云应用引擎

最佳实践

文档版本 01

发布日期 2025-01-17

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 CAE 最佳实践汇总

本文汇总了基于云应用引擎服务（CAE，Cloud Application Engine）常见应用场景的操作实践，为每个实践提供详细的方案描述和操作指导，帮助用户轻松构建基于CAE的应用托管业务。

表 1-1 CAE 最佳实践一览表

最佳实践	说明
使用CAE托管Nginx静态文件服务器	本章节介绍如何使用CAE托管Nginx静态文件服务器。您可以将业务代码部分制作镜像后部署到CAE，静态文件部分存储到与该组件关联的并行文件系统中，即可实现混合业务和静态文件的前端组件托管。
Gitlab对接Jenkins自动构建并部署到CAE	本章节以Java项目的构建、部署为例，介绍如何完成“完整的代码提交 > Jenkins构建 > 软件包上传/镜像上传 > CAE部署”流程。
Jenkins流水线支持多组件按照依赖顺序部署	本章节介绍如何使用jenkins来构建部署升级微服务组件，升级涉及多个微服务组件。
SpringCloud应用部署到CAE自动对接Nacos引擎	本章节通过CAE源码部署能力自动化部署一个provider服务和一个consumer服务，帮助您体验自动接入Nacos引擎。
Spring Cloud应用无损上线	在组件运维过程中，不可避免要进行升级、重启、扩容等操作，在这些操作中，无损上线是常见的要求。本章节介绍如何配置Spring Cloud无损上线。
健康检查	本章节介绍如何通过CAE提供三种健康检查机制（存活探针、就绪探针和启动探针）来检测您的应用实例是否正常工作，保障业务正常运行。
生命周期管理	生命周期管理是用于在特定阶段执行调用的方法。本章节介绍如何通过CAE提供的两种生命周期管理（启动后处理、停止前处理）来处理事件。
发送事件告警到企业微信	本章节介绍如何通过设置事件通知规则，可以帮助您及时了解组件运行时的状态，快速定位问题。

最佳实践	说明
对接软件开发生产线CodeArts流水线自动升级到CAE	CAE目前提供了Codearts商业插件，可对接Codearts流水线自动升级组件到CAE。本章节将根据不同场景为您介绍“CAE升级插件”的使用方法。
通过配置PromQL实现自定义弹性伸缩	假设某个组件提供自定义指标 http_requests_total（表示http请求总量），本章节以该指标为例，介绍如何使用PromQL实现自定义弹性伸缩。
配置CAE对接DEW，帮助应用从DEW获取加密凭据	本章节介绍如何配置CAE对接DEW，帮助应用从DEW获取加密凭据。
ASP.NET Core应用部署到CAE	CAE源码部署支持Docker运行时，您可以自行配置Dockerfile文件，在Dockerfile中安装构建环境，定义构建命令，以此支持更多编程语言的项目在CAE部署。 本章节以以ASP.NET Core应用为例，介绍如何将ASP.NET Core应用部署到CAE。
通过企业路由打通网络	CAE提供CAE环境访问租户侧VPC的能力，本章节将指导您在关闭公网后，如何通过企业路由打通网络。

2 使用 CAE 托管 Nginx 静态文件服务器

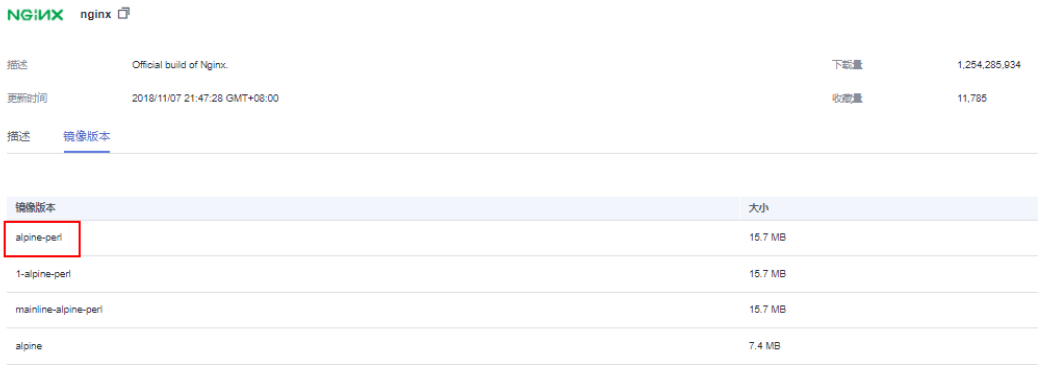
2.1 概述

应用场景

本服务适用于业务代码及静态页面混合WEB前端组件托管场景。用户可以将业务代码部分制作镜像后部署到CAE，静态文件部分存储到与该组件关联的并行文件系统中，即可实现混合业务和静态文件的前端组件托管。组件部署后，用户可以通过更新并行文件系统中的静态页面文件来实现实时更新前台应用。

此方案使用的Nginx版本为“alpine-perl”，此版本已在开源镜像提供。

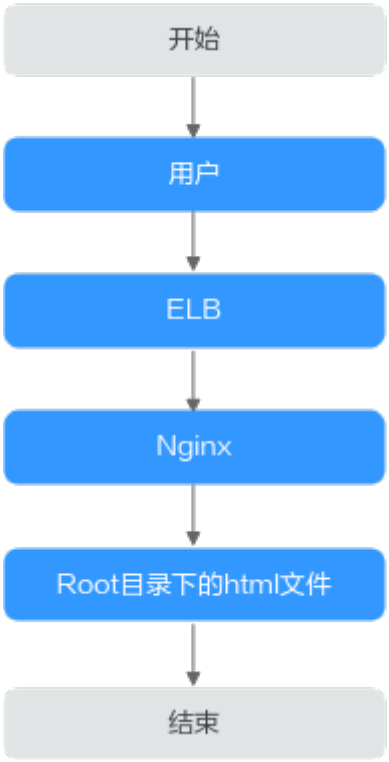
图 2-1 镜像版本



方案架构

Nginx是一个轻量级的web服务器，本身也是一个静态资源的http服务器。本实践以Nginx为例，通过配置云存储中的并行文件系统来实现静态文件的托管，并通过更新并行文件系统中的静态文件来实时更新Nginx访问页面。

