云容器引擎
Autopilot
最佳实践

文档版本 01

发布日期 2024-11-21

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 在 CCE Autopilot 集群中部署并使用 Jenkins

1.1 CCE Autopilot 集群部署并使用 Jenkins 方案概述

Jenkins是一个开源的自动化服务器，广泛应用于持续集成（CI）和持续交付/部署（CD）。当您的代码库发生变更时，Jenkins可以帮助您自动构建、测试和部署应用程序，提高开发效率和产品质量。Jenkins可以在多种环境中进行部署，具体对比请参见表1-1。CCE Autopilot集群是云容器引擎服务推出的Serverless版集群，提供免运维的容器服务，您无需购买和管理节点即可部署应用，降低运维成本，同时提升应用的可靠性和扩展性。通过在CCE Autopilot集群中部署Jenkins，可以利用集群的自动化资源管理和按需付费优势，避免不必要的资源浪费，并且可以根据实际需求动态调整计算资源。此外，Autopilot集群还提供了更高的可用性和容错性，确保Jenkins能够高效、稳定地运行，降低了运维复杂度和成本。

表 1-1 Jenkins 部署环境对比

部署环境	适合场景	优点	缺点
物理机	对性能和硬件要求高，且资源需求相对稳定，不需要频繁扩展的场景。	• 高性能：物理机直接访问硬件资源，性能表现好。 • 完全控制：用户可以完全控制硬件配置、操作系统和网络等资源。 • 稳定性：系统不会受到虚拟化平台或云环境中的资源竞争的影响。	• 扩展性差：如果遇到任务负载增加，扩展困难，且扩展成本较高。 • 高成本：需要采购、维护和管理硬件设备，初期投入较大。 • 资源利用率低：物理机资源在负载较低时无法高效利用，可能存在资源浪费。 • 维护复杂：物理机需要定期进行硬件维护（如更换硬盘、内存等），且硬件故障时可能影响整个系统的运行。

部署环境	适合场景	优点	缺点
虚拟机	中小型项目、多个团队或项目共享一台物理机的场景。	• 快速扩展：部署虚拟机时，可以根据需求快速创建新的虚拟机实例，便于横向扩展。 • 资源隔离：每个虚拟机都是独立的，可以确保不同项目之间的环境隔离，避免互相影响。 • 成本较低：相比物理机，虚拟机在资源利用上更加高效，减少了硬件投资。	• 性能损失：虚拟机的性能会受到虚拟化层的影响，性能较物理机低。 • 资源竞争：多个虚拟机在同一物理主机上运行时，可能会发生资源争用，影响性能和稳定性。 • 管理复杂：虚拟机的管理比物理机稍微复杂，需要有效的资源调度和管理工具。
CCE Standard/ Turbo集群	大规模分布式环境、持续集成和持续交付的场景。	• 弹性伸缩：Jenkins可以在集群中实现自动伸缩。根据负载自动增加或减少节点，具有非常高的弹性。 • 高可用性：具有自愈和容错能力。如果某个节点或容器失败，系统会自动恢复任务，保证Jenkins的持续运行。 • 资源高效利用：通过容器化，Jenkins可以以更高效的方式共享和利用计算资源。每个容器都能独立运行，减少了资源浪费。	• 运维管理复杂：需要手动管理底层服务器的资源分配和扩展，运维成本高。 • 弹性速度慢：需要制定节点和负载的弹性联合策略，过程通常需要分钟级别的等待，影响效率和响应速度。 • 成本控制困难：容器节点需要预先分配资源，当资源未被充分利用时，会造成资源浪费。
CCE Autopilot集群	自动化管理需求高、持续集成和持续交付的场景。	• 简化运维：作为全托管的Serverless解决方案，简化容量规划和节点购买流程，用户无需管理和维护底层资源设施，大幅减少运维工作量。 • 秒级弹性伸缩：无需提前规划和预留资源，实现Jenkins秒级弹性，根据负载规模自动进行扩缩，确保业务的连续性和性能的最优化。 • 成本可控：采用按量计费模式，按照Jenkins实际使用的资源量（以秒为单位）支付费用，实现真正的按需付费，减少不必要的成本支出。	• 灵活性较差：无法对集群资源进行精细控制。 • 故障排除难度较高：集群出现问题时，用户难以获取详细的底层日志和诊断信息，可能会延长故障排除的时间。

约束与限制

Jenkins系统的维护由开发者自行负责，使用过程中CCE服务不对Jenkins系统提供额外维护与支持。

Jenkins 的基本概念

- Master节点（Jenkins Master）：是Jenkins系统的核心部分，负责管理和协调所有工作。可以将其理解为“管理者”，不直接执行构建任务，而是分配任务给其他“工人”（即Agent节点）。

  说明

  Master节点提供Web界面供用户操作和查看任务进度，后续步骤中提到的Jenkins界面都是指Master节点的Web界面。

- Agent节点（Jenkins Agent）：是Jenkins负责执行实际构建任务的Pod或机器，执行的是具体的工作任务。可以同时设置多个Agent节点，分担工作负载，提高任务的并行度和效率。
- 插件（Plugin）：是扩展Jenkins功能的核心方式。Jenkins可以根据需求安装不同的插件来支持版本控制、构建工具和部署等功能。同时，插件可以帮助Jenkins集成不同的工具和技术，如Kubernetes、Git和Maven等。
- 流水线（Pipeline)：是一个自动化的工作流程，把软件开发过程中的多个阶段（如构建、测试和部署等）串联在一起，确保每一个步骤都能按照一定的顺序和规则自动执行。
- 云（Cloud）：用于配置各种云环境，如集群、容器和虚拟机等，可以灵活地使用外部云平台的计算资源，实现Jenkins Agent的动态管理。

方案架构

Jenkins部署方案有两种：单节点部署和分布式部署。

- 单节点部署：Jenkins作为独立的实例运行，所有的构建和操作都在同一节点（Jenkins Master）上，即Jenkins Master既负责任务调度和系统管理，又负责执行具体的构建任务。所有任务都在同一节点上运行，容易导致系统资源的过度消耗，并且随着项目规模和构建任务的增多，单节点部署可能成为性能瓶颈。该部署方式主要适用于小型团队或个人开发环境。
- 分布式部署：Jenkins Master负责任务调度和系统管理，而Jenkins Agent负责执行具体的构建任务。Jenkins Master接受来自用户的构建请求，并将这些任务分发给可用的Jenkins Agent完成。每个Jenkins Agent可以独立配置，支持不同的操作系统和构建工具，从而提供灵活的构建环境和可扩展性。同时，管理与执行分开可以有效地提升系统性能和响应速度。该架构适用于大规模的生产环境，特别是当构建任务量较大或对并发构建有较高需求时。

本文以分布式部署为例，介绍如何在CCE Autopilot集群中部署并使用Jenkins，具体的方案架构和操作步骤请参见**图1-1**和**表1-2**。

图 1-1 方案架构

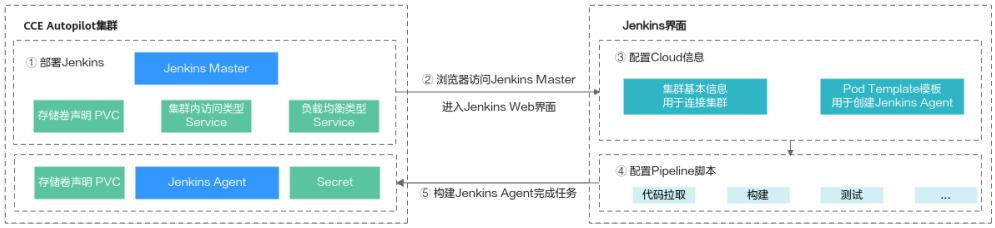


表 1-2 操作步骤

步骤	说明	关联镜像
在集群中安装部署Jenkins Master	在CCE Autopilot集群中安装部署Jenkins Master，负责管理任务。	Jenkins Master工作负载：本示例中使用jenkins:lts镜像。 **说明** jenkins:lts表示Jenkins LTS版本的Docker镜像。LTS版本是Jenkins官方提供的长期支持的版本镜像，相对稳定，并且会在更长时间内接受安全更新和bug修复，通常适用于需要稳定环境的生产系统，更多信息请参见**LTS Release Line**。
在Jenkins界面中配置Jenkins Agent	在Jenkins界面安装Kubernetes的插件，并在Cloud中配置集群信息用于连接集群。配置Pod Template模板，作为后续在集群中动态创建Agent Pod的模板。	Jenkins Agent的Pod实例：本示例中使用3个镜像，分别是jenkins/inbound-agent:4.13.3-1、maven:3.8.1-jdk-8和gcr.io/kaniko-project/executor:v1.23.2-debug。 • jenkins/inbound-agent:4.13.3-1：用于连接Jenkins Agent和Jenkins Master，保证任务的连续执行。 • maven:3.8.1-jdk-8：用于执行Pipeline中的打包任务。 • gcr.io/kaniko-project/executor:v1.23.2-debug：用于在容器内构建和推送Docker镜像。

