devfile 描述的所有项目。

```
{
 "folders": [
  {
   "name": "project-1",
   "path": "/projects/project-1"
  },
  {
   "name": "project-2",
   "path": "/projects/project-2"
  }
 ]
}
```

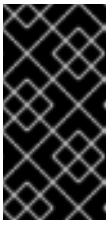
如果工作区文件已存在，则会更新它，所有缺少的项目都将从 devfile 中获取。如果您从 devfile 中删除项目，它将保留在工作区文件中。

您可以更改默认行为，并提供自己的工作区文件或切换到单根工作区。

流程

- 提供您自己的工作区文件。
  - 将名为 .code-workspace 的工作区文件放在存储库的根目录中。创建工作区后， Visual Studio Code - Open Source ("Code - OSS")将使用工作区文件，因为它是.

```
{
 "folders": [
  {
   "name": "project-name",
   "path": "."
  }
 ]
}
```



重要

创建工作区文件时请小心。如果出现错误，将打开一个空的 Visual Studio Code - Open Source ("Code - OSS")。

重要

如果您有多个项目，则工作区文件将从第一个项目获取。如果第一个项目中不存在工作区文件，则会创建一个新工作区，并放在 /projects 目录中。

- 指定备用工作区文件。
  - 在 devfile 中定义 *VSCODE_DEFAULT_WORKSPACE* 环境变量，并指定到工作区文件的正确位置。

```
env:
  - name: VSCODE_DEFAULT_WORKSPACE
    value: "/projects/project-name/workspace-file"
```

- 以单 root 模式打开工作区。

- 定义 *VSCODE_DEFAULT_WORKSPACE* 环境变量并将其设置为 **root**。

```
env:
  - name: VSCODE_DEFAULT_WORKSPACE
    value: "/"
```

第 6 章 使用 DEV SPACES 服务器 API

要管理 OpenShift Dev Spaces 服务器工作负载，请使用 Swagger Web 用户界面来浏览 OpenShift Dev Spaces 服务器 API。

流程

- 进入 **Swagger API web 用户界面： https:// <*openshift_dev_spaces_fqdn*> /swagger。**

其他资源

- Swagger

第 7 章 升级 DEV SPACES

本章论述了如何从 CodeReady Workspaces 3.1 升级到 OpenShift Dev Spaces 3.14。

7.1. 升级 CHECTL 管理工具

这部分论述了如何升级 dsc 管理工具。

流程

- 第 1.2 节 “安装 dsc 管理工具”.

7.2. 指定更新批准策略

Red Hat OpenShift Dev Spaces Operator 支持两个升级策略：

自动

当 Operator 可用时，Operator 会安装新的更新。

Manual（手动）

在开始安装前，需要手动批准新的更新。

您可以使用 OpenShift Web 控制台为 Red Hat OpenShift Dev Spaces Operator 指定更新批准策略。

先决条件

- 集群管理员的 OpenShift Web 控制台会话。请参阅 访问 Web 控制台。
- 使用红帽生态系统目录安装的 OpenShift Dev Spaces 实例。

流程

1. 在 OpenShift Web 控制台中，进入到 Operators → Installed Operators。

2. 在安装的 Operator 列表中，点 Red Hat OpenShift Dev Spaces。

3. 进入 Subscription 选项卡。

4. 将 Update 批准策略 配置为 Automatic 或 Manual。

其他资源

- 更改 Operator 的更新频道

7.3. 使用 OPENSHIFT WEB 控制台升级 DEV SPACES

您可以使用 OpenShift web 控制台中的 Red Hat OpenShift Dev Spaces Operator 从 Red Hat OpenShift Dev Spaces Operator 手动批准升级。