图 2-2 Nginx 访问关系图



Nginx 默认配置

查询nginx默认配置方法：

Nginx默认配置如图2-3所示。

此版本的Nginx配置文件（ nginx.conf ）地址为“/etc/nginx/nginx.conf”。

图 2-3 nginx.conf

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

Nginx默认的default.conf如图2-4所示。“/usr/share/nginx/html”为静态文件存放的路径，该路径下存在两个默认文件分别为“50x.html”和“index.html”。

图 2-4 default.conf

```
server {
    listen       80;
    server_name  localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
```

如需替换页面显示，则将需替换的html文件放入“/usr/share/nginx/html”目录下。替换的html文件可以通过**获取静态文件**提供的链接中直接下载。

2.2 部署前准备

创建环境

步骤1 登录CAE控制台。

步骤2 选择以下任意方式创建环境。

- 在您首次使用本服务时，页面会提醒您尚未创建环境。
  a. 单击**创建环境**卡片中的“立即创建”。

图 2-5 创建环境

  b. 在弹出的对话框中输入对应的参数，具体参照**表2-1**。

表 2-1 创建环境

参数	说明
环境名称	输入自定义的环境名称。
企业项目	设置企业项目。 企业项目管理提供了一种按企业项目管理云资源的方式，帮助您实现以企业项目为基本单元的资源及人员的统一管理，默认项目为default。 **开通企业项目**后可以使用。
虚拟私有云	下拉框中选择环境资源所在VPC。 如需创建VPC，请单击“创建虚拟私有云”，具体操作参考**创建虚拟私有云**。 **说明** 环境创建完成后，不支持修改VPC。

参数	说明
子网	下拉框中选择环境子网。 无可用子网时，单击“创建子网”，进入网络控制台创建新子网，具体操作参考**为虚拟私有云创建新的子网**。 **说明** 子网需要保留至少2个可用网络IP地址，以供CAE配置和优化使用，如果不满足条件，会创建失败。
安全组	选择“自动生成”。 **说明** 安全组需要放通所选择的子网到子网网关地址，以及需要访问的中间件如RDS，CSE等服务的访问地址和端口。
组织	如果您是首次使用本服务，在该下拉框中单击“创建组织”，输入自定义的组织名称。

- 非首次使用本服务，选择“组件列表”。
  a. 单击页面上方环境模块右侧 。
  b. 在弹出的“新增环境”对话框中输入新增环境的名称。

  说明

  可直接使用已创建的环境。

步骤3 单击“确定”，完成创建环境。

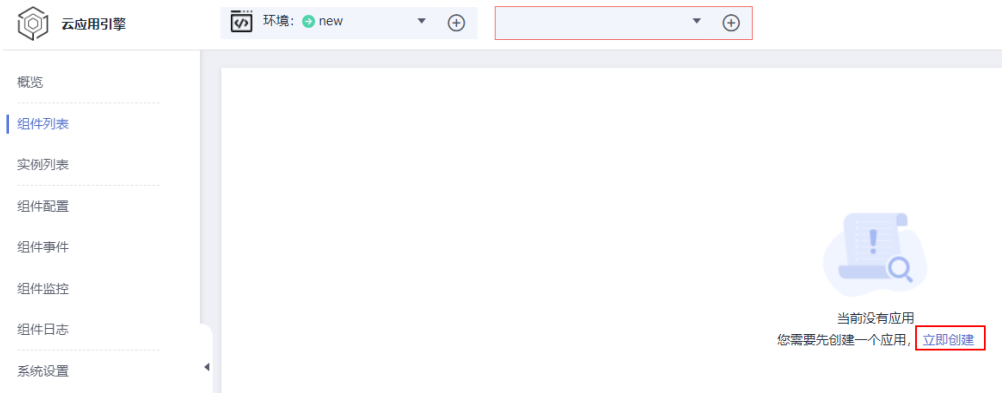
----结束

创建应用

步骤1 在导航栏中选择“组件列表”，选择以下方式创建应用。

- 单击页面上方应用模块右侧 。
- 单击“组件列表”页面中的“立即创建”。

图 2-6 创建应用



步骤2 在弹出的“新增应用”对话框中输入新增应用的名称。

步骤3 单击“确定”，完成创建应用。

----结束

获取静态文件

您可以通过以下链接下载“index.html”和“test.html”静态文件进行更新操作。

下载地址：**https://international-cae-demo.obs.ap-southeast-3.myhuaweicloud.com/nginx.zip**或**https://international-cae-demo.obs.ap-southeast-3.myhuaweicloud.cn/nginx.zip**

2.3 操作步骤

约束与限制

官方Nginx镜像使用nginx用户运行，您需要将挂载文件的文件掩码（umask）设置为0022，确保nginx用户拥有读取挂载文件的权限，指导操作请参考**配置云存储挂载路径**。

创建并部署 Nginx 组件

步骤1 登录CAE控制台。

步骤2 选择“组件列表”