步骤	说明	关联镜像
在Jenkins界面中构建并执行Pipeline	在Jenkins界面编写Pipeline脚本，将任务编译成Jenkins Master能够理解的语言。Jenkins Master负责协调整个流水线的执行过程，通过Kubernetes插件在集群中动态创建Jenkins Agent（Pod形式呈现），并将任务分发给Jenkins Agent处理。任务完成后，Jenkins Agent自动销毁。 在本示例中，Pipeline的任务内容是从代码仓中拉取代码，将代码打包成镜像，并推送到SWR镜像仓库中。	推送至SWR的镜像：tomcat。

1.2 资源和成本规划

本示例涉及的资源如下：

表 1-3 资源和成本规划

资源	资源规格	说明
CCE Autopilot集群	• 集群类型：CCE Autopilot集群 • 计费模式：按需计费 • 集群版本：v1.28 • 插件选择：CoreDNS域名解析、Kubernetes Metrics Server	需要创建1个集群。 涉及集群管理和终端节点等费用，具体请参见**计费说明**。
Pod实例	Jenkins Master: • CPU：4Cores • 内存：4Gi • 存储：30GiB Jenkins Agent: • CPU：0.5Cores • 内存：1Gi • 存储：30GiB	需要创建2个Pod，即Jenkins Master和Jenkins Agent。 涉及Pod费用，具体请参见**计费说明**。

资源	资源规格	说明
弹性云服务器ECS	• 计费模式：按需计费 • 虚拟机节点类型：通用计算增强型 • 节点规格：2vCPUs | 4GiB • 操作系统：CentOS 7.6 • 系统盘：40GiB | 通用型SSD • 弹性公网IP： – 共享类型：独享 – 计费方式：按流量计费 – 带宽大小：5Mbit/s	需要创建1台ECS，并与集群处于同一VPC。 涉及ECS的配置费用和弹性公网IP流量费用，具体请参见。
极速文件存储SFS Turbo	• 计费模式：按需计费 • 类型：40MB/s/TiB • 容量：1.2TB	需要创建1个SFS Turbo。 涉及SFS Turbo的使用费用，具体请参见**价格计算器-SFS Turbo**。
弹性负载均衡ELB	• 计费模式：按需计费 • 实例规格：共享型 • 公网带宽：按流量计费 • 带宽：5Mbit/s	需要创建1个ELB。 涉及ELB的使用费用，具体请参见**价格计算器-ELB**。
容器镜像服务SWR（共享版）	-	需要创建1个组织。 不涉及费用。

1.3 实施步骤

1.3.1 在集群中安装部署 Jenkins Master

在CCE Auotpilot集群中安装部署Jenkins Master（无状态工作负载），负责管理任务。

说明

Jenkins界面中的词条可能因版本不同而存在一些差异，本文中的截图仅供您参考。

准备工作

- 您已购买一个CCE Autopilot集群，具体操作请参见**购买CCE Autopilot集群**。
- 您已购买一台Linux系统的ECS虚拟机，该ECS与集群处于同一VPC，并绑定弹性公网IP，具体操作请参见**快速购买和使用Linux ECS**。此外，该ECS还需配备kubectl命令并**通过kubectl连接集群**。

- 您已创建一个状态可用的极速文件存储（SFS Turbo），且该SFS Turbo与集群处于同一VPC。
- 您已在SWR中创建一个组织，并且该组织与集群处于同一区域，具体操作请参见**创建组织**。

通过 CCE Autopilot 集群安装部署 Jenkins Master

步骤1 登录ECS虚拟机，具体操作请参见**通过CloudShell登录Linux ECS**。

步骤2 创建SFS Turbo类型的持久化存储卷（PV）和持久化存储卷声明（PVC），供Jenkins Master存储持久化数据。

1. 创建一个名为pv-jenkins-master.yaml的YAML文件，用于创建PV，文件名称可自定义。

```
vim pv-jenkins-master.yaml
```

文件内容如下，本示例仅涉及必要参数，更多参数信息请参见**通过静态存储卷使用已有极速文件存储**。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: everest-csi-provisioner  # 指定存储驱动，固定为everest-csi-provisioner
  name: pv-jenkins-master    # PV的名称，可自定义
spec:
  accessModes:
  - ReadWriteMany      # 访问模式，极速文件存储必须为ReadWriteMany
  capacity:
    storage: 500Gi     # PV申请容量大小
  csi:
    driver: sfsturbo.csi.everest.io    # 挂载依赖的存储驱动，固定为sfsturbo.csi.everest.io
    fsType: nfs        # 指定存储类型，固定为nfs
    volumeHandle: ea8a59b6-485c-xxx    # 极速文件存储的ID
    volumeAttributes:
      everest.io/share-export-location: ea8a59b6-485c-xxx.sfsturbo.internal:/    # 极速文件存储的共享路径
  persistentVolumeReclaimPolicy: Retain    # 回收策略
  storageClassName: csi-sfsturbo           # SFS Turbo存储类名称
```

输入完成后，**Esc键**退出编辑，输入**:wq**保存。

表 1-4 关键参数说明

参数	示例	描述
storage	**500Gi**	表示PV申请容量，单位为Gi。
volumeHandle	**ea8a59b6-485c-xxx**	表示SFS Turbo的ID。 **获取方法：**在CCE控制台，单击左上角，单击“存储 > 弹性文件服务 SFS”，左侧导航栏单击“SFS Turbo > 文件系统列表”。在列表中单击对应的SFS Turbo文件存储名称，在“基本信息”页签中复制“ID”后的内容即可。

参数	示例	描述
everest.io/share-export-location	**ea8a59b6-485c-xxx.sfsturbo.internal:/**	表示SFS Turbo的共享路径，是用于共享存储的目录。多个Pod可以通过网络访问该路径，从而共享同一存储资源。 **获取方法：**在CCE控制台，单击左上角，单击“存储 > 弹性文件服务 SFS”，左侧导航栏单击“SFS Turbo > 文件系统列表”。在列表中单击对应的SFS Turbo文件存储名称，在“基本信息”页签中复制“共享路径”后的内容即可。
persistentVolumeReclaimPolicy	**Retain**	表示PV的回收策略，仅支持Retain回收策略。 Retain：删除PVC，PV资源与底层存储资源均不会被删除，需要手动删除回收。PVC删除后PV资源状态为“已释放（Released）”，不能直接再次被PVC绑定使用。
storageClassName	**csi-sfsturbo**	表示极速文件存储的存储类名称。

2. 执行以下命令，创建PV。

```
kubectl create -f pv-jenkins-master.yaml
```

回显如下，表示名为pv-jenkins-master的PV已创建。

```
persistentvolume/pv-jenkins-master created
```

3. 创建一个名为pvc-jenkins-master.yaml的YAML文件，用于创建PVC，文件名称可自定义。