先决条件

- 集群管理员的 OpenShift Web 控制台会话。请参阅 访问 Web 控制台。

- 使用红帽生态系统目录安装的 OpenShift Dev Spaces 实例。

- 订阅中的批准策略为 Manual。请参阅 第 7.2 节 “指定更新批准策略”。

流程

- 手动批准待处理的 Red Hat OpenShift Dev Spaces Operator 升级。请参阅 手动批准待处理的 Operator 升级。

验证步骤

1. 进入到 OpenShift Dev Spaces 实例。

2. 3.14 版本号在页面的底部可见。

其他资源

- 手动批准待处理的 Operator 升级

7.4. 使用 CLI 管理工具升级 DEV SPACES

本节描述了如何使用 CLI 管理工具从以前的次要版本升级。

先决条件

- OpenShift 上的管理帐户。
- 一个以前的 CodeReady Workspaces 版本的实例，使用同一 OpenShift 实例中的 CLI 管理工具安装在 openshift-devspaces OpenShift 项目中。
- 用于 OpenShift Dev Spaces 版本 3.14 的 DSC。请参阅： 第 1.2 节 “安装 dsc 管理工具”。

流程

1. 保存更改并推送回所有正在运行的 CodeReady Workspaces 3.1 工作区的 Git 存储库。
2. 关闭 CodeReady Workspaces 3.1 实例中的所有工作区。
3. 升级 OpenShift Dev Spaces：

```
$ dsc server:update -n openshift-devspaces
```

注意

对于缓慢的系统或互联网连接，请添加 --k8spodwaittimeout=1800000 标志选项，将 Pod 超时周期扩展到 1800000 ms 或更长时间。

验证步骤

1. 进入到 OpenShift Dev Spaces 实例。

2. 3.14 版本号在页面的底部可见。

7.5. 在受限环境中升级 DEV SPACES

本节论述了如何使用受限环境中的 CLI 管理工具升级 Red Hat OpenShift Dev Spaces 并执行次要版本更新。

先决条件

- OpenShift Dev Spaces 实例是在 OpenShift 上使用 openshift-devspaces 项目中的 dsc --installer operator 方法安装的。请参阅 第 2.1.4 节 “在受限环境中安装 Dev Spaces”。
- OpenShift 集群至少有 64 GB 磁盘空间。
- OpenShift 集群已准备好在受限网络上运行，OpenShift control plane 可以访问互联网。请参阅 关于断开连接的安装镜像，以及 在受限网络中使用 Operator Lifecycle Manager。
- 具有 OpenShift 集群的管理权限的活跃 oc 会话。请参阅 OpenShift CLI 入门。
- 一个到 registry.redhat.io 红帽生态系统目录的活跃的 oc registry 会话。请参阅： Red Hat Container Registry 身份验证。
- opm.请参阅安装 opm CLI。
- jq.请参阅 下载 jq。
- Podman.请参阅 Podman 安装说明。
- Skopeo 版本 1.6 或更高版本。请参阅 安装 Skopeo。

- 一个活跃的 skopeo 会话，用于管理对私有 Docker 注册表的访问。向 registry 进行身份验证，并为断开连接的安装 mirror 镜像。

- 用于 OpenShift Dev Spaces 版本 3.14 的 DSC。请参阅 第 1.2 节 “安装 dsc 管理工具”。

流程

1. 下载并执行镜像脚本，以安装自定义 Operator 目录并镜像相关的镜像： prepare-restricted-environment.sh。

```
$ bash prepare-restricted-environment.sh \
  --devworkspace_operator_index registry.redhat.io/redhat/redhat-operator-index:v4.16\
  --devworkspace_operator_version "v0.28.0" \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.16" \
  --prod_operator_package_name "devspaces" \
  --prod_operator_bundle_name "devspacesoperator" \
  --prod_operator_version "v3.14.0" \
  --my_registry "<my_registry>" (1)
```

(1) 镜像将要镜像的私有 Docker registry

2. 在 CodeReady Workspaces 3.1 实例中运行的所有工作区中，保存更改并将其推送回 Git 存储库。

3. 停止 CodeReady Workspaces 3.1 实例中的所有工作区。

4. 运行以下命令：

```
$ dsc server:update --che-operator-image="$TAG" -n openshift-devspaces --
k8spodwaittimeout=1800000
```