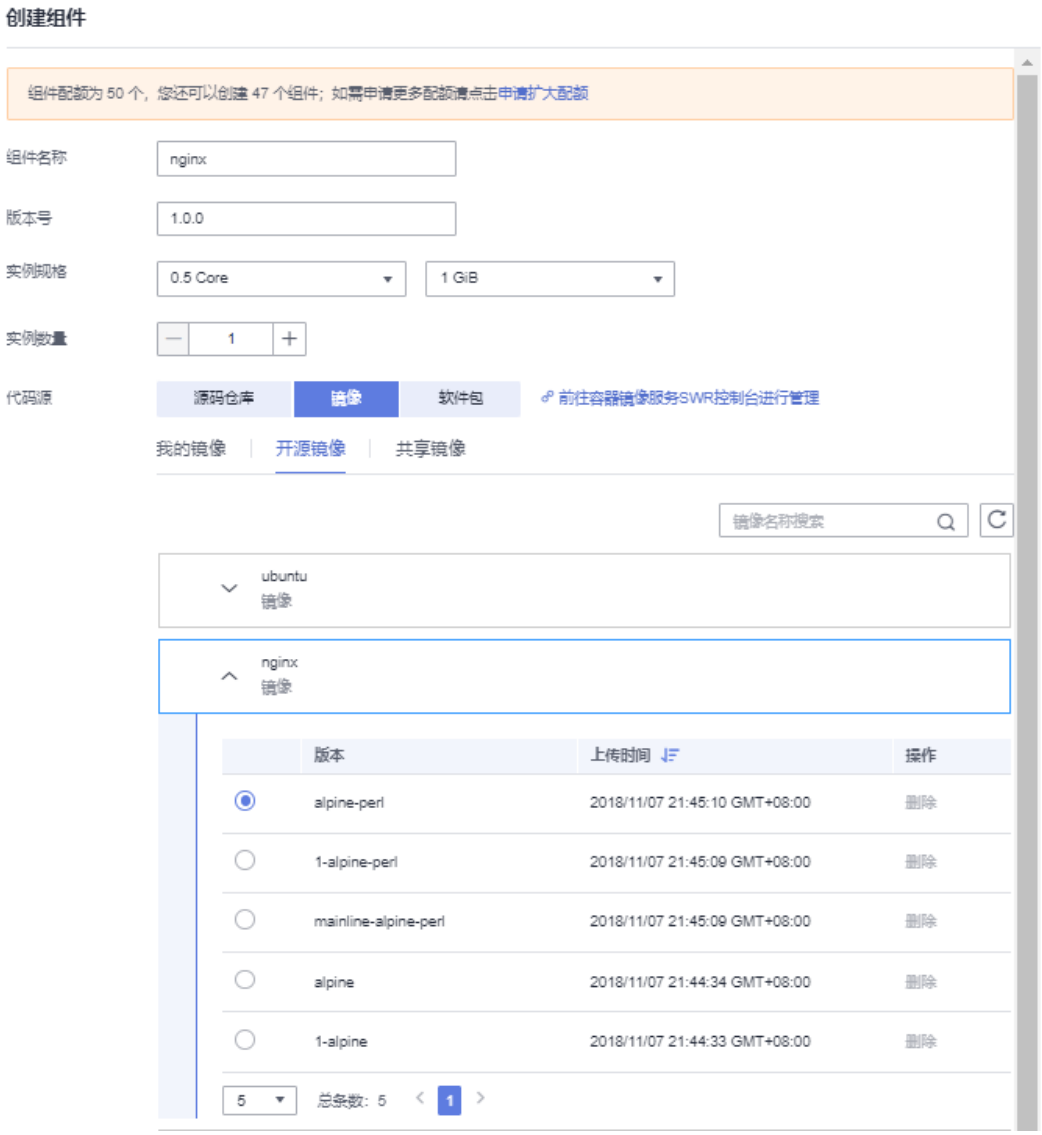
步骤3 在页面上方，下拉选择已创建的应用和环境，单击“新增组件”。

步骤4 参考**表2-2**设置组件信息。

表 2-2 组件基本信息

参数	说明
组件名称	新建组件的名称。本实践输入名称为“nginx”。
版本号	组件的版本号。 本实践版本号为1.0.0。
实例规格	选择实例规格，例如：0.5core、1GiB。
实例数量	输入实例数为1。
代码源	选择“镜像 > 开源镜像 > nginx”。此方案使用的nginx的版本为“alpine-perl”。

图 2-7 创建组件



步骤5 单击“配置组件”。

步骤6 在“组件配置”页面，单击“访问方式”模块的“编辑”，对Nginx组件进行配置。

步骤7 在“负载均衡配置”页签下，单击“添加负载均衡配置”并设置参数。

- 负载均衡器：选择“内置负载均衡器”。
- 健康检查：使用默认值“启动”。
- 访问控制：使用默认值“允许所有IP访问”。
- 协议：选择“TCP”。
- 监听端口：输入“80”。
- 访问端口：自定义访问端口。本实践设置为“14632”。

图 2-8 添加负载均衡外网访问

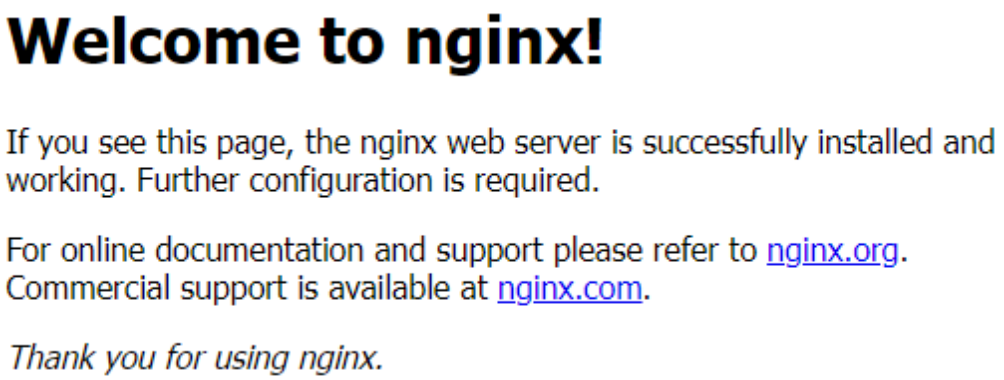
步骤8 单击“确定”，完成对该组件访问方式的配置。

步骤9 在“组件配置”页面，单击“配置并部署组件”，使Nginx组件访问方式生效。

步骤10 部署成功后，在左侧导航栏中选择“组件列表”，进入“组件列表”页面。

单击对应组件“访问地址”列的ip地址，查看Nginx静态WEB页面。

图 2-9 访问静态页面



----结束

云存储授权

步骤1 选择“系统设置 > 云存储授权”，单击“编辑”，打开“已授权对象存储”弹框。

步骤2 单击“授权并行文件系统”，进入“授权并行文件系统”页面。

图 2-10 授权并行文件系统

步骤3 单击“创建并行文件系统”，在输入框中输入自定义名称，例如：test-nginx。

步骤4 单击“确认”。

步骤5 选择已创建的并行文件系统。您还可以通过列表上方搜索框搜索关键词筛选数据。

步骤6 单击“授权”。

在访问密钥授权弹框，输入访问密钥ID（AK）和秘密访问密钥（SK），并单击“确认”。

可单击“获取访问密钥”，获取访问密钥AK/SK，具体操作请参考**访问密钥**。

说明

如果您的环境是2024年4月20日前创建的，直接跳过此步骤。

步骤7 当页面显示“授权成功”，可在“已授权云存储”页面查看授权成功的并行文件系统。

----结束

上传文件

步骤1 登录OBS控制台。

步骤2 在左侧导航栏选择“并行文件系统”，单击名称为“test-nginx”的文件系统，进入该并行系统页面。

图 2-11 并行文件系统

步骤3 在左侧导航栏选择“文件”，单击“上传文件”。

步骤4 将**已获取的静态文件**“index.html”、“test.html”拖拽至文件上传处，也可单击“添加文件”上传。存储类别默认为“标准存储”。

图 2-12 上传文件

步骤5 单击“上传”，完成文件上传。

----结束

配置云存储挂载路径

步骤1 返回CAE控制台，选择“组件配置”。

步骤2 在“组件配置”页面上方的下拉框中选择Nginx组件。

步骤3 单击“云存储配置”模块中的“编辑”，进入云存储配置页面。

图 2-13 配置云存储

步骤4 在“云存储配置”页面单击“配置并行文件系统”，并输入挂载路径并设置权限。

- 并行文件系统名称：选择**云存储授权**中授权的并行文件系统“test-nginx”。
- 文件掩码（umask）：文件掩码设置为0022。
- 挂载路径：数据存储挂载到组件上的路径。本实践使用“nginx”默认路径“/usr/share/nginx/html”。
- 权限：挂载路径及挂载路径下文件，有“读写”、“只读”两种权限，此处选择“读写”。