```
vim pvc-jenkins-master.yaml
```

文件内容如下，本示例仅涉及必要参数，更多参数信息请参见**通过静态存储卷使用已有极速文件存储**。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-jenkins-master    # PVC的名称，可自定义
  namespace: default    # 指定命名空间，需要与工作负载处于同一命名空间
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: everest-csi-provisioner  # 指定存储驱动，固定为everest-csi-provisioner
spec:
  accessModes:
  - ReadWriteMany                # 访问模式，极速文件存储必须为ReadWriteMany
  resources:
    requests:
      storage: 500Gi               # PVC申请容量大小，与PV的容量保持一致
  storageClassName: csi-sfsturbo      # SFS Turbo存储类名称，必须与PV的存储类一致
  volumeName: pv-jenkins-master       # 关联的PV的名称
```

输入完成后，**Esc键**退出编辑，输入**:wq**保存。

表 1-5 关键参数说明

参数	示例	描述
storage	**500Gi**	表示PVC申请容量，单位为Gi。 必须与1中PV申请容量一致。
storageClassName	**csi-sfsturbo**	表示存储类名称。 必须与1中PV的存储类一致。
volumeName	**pv-jenkins-master**	表示关联的PV名称。 必须与1中PV名称一致。

4. 执行以下命令，创建PVC。
```
kubectl create -f pvc-jenkins-master.yaml
```
回显如下，表示名为pvc-jenkins-master的PVC已创建。
```
persistentvolumeclaim/pvc-jenkins-master created
```
5. 检查PV和PVC是否绑定成功，绑定成功后才能在Pod中挂载PVC。当PV和PVC都为绑定状态时，可以认为二者绑定成功。

首先，利用以下命令检查PV状态。
```
kubectl get pv
```
回显结果如下，STATUS为Bound，说明PV为绑定状态。
```
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                    STORAGECLASS   REASON   AGE
pv-jenkins-master   500Gi      RWX            Retain           Bound    default/pvc-jenkins-master   csi-sfsturbo            88s
```
其次，利用以下命令检查PVC状态。
```
kubectl get pvc
```
回显结果如下，STATUS为Bound，说明PVC为绑定状态。
```
NAME                STATUS   VOLUME              CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvc-jenkins-master   Bound    pv-jenkins-master   500Gi      RWX            csi-sfsturbo   61s
```
该步骤创建的PV和PVC都为绑定状态，可以认为二者绑定成功。

步骤3 利用jenkins:lts镜像创建无状态工作负载jenkins-master，并挂载**步骤2.4**中创建的PVC。

说明

本示例使用jenkins:lts镜像，jenkins:lts表示Jenkins LTS版本的Docker镜像。LTS版本是Jenkins官方提供的长期支持的版本镜像，相对稳定，并且会在更长时间内接受安全更新和bug修复，通常适用于需要稳定环境的生产系统，更多信息请参见**LTS Release Line**。

本示例将Jenkins Master部署为无状态工作负载。Jenkins Master核心功能在于管理和调度任务，不依赖于持久化数据，因此将其设置为无状态工作负载能够提高系统的灵活性和可伸缩性。

您可以根据需要选择不同的镜像和工作负载类型。

1. 创建名为jenkins-master的YAML文件，用于创建jenkins-master工作负载，文件名称可自定义。
```
vim jenkins-master.yaml
```
文件内容如下，本示例仅涉及必要参数，更多参数信息请参见**创建无状态负载（Deployment）**。
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins-master    # 无状态工作负载的名称
```



```
  namespace: default      # 指定命名空间，与PVC命名空间保持一致
spec:
  replicas: 1       # Pod实例的个数
  selector:
    matchLabels:    # 工作负载的标签选择器，用于匹配所选择的Pod，确保创建的Deployment可以正确地选择和管理所需的Pod
      app: jenkins-master
  template:
    metadata:
      labels:       # 指定Pod实例的标签，需要与工作负载的matchLabels一致，确保创建的Pod实例受Deployment的统一管理
        app: jenkins-master
    spec:
      containers:
        - name: container-1
          image: jenkins/jenkins:lts    # 使用jenkins:lts镜像
          resources:        # 用于配置容器的资源限制和请求
            limits:         # 表示容器可使用的最大资源量
              cpu: '4'
              memory: 4Gi
            requests:       # 表示容器启动时所需的资源量
              cpu: '4'
              memory: 4Gi
          volumeMounts:    # 指定容器挂载的卷
            - name: pvc-jenkins-master
              mountPath: /var/jenkins_home   # 挂载路径，一般设置为“/var/jenkins_home”
      volumes:    # 用于定义Pod使用的存储卷，对应一个已创建的PVC
        - name: pvc-jenkins-master    # 定义卷的名称，可自定义
          persistentVolumeClaim:
            claimName: pvc-jenkins-master    # 指定要使用的PVC
      imagePullSecrets:
        - name: default-secret
```

输入完成后，**Esc键**退出编辑，输入**:wq**保存。

2. 执行以下命令，创建无状态工作负载jenkins-master。

```
kubectl create -f jenkins-master.yaml
```

回显如下：

```
deployment/jenkins-master created
```

3. 检查无状态工作负载jenkins-master是否创建成功，即检查该工作负载创建的Pod的STATUS是否为Running。

```
kubectl get pod
```

回显结果如下，Name为jenkins-master-xxx的Pod的STATUS皆为Running，说明无状态工作负载jenkins-master创建成功。