验证步骤

1. 进入到 OpenShift Dev Spaces 实例。

2. 3.14 版本号在页面的底部可见。

其他资源

- 红帽提供的 Operator 目录
- 管理自定义目录

7.6. 在 OPENSHIFT 上修复 DEV WORKSPACE OPERATOR

在某些情况下，如 OLM 重启或集群升级，OpenShift Dev Spaces 的 Dev Spaces Operator 可能会自动安装 Dev Workspace Operator，即使在集群中已存在。在这种情况下，您可以修复 OpenShift 上的 Dev Workspace Operator，如下所示：

先决条件

- 一个活跃的 oc 会话，作为集群管理员到目标 OpenShift 集群。请参阅 CLI 入门。
- 在 OpenShift Web 控制台的 Installed Operators 页面中，您会看到 Dev Workspace Operator 的多个条目，或看到一个处于 Replacing 和 Pending 循环中的一个条目。

流程

1. 删除包含故障 pod 的 devworkspace-controller 命名空间。
2. 通过将转换策略设置为 None 并删除整个 webhook 部分，更新 DevWorkspace 和 DevWorkspaceTemplate 自定义资源定义(CRD)：

```
spec:
  ...
  conversion:
    strategy: None
status:
...
```

提示

您可以通过在 Administration → CustomResourceDefinitions 中搜索 DevWorkspace 和 DevWorkspaceTemplate CRD，在 OpenShift Web 控制台的 Administrator 视角中找到并编辑 DevWorkspace 和 DevWorkspaceTemplate CRD。

注意

DevWorkspaceOperatorConfig 和 DevWorkspaceRouting CRD 默认将转换策略设置为 None。

3. **删除 Dev Workspace Operator 订阅：**

```
$ oc delete sub devworkspace-operator \
-n openshift-operators (1)
```

(1) **安装 Dev Workspace Operator 的 openshift-operators 或 OpenShift 项目。**

4. **获取 < *devworkspace_operator.vX.Y.Z* > 格式的 Dev Workspace Operator CSV：**

```
$ oc get csv | grep devworkspace
```

5. **删除每个 Dev Workspace Operator CSV：**

```
$ oc delete csv <devworkspace_operator.vX.Y.Z> \
-n openshift-operators (1)
```

(1) **安装 Dev Workspace Operator 的 openshift-operators 或 OpenShift 项目。**

6. **重新创建 Dev Workspace Operator 订阅：**

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
metadata:
  name: devworkspace-operator
  namespace: openshift-operators
spec:
  channel: fast
  name: devworkspace-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic (1)
  startingCSV: devworkspace-operator.v0.28.0
EOF
```

(1)

Automatic 或 Manual。

重要

对于 installPlanApproval: Manual，在 OpenShift Web 控制台的 Administrator 视角中，进入 Operators → Installed Operators，为 Dev Workspace Operator 选择以下内容：Upgrade available → Preview InstallPlan → Approve。

7. **在 OpenShift Web 控制台的 Administrator 视角中，进入 Operators → Installed Operators 并验证 Dev Workspace Operator 的 Succeeded 状态。**

第 8 章 卸载 DEV SPACES

警告

卸载 OpenShift Dev Spaces 删除所有 OpenShift Dev Spaces 相关的用户数据！

使用 oc 卸载 OpenShift Dev Spaces 实例。

先决条件

- DSC.请参阅： 第 1.2 节 “安装 dsc 管理工具”。

流程

- 删除 OpenShift Dev Spaces 实例：

```
$ dsc server:delete
```

提示

--delete-namespace 选项删除 OpenShift Dev Spaces 命名空间。

--delete-all 选项会删除 Dev Workspace Operator 和相关资源。