图 2-14 配置云存储挂载路径

说明

- 挂载静态文件路径到云存储上时需要注意：请不要挂载到含有系统文件的目录下，如“/”、“/var/run”等，否则可能会导致部署的组件异常。
- 挂载路径选择权限时需要注意：读写权限表示组件对于该挂载路径及路径下的所有文件拥有读写权限，只读即只有只读权限。

步骤5 单击“确定”后在“云存储配置”页面再次单击“确定”，完成配置。

步骤6 在“组件配置”页面单击“生效配置”。

图 2-15 生效配置

----结束

查看更新页面

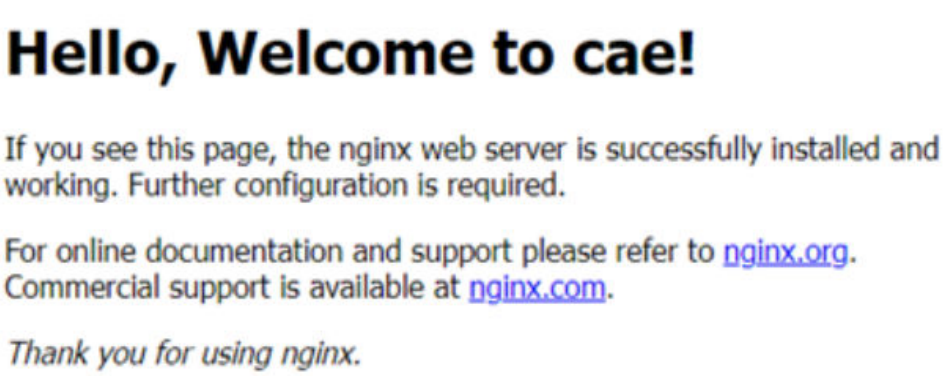
步骤1 在左侧导航栏中选择“组件列表”，进入“组件列表”页面。

步骤2 单击对应Nginx组件“访问地址”列的ip地址，再次访问Nginx，可以实时更新页面（单击F5刷新页面）。

说明

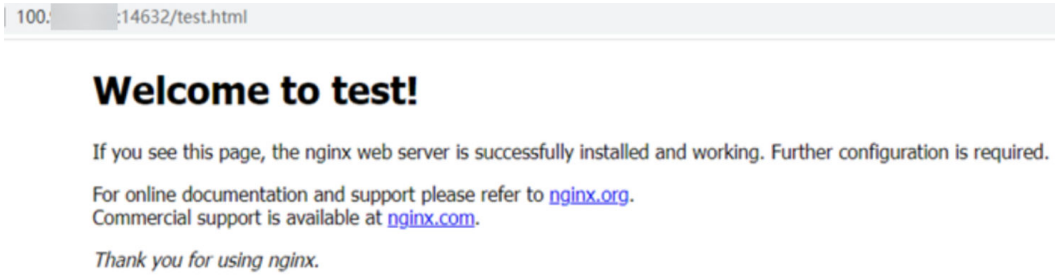
新的“index.html”文件可以根据您的需求自定义并上传。

图 2-16 更新后静态页面



您还可以通过/test.html访问新的静态页面文件。

图 2-17 访问 test 页面



----结束

3 Gitlab 对接 Jenkins 自动构建并部署到 CAE

3.1 概述

背景

完成代码开发后，每次上线前需先在Jenkins上打包成镜像或软件包，再将镜像手动上传到swr或者将软件包手动上传到obs，然后去CAE升级组件。该流程较为繁琐，频繁发版测试导致开发和运维效率低，体验差，提供如下最佳实践提高开发效率。下面以Java项目的构建、部署为例提供了完整的代码提交>>Jenkins构建>>软件包上传/镜像上传>>CAE部署的指导，如涉及其他语言如Golang等只需对应安装编译构建环境即可。