```
NAME                              READY   STATUS    RESTARTS   AGE
jenkins-master-6f65c7b8f7-255gn   1/1     Running   0          72s
```

步骤4 创建服务（Service），用来访问Jenkins Master。Jenkins容器镜像有两个端口：8080和50000，需要分别配置。其中8080端口供Web登录使用，50000端口供Jenkins Master和Jenkins Agent连接使用。

本示例需要创建两个Service：

- 负载均衡（LoadBalancer)类型Service：仅用于提供Web的外部访问（公网访问），使用8080端口。

  Service名称：jenkins-web（可自定义），容器端口：8080，访问端口：8080，ELB网络类型：public。

- 集群内访问（ClusterIP）类型Service：用于Jenkins Agent连接Jenkins Master。Jenkins要求Web的外部访问地址和Agent访问地址一致，因此包含Web访问的8080端口和Agent访问的50000端口。

  Service名称：jenkins-agent（可自定义），容器端口1：8080，访问端口1：8080，容器端口2：50000，访问端口2：50000。

说明

本示例中，后续步骤创建的Jenkins Agent均与Jenkins Master处于同一集群，因此Jenkins Agent连接使用ClusterIP类型的Service。

如果Jenkins Agent需要跨集群或使用公网连接Jenkins Master，请自行选择合适的Service类型。但需要注意的是，Jenkins要求Web的外部访问地址和Agent访问地址一致，因此Agent连接的地址必须同时开放8080和50000端口，而仅用于Web访问的地址可以只开放8080端口、不开放50000端口。

1. 创建名为jenkins-web的YAML文件，用于创建负载均衡类型的Service，文件名称可自定义。

   本示例基于自动创建的弹性负载均衡器（ELB）创建Service，如果您想使用已有的ELB创建Service，请参见**通过kubectl命令行创建-使用已有ELB**。

   ```
   vim jenkins-web.yaml
   ```

   文件内容如下，本示例仅涉及必要参数，更多参数信息请参见**通过kubectl命令行创建-自动创建ELB**。

   ```
   apiVersion: v1
   kind: Service
   metadata:
     name: jenkins-web    # Service名称，可自定义
     namespace: default   # 指定命名空间
     labels:
       app: jenkins-web   # 指定Service的标签
     annotations:    # 自动创建ELB
       kubernetes.io/elb.class: performance    # 指定ELB类型，仅支持独享型负载均衡，即performance
       kubernetes.io/elb.autocreate: '{
         "type": "public",
         "bandwidth_name": "cce-bandwidth-xxx",
         "bandwidth_chargemode": "traffic",
         "bandwidth_size": 5,
         "bandwidth_sharetype": "PER",
         "eip_type": "5_bgp",
         "available_zone": [
           "cn-east-3a"
         ],
         "l4_flavor_name": "L4_flavor.elb.s1.small"
       }'
   spec:
     selector:    # 用于选择需要绑定的Pod实例
       app: jenkins-master
     ports:       # 定义Service的端口信息
     - name: cce-service-0
       targetPort: 8080   # Service访问目标Pod的端口，与Pod中运行的应用密切相关
       port: 8080 # 外部访问Service的端口，也是负载均衡上的监听端口
       protocol: TCP
     type: LoadBalancer   # Service的类型，LoadBalancer表示负载均衡类型的服务Service
   ```

   输入完成后，**Esc键**退出编辑，输入**:wq**保存。

表 1-6 kubernetes.io/elb.autocreate 字段关键参数说明

参数	示例	描述
type	**public**	表示ELB的网络类型，公网或者私网。 – public：公网型ELB，需要绑定弹性公网IP，允许公网和私网访问。 – inner：私网型ELB，不需要绑定弹性公网IP，只允许私网访问。 由于该Service用于提供Web的外部访问（公网访问），因此需要设置为public。

参数	示例	描述
bandwidth_name	**cce-bandwidth-xxx**	表示带宽的名称，默认值为：cce-bandwidth-xxx，“xxx”可自定义。 取值范围：只能由中文、英文字母、数字、下划线、中划线和点组成，且长度范围为1-64个字符。
bandwidth_chargemode	**traffic**	表示带宽付费模式。 – bandwidth：按固定的带宽计费。 – traffic：按实际消耗的流量计费。
bandwidth_size	**5**	表示带宽大小，默认1Mbit/s~2000Mbit/s，请根据区域带宽支持范围设置。 调整带宽时的最小单位会根据带宽范围不同存在差异，只能选择最小单位的整数倍设置带宽。 – 小于等于300Mbit/s：默认最小单位为1Mbit/s。 – 300Mbit/s~1000Mbit/s：默认最小单位为50Mbit/s。 – 大于1000Mbit/s：默认最小单位为500Mbit/s。
bandwidth_sharetype	**PER**	表示带宽共享方式，PER为独享带宽。
eip_type	**5_bgp**	表示弹性公网IP类型。 – 5_bgp：全动态BGP。 – 5_sbgp：静态BGP。
available_zone	***cn-east-3a***	表示负载均衡所在可用区，独享型负载均衡器独有字段。 可用区的取值为区域后加入序号a，b，c等，如上海一可用区1为“cn-east-3a”，上海一可用区2为“cn-east-3b”，以此类推。 集群所在的区域对应的值请参见**地区和终端节点**。
l4_flavor_name	**L4_flavor.elb.s1.small**	表示四层负载均衡实例规格名称，独享型负载均衡器独有字段。 可以通过**查询规格列表**获取所有支持的类型。

2. 执行以下命令，创建负载均衡类型的Service，用于提供Web的外部访问。
```
kubectl create -f jenkins-web.yaml
```
回显如下：
```
service/jenkins-web created
```
3. 创建名为jenkins-agent的YAML文件，用于创建集群内访问类型的Service，文件名称可自定义。
```
vim jenkins-agent.yaml
```
文件内容如下，本示例仅涉及必要参数，更多参数信息请参见**集群内访问（ClusterIP）**。
```
apiVersion: v1
kind: Service
```



```
metadata:
  name: jenkins-agent      # Service名称，可自定义
  namespace: default       # 指定命名空间
  labels:
    app: jenkins-agent
spec:
  ports:    # 定义Service的端口信息
  - name: service0          # 端口1：用于保证Web的外部访问地址和Agent访问地址一致
    port: 8080             # 内部访问Service的端口
    protocol: TCP          # 访问Service的协议，支持TCP和UDP
    targetPort: 8080       # Service访问目标容器的端口，此端口与容器中运行的应用强相关
  - name: service1          # 端口2：用于Master和Agent连接使用
    port: 50000
    protocol: TCP
    targetPort: 50000
  selector:                # 标签选择器，Service通过标签选择Pod，将访问Service的流量转发给Pod
    app: jenkins-master
  type: ClusterIP          # Service的类型，ClusterIP表示在集群内访问类型的Service
```

输入完成后，**Esc键**退出编辑，输入**:wq**保存。

4. 执行以下命令，创建集群内访问类型的Service，用于Agent连接Master。

```
kubectl create -f jenkins-agent.yaml
```

回显如下：

```
service/jenkins-agent created
```

5. 检查上述服务是否创建成功。

```
kubectl get svc
```

回显如下，可以通过“jenkins-web的负载均衡公网IP：8080”登录Jenkins，即“xx.xx.xx.xx:8080”。

```
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP                  PORT(S)              AGE
jenkins-agent   ClusterIP      10.247.22.139    <none>                       8080/TCP,50000/TCP   34s
jenkins-web     LoadBalancer   10.247.76.78     xx.xx.xx.xx,192.168.0.239    8080:31694/TCP       15m
kubernetes      ClusterIP      10.247.0.1       <none>                       443/TCP              3h3m
```

步骤5 登录并初始化Jenkins。

1. 在浏览器中输入负载均衡的“jenkins-web的负载均衡公网IP:8080”，打开jenkins配置界面。

   本示例中浏览器访问地址为*113.44.78.102:8080*。

2. 初次访问时界面会提示获取初始管理员密码，该密码可在jenkins的Pod中获取，具体步骤如下。

   首先，返回ECS终端界面，输入以下命令查找Pod名称。

```
kubectl get pod|grep jenkins-master
```

   回显如下，其中“jenkins-master-6f65c7b8f7-255gn”即为Pod名称。

```
jenkins-master-6f65c7b8f7-255gn   1/1     Running   0          144m
```

   其次，输入以下命令进入“jenkins-master-6f65c7b8f7-255gn”内部。

```
kubectl exec -it jenkins-master-6f65c7b8f7-255gn -- /bin/sh
```

   最后，输入以下命令获取初始管理员密码。

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

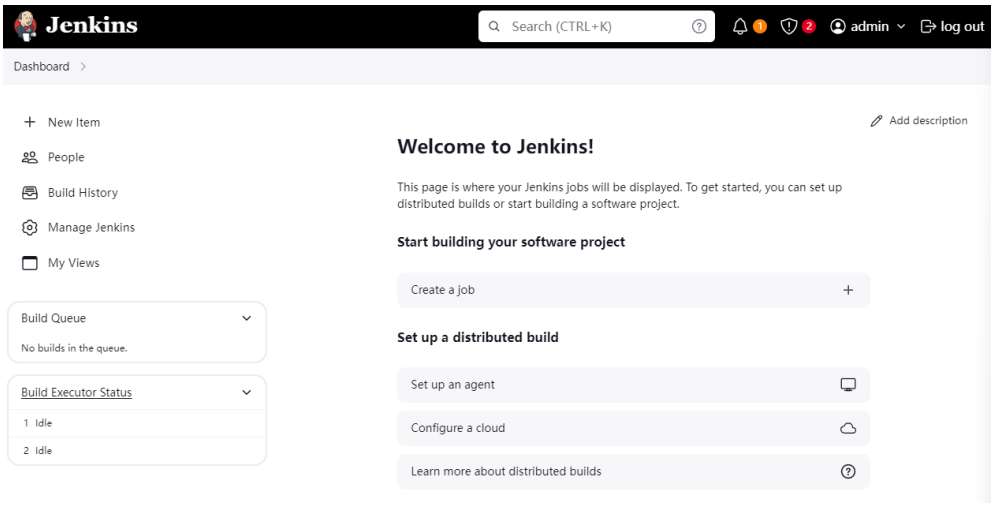
   回显如下，回显内容即为初始管理员密码。