适用场景

此功能适用于您的代码已在Gitlab上管理，通过Jenkins构建打包，使用CAE进行应用托管， 并且已经部署了组件之后，需要进行组件升级的场景。

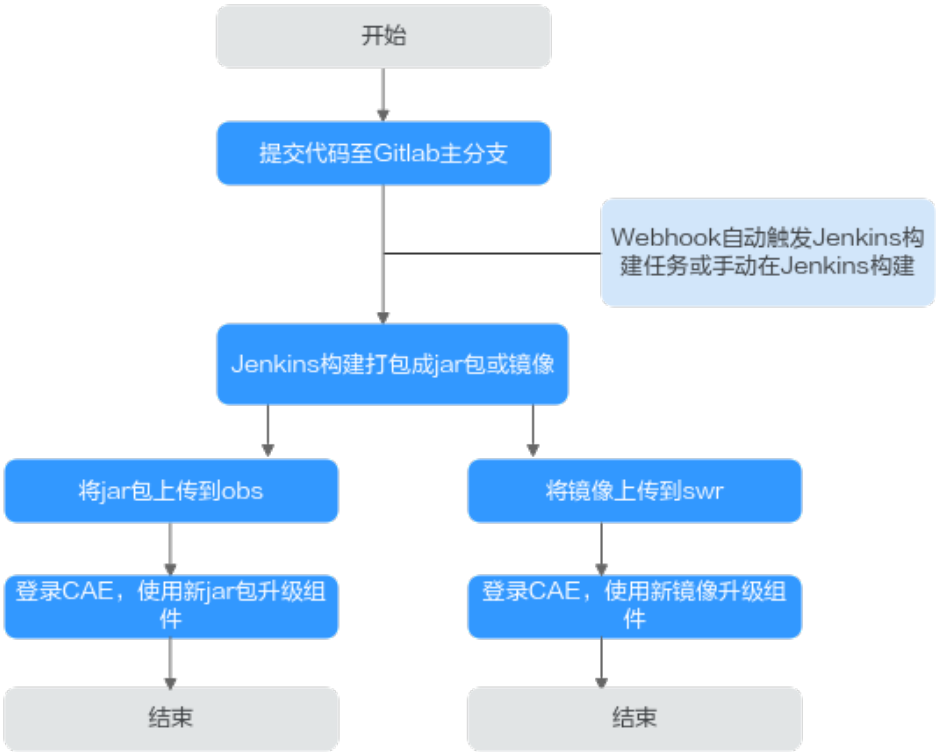
第一次部署请参考**创建组件**。

解决方案

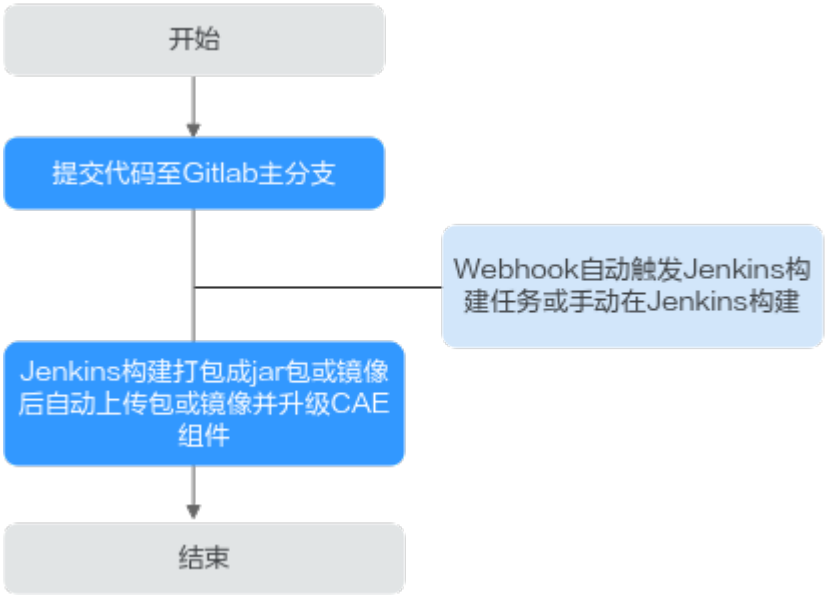
提供上传jar包和镜像并升级CAE组件的方案，输出一个shell脚本在Jenkins构建打包完成之后调用脚本自动部署到CAE环境中，实现代码合入后自动构建打包部署。

流程优化对比

当前上线流程图如下：



使用该方案后的流程图如下：



3.2 环境准备

3.2.1 准备 Jenkins 环境

环境信息说明

说明

如果已安装好jenkins环境，请跳过本章节。

在linux虚拟机上安装好Jenkins，本实践使用的具体环境信息如下所示。如果使用镜像部署，需要在虚拟机中安装docker。

- 虚拟机：Centos7.9
- Jenkins：2.331
- git：yum安装
- jdk：11.0.8
- Apache Maven ： 3.8.6

说明

部署的Jenkins启动时需添加如下参数：

```
-Dhudson.security.csrf.GlobalCrumbIssuerConfiguration.DISABLE_CSRF_PROTECTION=true
```

否则Gitlab对接Jenkins会失败，报错信息如下：

```
HTTP Status 403 - No valid crumb was included in the request
```

相关软件下载及安装

- Jenkins下载链接：
  https://mirrors.jenkins.io/war-stable/
- 安装git用于拉取代码进行构建：
  ```
  yum install git –y
  ```
- jdk安装包下载链接：
  https://www.oracle.com/cn/java/technologies/downloads/#java11
- maven安装包下载链接：
  https://maven.apache.org/download.cgi
- docker安装用于打包镜像并上传到镜像仓库：
  ```
  yum install docker
  ```

安装后检查

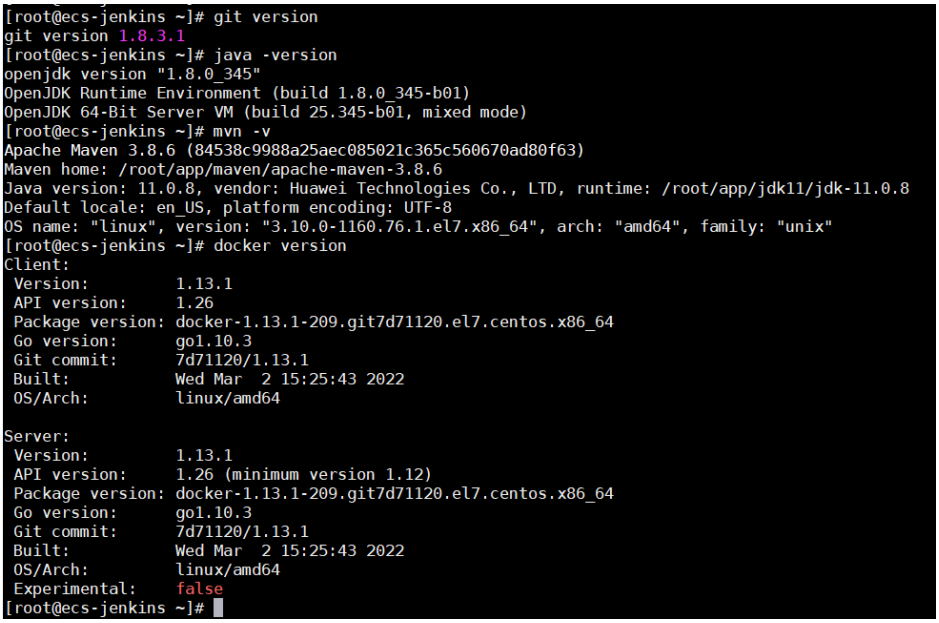
- 检查git：
  - 检查git：
    ```
    [root@ecs-jenkins ~]# git version
    git version 1.8.3.1
    ```
  - 检查JDK：
    ```
    [root@ecs-jenkins jar]# java -version
    openjdk version "1.8.0_345"
    OpenJDK Runtime Environment (build 1.8.0_345-b01)
    OpenJDK 64-Bit Server VM (build 25.345-b01, mixed mode)
    ```
  - 检查Maven：
    ```
    [root@ecs-jenkins jar]# mvn -v
    Apache Maven 3.8.6 (84538c9988a25aec085021c365c560670ad80f63)
    Maven home: /root/app/maven/apache-maven-3.8.6
    ```

```
Java version: 11.0.8, vendor: Huawei Technologies Co., LTD, runtime: /root/app/jdk11/jdk-11.0.8
Default locale: en_US, platform encoding: UF-8
OS name: "linux", version: "3.10.0-1160.76.1.el7.x86_64", arch: "amd64", family: "unix"
```

- 检查Docker：

```
[root@ecs-jenkins jar]# docker version
Client:
 Version:         1.13.1
 API version:     1.26
 Package version: docker-1.13.1-209.git7d71120.el7.centos.x86_64
 Go version:      go1.10.3
 Git commit:      7d71120/1.13.1
 Built:           Wed Mar  2 15:25:43 2022
 OS/Arch:         linux/amd64
Server:
 Version:         1.13.1
 API version:     1.26 (minimum version 1.12)
 Package version: docker-1.13.1-209.git7d71120.el7.centos.x86_64
 Go version:      go1.10.3
 Git commit:      7d71120/1.13.1
 Built:           Wed Mar  2 15:25:43 2022
 OS/Arch:         linux/amd64
 Experimental:    false
```

图 3-1 版本查验



3.2.2 上传代码到 Gitlab 代码仓库

本实践使用的是Java项目代码，使用Maven构建Jar包。

前提条件

1. Jenkins所在Linux虚拟机能够访问GitLab代码仓库。
2. 已经在GitLab创建账号和仓库。

操作步骤

步骤1 登录GitLab。

步骤2 上传代码到已创建好的代码仓库。

----结束

3.2.3 安装 obsutil 工具

说明

如果不涉及软件包部署，请跳过本章节。

前提条件

- 获取AK/SK，请参考**AK/SK获取方法**。
- 已获取部署组件的CAE所在区域的终端节点，参考**地区和终端节点**。
- 已在和部署组件的CAE在同一区域的OBS中创建桶，用于存储软件包，具体操作请参见**创建桶**。示例选择的桶名为cae-obs。

下载和安装

- 操作系统的选择：
  下载安装前在Jenkins所在虚拟机中执行命令查看虚拟机操作系统类型：
  ```
  echo $HOSTTYPE
  ```
  - 若执行如上命令的输出值是“x86_64”，请下载AMD 64位系统；
  - 若执行如上命令的输出值是“aarch64”，请下载ARM 64位系统。
- obsutil工具用于构建后上传软件包到obs，请参见**下载和安装obsutil**。

初始化配置

执行以下配置命令，初始化配置obsutil工具。

```
{path}/obsutil config -i=ak -k=sk -e={endpoint}
```

其中：

- {path}需要替换为obsutil安装路径，例如：/root/tools/obsutil/obsutil_linux_amd64_5.4.6。
- {endpoint}需要替换为已获取到的部署组件的CAE所在区域的终端节点。

验证 obsutil 上传文件到 OBS 是否正常

步骤1 创建测试文件。例如：test.txt。

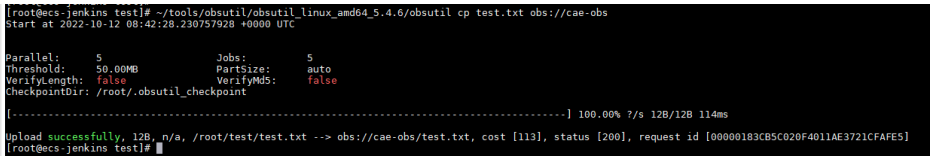
```
touch test.txt
```

步骤2 使用obsutil上传创建的文件到obs。

```
~/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil cp test.txt obs://{OBS桶名称}
```

请将{OBS桶名称}替换为已创建的待使用的OBS桶名称，本示例选择的桶名为cae-obs，将在当前目录新建的test.txt文件上传到cae-obs桶中。提示“Upload successfully”表示上传成功。

图 3-2 上传文件到 obs



步骤3 登录OBS控制台，选择“对象存储”。

步骤4 单击本示例桶名名称cae-obs，进入“对象”页面，查看已经上传的文件test.txt。

----结束

3.2.4 安装 KooCLI 工具

KooCLI工具用于调用CAE服务提供的接口，对CAE组件执行升级等操作。

使用KooCLI工具之前，您需要先安装和初始化配置KooCLI工具：

- 安装KooCLI：您可以选择**方式一：联网安装**或者**方式二：软件包安装**安装KooCLI工具。
- **初始化配置KooCLI**：使用KooCLI工具前，需要先进行初始化配置。

方式一：联网安装

步骤1 登录Jenkins所在虚拟机.

步骤2 执行安装命令：

```
curl -sSL https://hwcloudcli.obs.cn-north-1.myhuaweicloud.com/cli/latest/hcloud_install.sh -o ./hcloud_install.sh && bash ./hcloud_install.sh -y
```

说明

如上命令默认将KooCLI下载至“/usr/local/hcloud/”目录下，同时在“/usr/local/bin/”目录下创建KooCLI的符号链接。

----结束

方式二：软件包安装

步骤1 登录Jenkins所在虚拟机，执行如下命令确认所需安装操作系统：

```
echo $HOSTTYPE
```

- 若执行如上命令的输出值是“x86_64”，请下载AMD 64位系统；
- 若执行如上命令的输出值是“aarch64”，请下载ARM 64位系统。

步骤2 执行如下命令下载对应的软件包。

- AMD

  ```
  wget "https://hwcloudcli.obs.cn-north-1.myhuaweicloud.com/cli/latest/huaweicloud-cli-linux-amd64.tar.gz" -O huaweicloud-cli-linux-amd64.tar.gz
  ```

- ARM

  ```
  wget "https://hwcloudcli.obs.cn-north-1.myhuaweicloud.com/cli/latest/huaweicloud-cli-linux-arm64.tar.gz" -O huaweicloud-cli-linux-arm64.tar.gz
  ```

步骤3 执行如下命令解压软件包。

- AMD

  ```
  tar -zxvf huaweicloud-cli-linux-amd64.tar.gz
  ```

- ARM

  ```
  tar -zxvf huaweicloud-cli-linux-arm64.tar.gz
  ```

步骤4 在解压后的目录执行如下命令创建软链接到“/usr/local/bin”目录：

```
ln -s $(pwd)/hcloud /usr/local/bin/
```