```
98afebb226944f6d81e6e45a7c56986d
```

3. 首次登录时选择安装推荐的的插件即可，并根据界面提示创建一个管理员。完成初始配置后，即可进入Jenkins界面。

图 1-2 首次登录页面

图 1-3 Jenkins 界面



----结束

1.3.2 在 Jenkins 界面中配置 Jenkins Agent

在本章中，您需要完成以下部分：

- 在Jenkins界面安装Kubernetes的插件，并在Cloud中配置集群信息用于连接集群。
- 在Jenkins界面配置Pod Template模板，作为后续在集群中动态创建Agent Pod的模板。

在上述过程中，需要同步在集群中进行一些配置，具体请见**集群的准备工作**。

集群的准备工作

在配置Jenkins Agent之前，集群侧需要提前进行一些操作，用来支持Jenkins Agent的后续配置。

步骤1 下载集群kubectl配置文件，作为Jenkins连接集群的凭证。

返回**CCE控制台**，单击对应集群名称，在“总览”界面的右侧“连接信息”模块，单击“配置”，下载kubectl配置文件。

图 1-4 连接信息

步骤2 创建PVC，用于持久化存储Jenkins Agent完成任务过程中产生的数据。

集群控制台左侧列表单击“存储”，右上角单击“创建存储卷声明 PVC”。在“创建存储卷声明 PVC”界面输入以下参数，单击“创建”。

- 存储卷声明类型：极速文件存储
- PVC名称：jenkins-agent
- 创建方式：新建存储卷 PV
- 极速文件存储：选择已创建的极速文件存储
- PV名称：pv-efs-jenkins-agent

图 1-5 创建 PVC

步骤3 创建具有SWR认证信息的Secret，用于后续创建的kaniko容器向SWR推送镜像的凭证。

返回ECS虚拟机，依次执行以下命令：

1. 下载jq命令，用于处理和操作JSON数据，支持查询、筛选、修改和格式化等功能。以下命令以CentOS 7.6操作系统为例。


```
yum install jq
```

2. 创建docker-registry类型的Secret，用于存储SWR的认证信息。同时，提取并解码SWR的认证信息，保存至/tmp/config.json文件中。
   - docker-server：填写SWR的镜像仓库地址，格式为swr.[区域].myhuaweicloud.com。
     获取方式：需要先获取不同区域对应的值，请参见**地区和终端节点**。随后将swr.[区域].myhuaweicloud.com中的[区域]替换为对应值即可，如上海一：swr.cn-east-3.myhuaweicloud.com。
   - docker-username：填写SWR登录指令中的用户名。
     获取方式：登录SWR控制台，“总览”界面右上角单击“登录指令”，查看临时登录指令页签中命令，命令中-u后的内容即为用户名。
   - docker-password：填写SWR登录指令中的密码。
     获取方式：登录指令的命令中-p后的内容即为密码。

     说明

     临时登录指令有效时间较短，过期后需要重新配置。
     您可以在“登录指令”界面中选择“长期有效登录指令”，根据页面提示配置相关信息，获取“长期有效登录命令”，进而获得用户名和密码。

   图 1-6 获取 docker-username 和 docker-password


```
kubectl create secret docker-registry swr-secret \
  --docker-server=https://swr.xxx.myhuaweicloud.com \
  --docker-username=xxx \
  --docker-password=xxx \
  --dry-run=client -o json | jq -r '.data.".dockerconfigjson"' | base64 -d > /tmp/config.json
```

3. 利用/tmp/config.json文件创建一个generic类型的Secret，该Secret可以直接挂载在后续创建的Jenkins Agent的Pod实例中。