步骤5 执行如下命令验证是否安装成功：

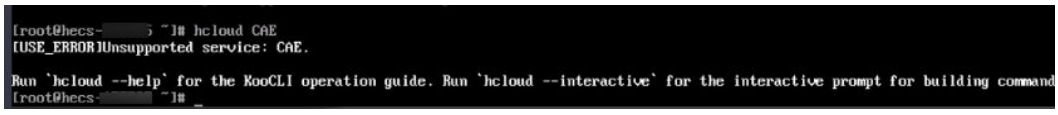

```
hcloud version
```

系统显示类似“当前KooCLI版本:3.4.1.1”版本信息，表示安装成功。

----结束

验证安装结果

步骤1 KooCLI工具安装完成后，请执行**hcloud CAE**命令测试是否支持CAE服务。



步骤2 如果出现**Unsupported service: CAE**，则需要将KooCLI工具切换成中文。

```
hcloud configure set --cli-lang=cn
```

步骤3 切换完成后，再次执行**hcloud CAE**命令。

步骤4 如果没有出现ERROR，则表示KooCLI工具已支持CAE服务。

----结束

初始化配置 KooCLI

步骤1 登录Jenkins所在Linux虚拟机。

步骤2 执行命令进行初始化配置，输入命令后按回车进入交互模式，根据界面提示输入各参数值，各参数配置参考**表3-1**。

```
hcloud configure init
```

表 3-1 初始化配置

参数	说明
Access Key ID	（必填参数）访问密钥ID，即AK。获取方法，请参考**访问密钥**。
Secret Access Key	（必填参数）与访问密钥ID（AK）结合使用的密钥，即SK，初始化时必填。获取方法，请参考**访问密钥**。
Region	（选填参数）区域，即CAE服务部署区域。获取方法，请参考**地区和终端节点**。

----结束

3.2.5 安装插件并配置 Jenkins 工具

在使用GitLab使用Gitlab对接Jenkins自动构建并部署组件到CAE前，需要安装Jenkins插件和并配置Jenkins全局参数。

- **安装Jenkins插件**：用于对接git以及支持在构建的时候使用脚本。
- **Jenkins全局工具配置**：用于Jekins流水线打包脚本对接git拉取代码并打包。

安装 Jenkins 插件

步骤1 在浏览器地址栏输入http://{安装Jekins的Linux虚拟机IP}:8080，登录Jenkins。

步骤2 选择“系统管理 > 插件管理”。

步骤3 单击“可选插件”，搜索表3-2中的插件进行安装。

表 3-2 插件安装说明

插件名称	是否必须	说明
Generic Webhook Trigger Plugin	是	用于对接Gitlab的webhook
GitLab Plugin	是	允许Gitlab触发Jenkins构建
Pipeline: Basic Steps	是	支持pipeline脚本语法
Pipeline: Build Step	是	支持pipeline脚本语法
Pipeline: Stage Step	是	支持pipeline脚本语法