```
kubectl create secret generic swr-secret --from-file=/tmp/config.json -n default
```

----结束

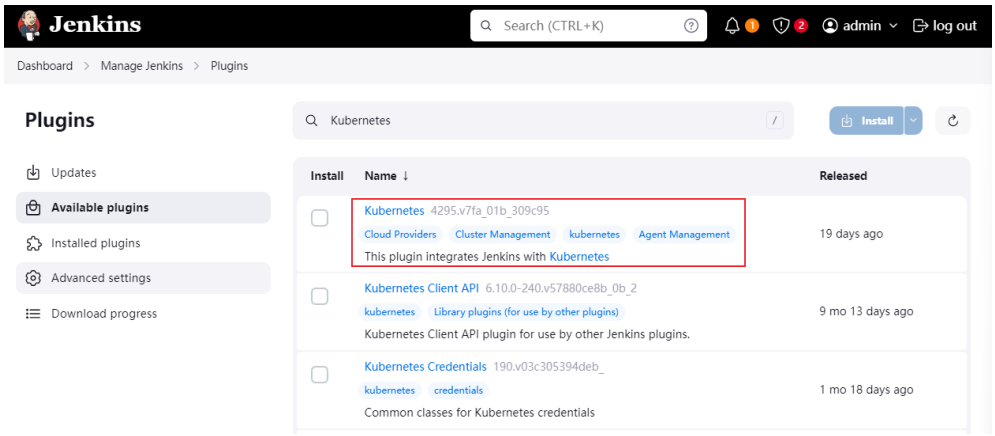
在 Jenkins 界面中配置 Jenkins Agent

步骤1 安装“Kubernetes”插件，用于在集群中动态创建Jenkins Agent（Pod形式呈现），并在每次构建完成后销毁Pod。

在Jenkins Dashboard界面单击左侧“Manage Jenkins”，选择“System Configuration > Plugins > Available plugins”，在“Available plugins”页签中查找并安装“Kubernetes”插件。

本示例中安装的插件版本为**Kubernetes pluginVersion：4295.v7fa_01b_309c95**，插件版本可能随时间变化发生变动，请您自行选择。此外，您可以根据需要安装其他插件，如**Kubernetes CLI Plugin**（允许为Job配置kubectl，从而与Kubernetes集群进行交互）等。

图 1-7 查找“Kubernetes”插件



步骤2 开启“Enable Proxy Compatibility”。

返回Jenkins Dashboard界面，左侧列表单击“Manage Jenkins”，单击“Security > Security”，在“CSRF Protection”模块中勾选“Enable Proxy Compatibility”，最下方单击“Apply”。

说明

开启“Enable Proxy Compatibility”的主要目的是避免“Error 403 No valid crumb was included in the request”错误。

Jenkins通过CSRF保护机制防止跨站请求伪造攻击。当用户执行敏感操作（如构建项目）时，Jenkins会要求提供有效的“crumb”。而在使用反向代理（如 Nginx 或 Apache）或负载均衡器时，请求会从客户端转发到Jenkins服务器。这些代理和负载均衡器可能会修改请求头，导致CSRF令牌（crumb）丢失或未能正确传递，从而引发“Error 403 No valid crumb was included in the request”错误。

启用“Enable Proxy Compatibility”后，Jenkins会采取一种容错机制，使其能够在代理环境下能够正常处理传递的请求，确保CSRF令牌（crumb）能够正确通过代理传递并进行验证。

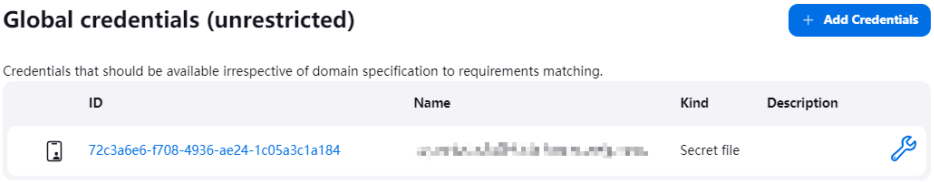
图 1-8 开启“Enable Proxy Compatibility”

步骤3 添加**步骤1**中下载的kubectl配置文件，作为Jenkins连接集群的凭证。

界面上方路径中单击“Manage Jenkins”，单击“Security > Credentials”，单击“Stores scoped to Jenkins > System > Global credentiials (unrestricted)”,右侧单击“Add Credentials”。

在“New credentials”界面中，“Kind”选择“Secret file”，“Scope”选择“Global (Jenkins, nodes, items, all child items, etc)”，“File”选择下载的kubectl配置文件，其他参数保持默认，单击“Create”。

图 1-9 上传的凭证



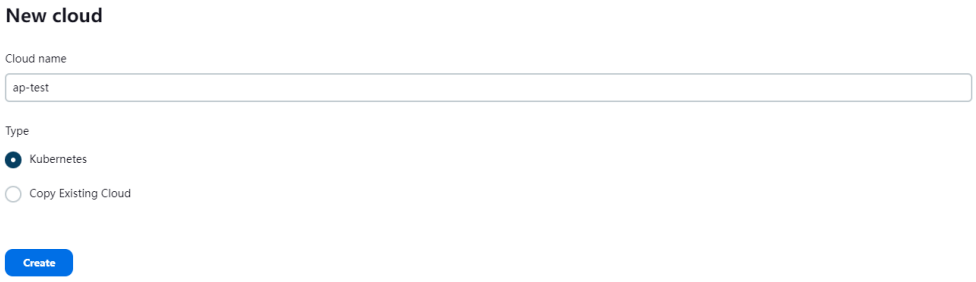
步骤4 创建Cloud，Cloud用于配置集群信息，便于Jenkins匹配到正确的集群并连接。

1. 填写Cloud基本信息。

   界面上方路径中单击“Manage Jenkins”，单击“SystemConfiguration > Clouds”，单击“New Cloud”创建新Cloud。

   “Cloud name”输入Cloud名称，名称可自定义，“Type”勾选“Kubernetes”，单击“Create”。

   图 1-10 Cloud 基本信息

2. 填写集群相关信息。

   图 1-11 集群详细信息

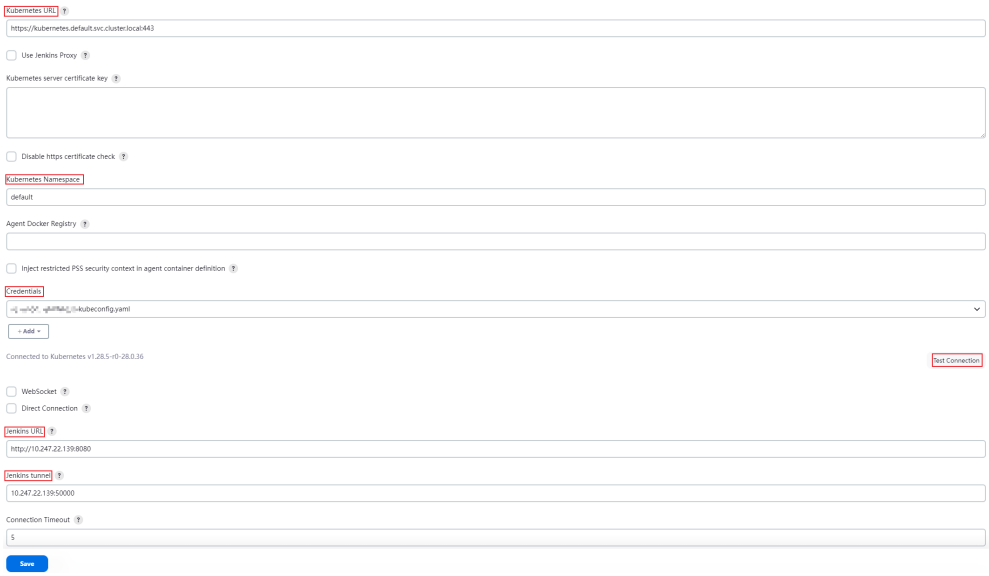
   

表 1-7 集群详细参数说明

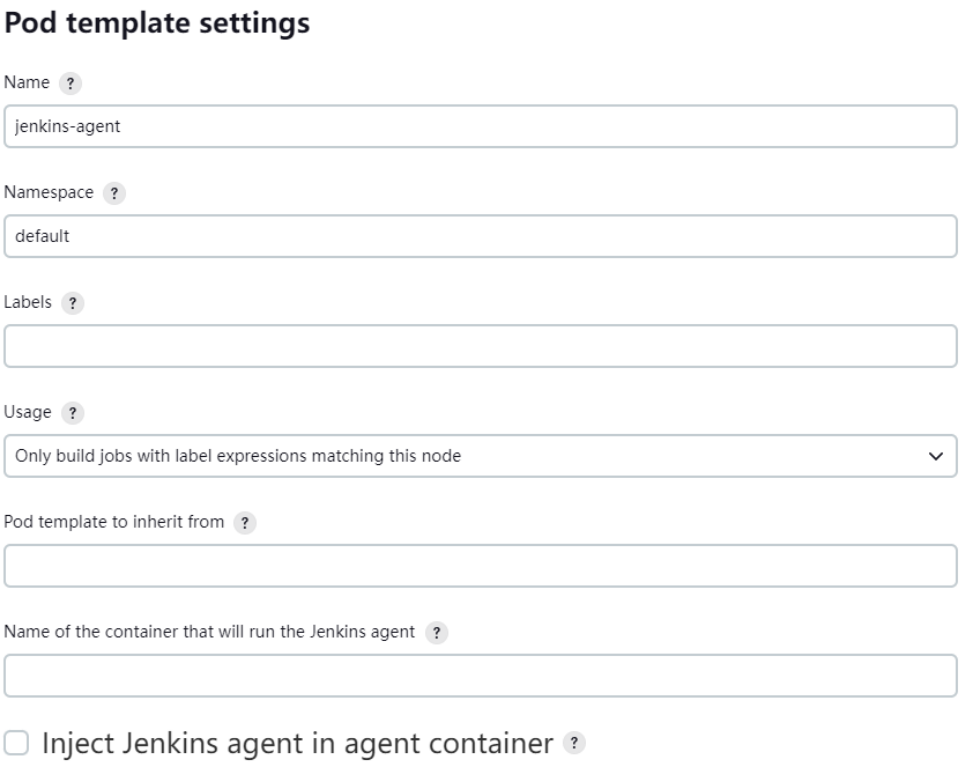
参数	示例	描述
Kubernetes URL	**https://kubernetes.default.svc.cluster.local:443**	表示集群APl Server地址。 可直接填写“https://kubernetes.default.svc.cluster.local:443"，该地址表示集群内部访问Kubernetes API Server的标准DNS地址。
Kubernetes Namespace	**default**	表示动态创建的Jenkins Agent运行的命名空间。 **说明** 该命名空间需与**步骤3**创建的工作负载jenkins-master所在命名空间一致。
Credentials	**xxx-kubeconfig.yaml**	表示集群连接凭证。 请选择**步骤3**中上传的连接凭证。 **说明** 选择凭证之后，请单击右侧“Test Connection”，查看是否能正常连接集群。 若左侧回显“Connected to Kubernetes xxx”，则说明集群能够正常连接。
Jenkins URL	**http://10.247.22.139:8080**	表示Jenkins的访问路径。 请填写**步骤4**的集群内访问的IP地址，端口号为8080。
Jenkins tunnel	**10.247.22.139:50000**	用于在Jenkins Master与Jenkins Agent之间建立连接。 请填写**步骤4**的集群内访问的IP地址，端口号为50000。

3. 确保以上信息无误后，请单击“Save”。

步骤5 配置Pod模板。通过该模板Jenkins能够在集群中按需创建Jenkins Agent的Pod实例，并使用创建的Pod运行Jenkin任务。这些Pod是按需创建的，任务执行完毕后会被自动销毁。

1. 单击新配置的Cloud名称，单击“Pod Templates > Add a pod template”。
2. 配置Pod模板基本参数。
   - Name：表示Pod模板的名称，可自定义，如jenkins-agent。
   - Namespace：表示创建Pod的命名空间，需要与Cloud中的命名空间一致，如default。
   - 其他参数：您可以根据需要进行填写，本示例保持默认。

图 1-12 Pod 模板基本参数



3. 添加容器模板。本示例需要添加3个容器模板，具体参数信息将以容器1、容器2和容器3的形式在**表1-8**中呈现，您可以按照表中信息自行添加3个容器模板。
   - 容器1：使用jenkins/inbound-agent:4.13.3-1镜像，主要用于连接Jenkins Agent和Jenkins Master，保证任务的连续执行。
   - 容器2：使用maven:3.8.1-jdk-8镜像，主要用于执行流水线（Pipeline）中的打包任务。
   - 容器3：使用gcr.io/kaniko-project/executor:v1.23.2-debug，主要用于在容器内构建Docker镜像。

     说明

     建议提前将上述3个镜像推送至SWR的镜像仓库中，这样可以提高构建速度和可靠性，具体操作步骤请参见**客户端上传镜像**。

     将镜像存储在SWR的镜像仓库中，Jenkins在构建Pod时无需重新从外部源拉取镜像，从而加快镜像获取的速度并减少网络延迟。这不仅能够优化构建时间，还能有效减少因网络波动或镜像拉取失败而导致的构建失败风险，确保构建过程更加稳定和高效。

图 1-13 容器模板参数

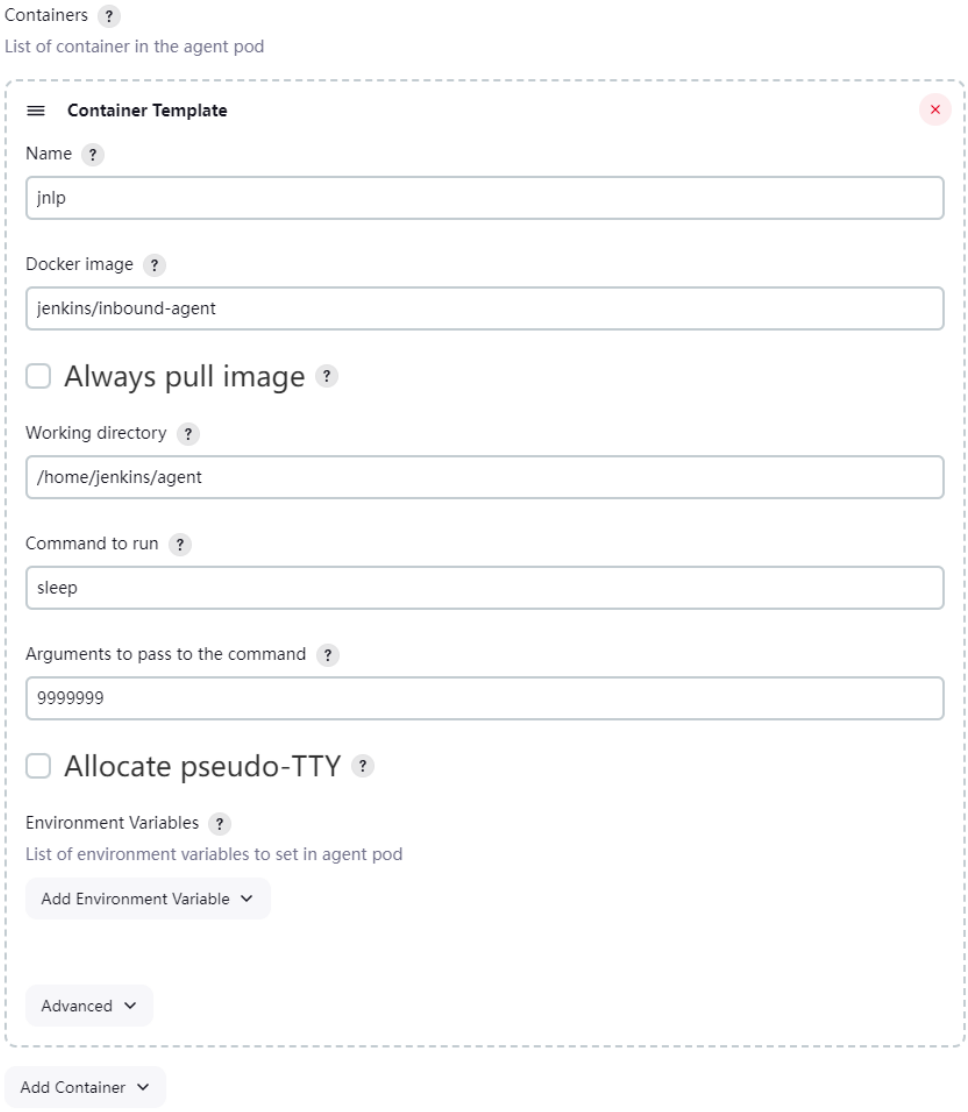


表 1-8 容器模板参数说明

参数	示例	描述
Name	**容器1：jnlp** **容器2：maven** **容器3：kaniko**	表示在集群中构建的容器名称。 容器1的名称通常固定为jnlp，其他可自定义。
Docker image	**容器1：jenkins/inbound-agent** **容器2：maven:3.8.1-jdk-8** **容器3：gcr.io/kaniko-project/executor:v1.23.2-debug**	表示构建容器时需要的镜像。 如果您已将相应的镜像上传至SWR，则需要将对应值改为SWR中的镜像地址。

参数	示例	描述
Working directory	**容器1~3：/home/jenkins/agent**	表示容器在执行构建任务期间默认的文件存储位置，可自定义。
Command to run	**容器1~3：sleep**	指定容器启动时要执行的命令。
Arguments to pass to the command	**容器1~3：9999999**	指定传递给“Command to run”的参数。 sleep 9999999命令表示容器持续运行，直到超过9999999秒或被手动终止。该配置主要用于让容器保持活跃状态，防止容器在没有任务时自动退出。

4. 配置Persistent Volume Claim。该PVC会被挂载到所有容器中，为各容器提供持久化存储。

   单击“Add Volume”，选择“Persistent Volume Claim”，填入以下信息：

   - Claim Name：填写**步骤2**中集群创建的PVC名称。
   - Mount path：表示挂载路径，固定填写/root/.m2。

   图 1-14 配置 Persistent Volume Claim

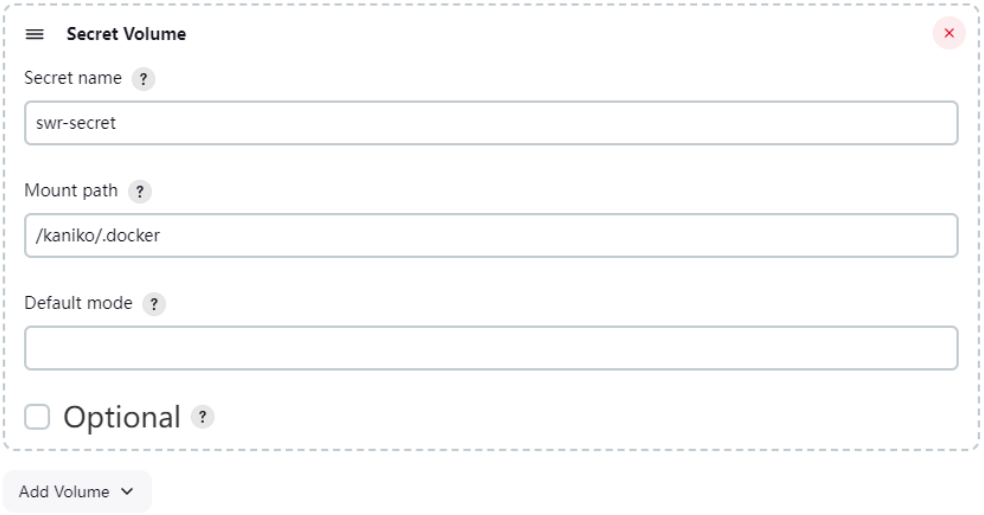
   

5. 配置Secret Volume。执行Pipeline作业时，该Secret作为kaniko容器向SWR中推送镜像的凭证。

   再次单击“Add Volume”，选择“Secret Volume”，填入以下信息：

   - Secret Name：填写**步骤3**中集群创建的Secret名称。
   - Mount path：表示挂载路径，固定填写/kaniko/.docker。

图 1-15 配置 Secret Volume



6. 配置拉取镜像的密钥，一般使用默认密钥default-secret。

图 1-16 配置 Image Pull Secret

7. 确保以上信息无误后，单击“Save”

----结束

1.3.3 在 Jenkins 界面中构建并执行 Pipeline

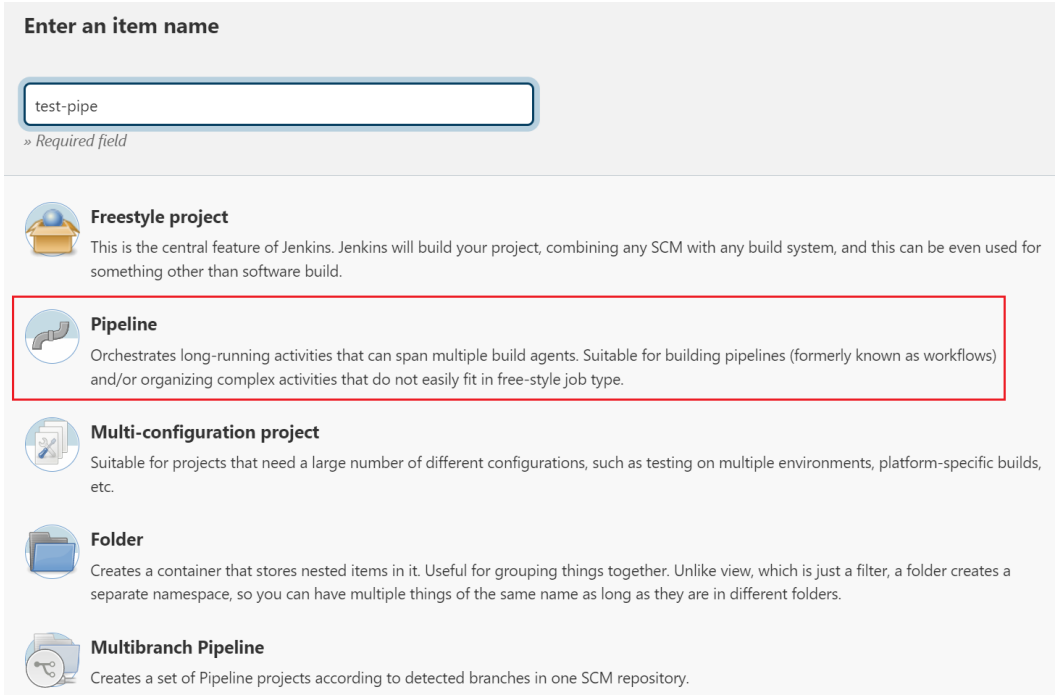
构建 Pipeline

本示例将使用Jenkins界面构建一个Pipeline。该Pipeline的内容是从代码仓中拉取代码，将代码打包成镜像，并推送到SWR镜像仓库中。

步骤1 在Jenkins Dashboard界面右侧导航栏单击“New Item”。

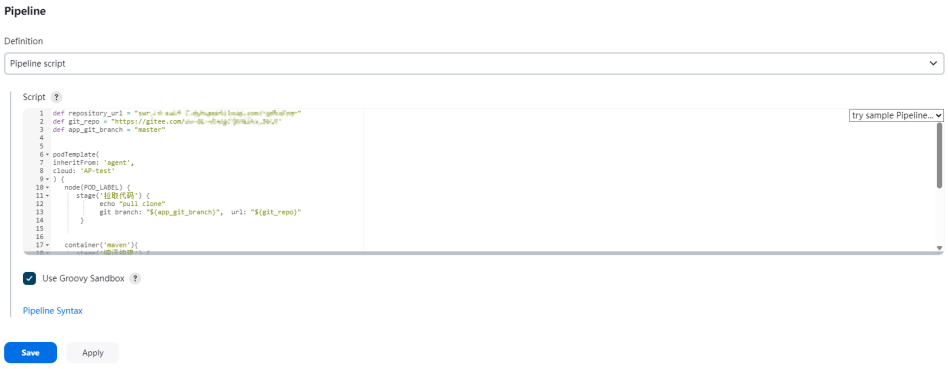
步骤2 输入任务名称，并选择创建Pipeline。

图 1-17 创建 Pipeline



步骤3 仅配置Pipeline脚本，其他保持默认。

图 1-18 配置 Pipeline 脚本



以下Pipeline脚本仅供您参考，您可根据自身业务自定义脚本内容，更多语法信息请参见**Pipeline**。

```
def swr_region = "cn-east-3"
def organization = "container"
def git_repo = "http://github.com/xxx.git"
def app_git_branch = "master"


podTemplate(
inheritFrom: 'jenkins-agent',    // 请替换为步骤5中创建的Pod Template名称
cloud: 'ap-test'    // 请替换为步骤4中创建的Cloud名称
) {
  // 拉取代码仓中的代码
  node(POD_LABEL) {
    stage('拉取代码') {
          echo "pull clone"
          git branch: "${app_git_branch}",  url: "${git_repo}"
```



```
    }

  // 利用maven容器将代码仓中拉取的代码打包（该打包方式仅适合Java，其他语言请更换打包方式）
  container('maven'){
    stage('代码打包') {
          echo "build package"
          sh "mvn clean package  -DskipTests"
    }
  }

  // 利用kaniko容器将打包后的代码推送至SWR中，并将镜像命名为tomcat
  container('kaniko'){
    stage('镜像推送') {
          echo "build images and push  images"
          sh "/kaniko/executor -f Dockerfile -c . -d swr.${swr_region}.myhuaweicloud.com/${organization}/
tomcat:${BUILD_ID} --force"
    }
   }
}
}
```

表 1-9 Pipeline 脚本参数说明

参数	示例	说明
swr_region	**cn-east-3**	表示SWR镜像仓库的区域。 **说明** 该SWR镜像仓库用于存放代码打包的镜像，区域需要与步骤3中的区域一致。
organization	**container**	表示SWR中的组织名称，该组织名称可以根据实际需求进行选择。
git_repo	**https://github.com/xxx.git**	表示代码存放的具体地址，即代码库地址。
app-git-branch	**master**	表示代码库的分支。

步骤4 脚本配置无误后，单击“Save”。

----结束

执行 Pipeline 并查看运行结果

执行Pipeline后，集群自动创建名为“pipe-xxx”的Pod实例，该Pod会根据“Pod Template”中的信息创建三个容器，分别是jnlp、kaniko和maven。该Pod会依次完成从代码仓拉取代码、将代码打包成镜像和将镜像推送到SWR镜像仓库的操作，完成后自动删除。

步骤1 右侧列表单击“Build Now”，开始执行Pipeline任务。

步骤2 返回**CCE控制台**，单击对应集群名称。左侧导航栏单击“工作负载”，选择“容器组”页签，可以看到Pipeline创建的Pod实例。

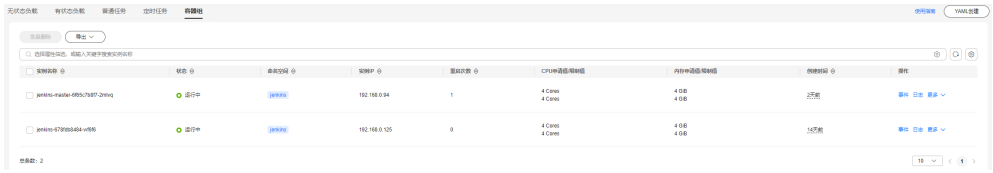
图 1-19 新建的 Pod 实例

步骤3 在新创建的Pod实例右侧单击“更多”，单击“容器列表”，可以看到按照“Pod Template”中的信息创建三个容器，即jnlp、kaniko和maven。

图 1-20 新建容器

步骤4 完成从代码仓拉取代码、将代码打包成镜像和将镜像推送到SWR镜像仓库的任务后，该Pod会自动删除，如图1-21所示。

图 1-21 Pod 自动删除



步骤5 检查SWR镜像仓库是否有新推送的镜像tomcat。

图 1-22 推送的镜像

----结束