----结束

Jenkins 全局工具配置

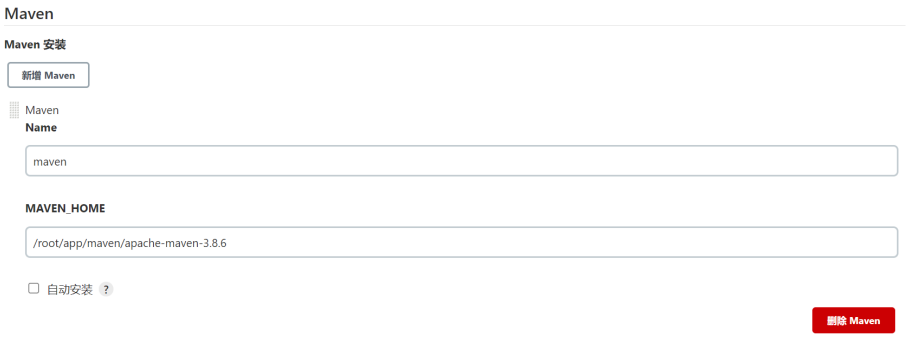
步骤1 选择“系统管理 > 全局工具配置”。

步骤2 配置maven。

示例中的maven安装目录“/root/app/maven/apache-maven-3.8.6”，请获取您的实际Maven安装目录。

图 3-3 Maven 配置

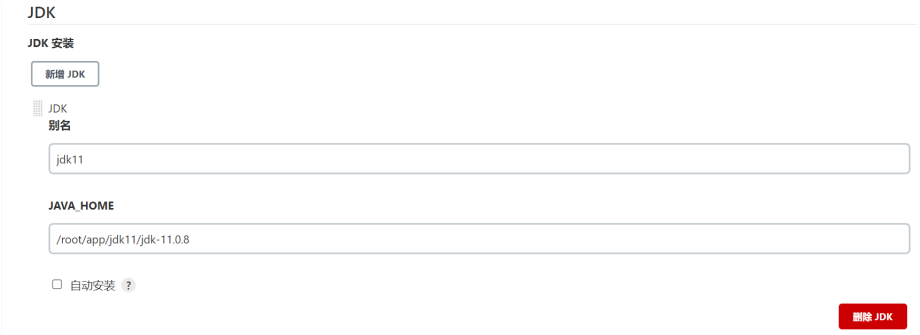
图 3-4 Maven 安装



步骤3 配置JDK。

示例中的JDK安装目录“/root/app/jdk11/jdk-11.0.8”，请获取您的实际JDK安装目录。

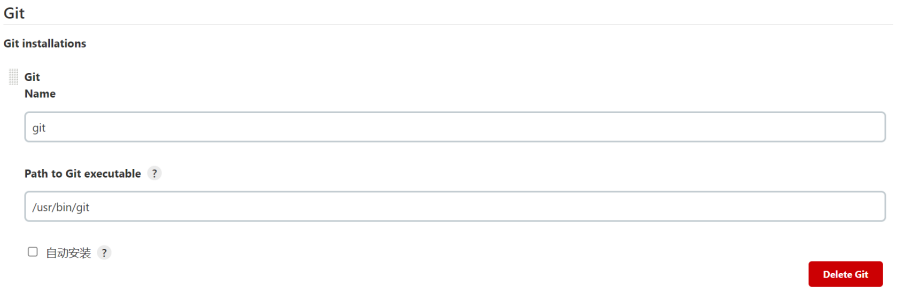
图 3-5 JDK 安装



步骤4 配置Git。

示例中的Git工具目录“/usr/bin/git”，请获取您的实际Git安装目录。

图 3-6 Git 安装



----结束

3.3 操作步骤

3.3.1 操作前对接测试

说明

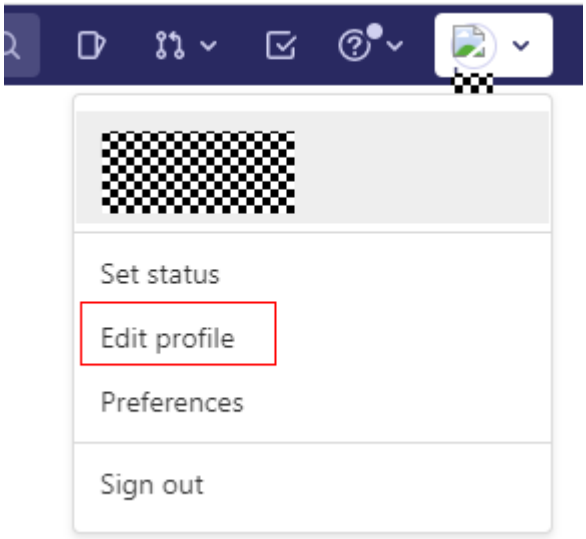
操作前需进行Jenkins对接Gitlab测试，保证Jenkins通过API访问Gitlab没有问题。

生成 Gitlab 访问令牌

步骤1 登录Gitlab。

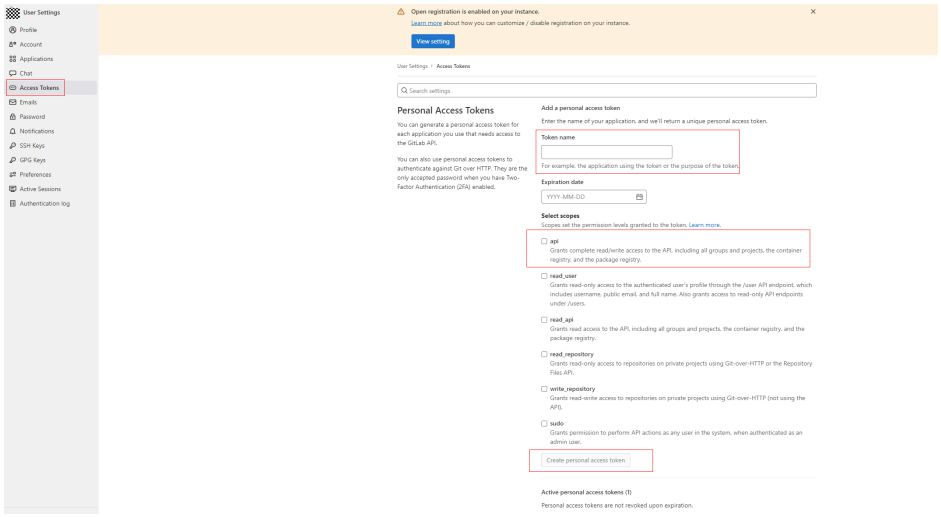
步骤2 鼠标移动到右上角的账号名上，单击“Edit profile”。

图 3-7 进入编辑页面



步骤3 单击“Access Tokens”，输入“Token name”，勾选“api”，单击“Create personal access token”创建访问令牌。

图 3-8 创建访问令牌

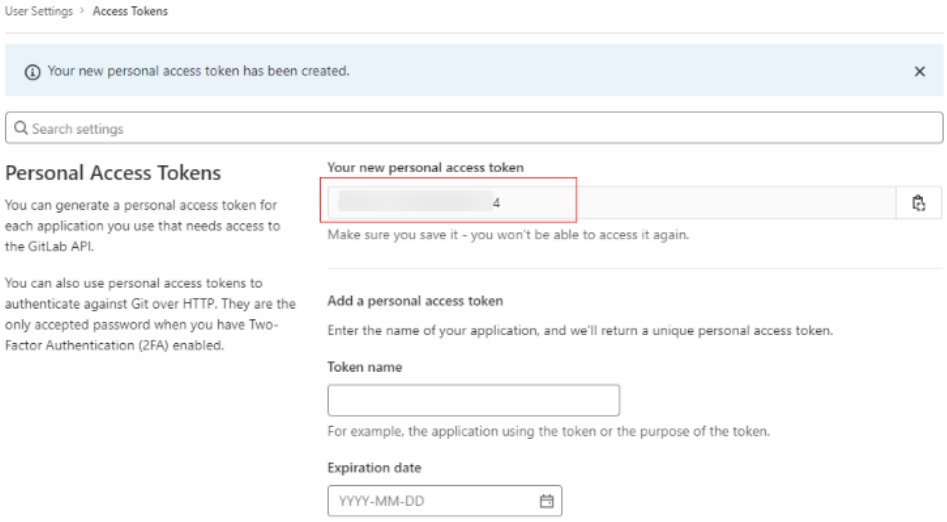


完成后在页面上方的“Personal Access Tokens”右侧显示token令牌。

说明

令牌仅在初次生成时显示，否则下次需要重新创建。该令牌仅用于Gitlab对接测试。

图 3-9 令牌显示



----结束

Jenkins 对接 Gitlab 测试配置

步骤1 在浏览器地址栏输入http://{安装Jekins的Linux虚拟机IP}:8080，登录Jenkins。

步骤2 选择“系统管理 > 系统配置”，在配置中选择“Gitlab”。

步骤3 配置Gitlab的url，并单击Credentials下方的“添加”，选择“Jenkins”。

步骤4 在下拉框单击“Username with password”，选择“Gitlab API token”，将**生成Gitlab访问令牌**中Gitlab的访问令牌配置到API token中。

步骤5 Credentials选择“Gitlab API token”，单击“Test Connection”，返回“Success”表示成功。

----结束

3.3.2 配置流水线构建任务

- 场景一：使用Jenkins构建生成的是软件包，如Jar包，就使用脚本中的软件包部署场景，软件包部署会将构建出来的软件包上传到OBS桶中并升级CAE组件。
- 场景二：使用Jenkins构建生成的是镜像包，就使用脚本中的镜像部署场景，镜像部署会将构建出来的镜像包上传到SWR镜像仓库中并升级CAE组件。

本章节以**配置流水线脚本**中的实例为Jar包的场景进行说明。

创建 Gitlab 凭证

使用具有Gitlab代码仓库权限的账号密码在Jenkins中创建凭证，用于拉取Gitlab代码。

步骤1 在浏览器地址栏输入http://{安装Jekins的Linux虚拟机IP}:8080，登录Jenkins。

步骤2 选择“系统管理 > 系统配置”，在配置中选择“Gitlab”。

步骤3 单击“Credentials”下方的“添加”，选择“Jenkins”。

步骤4 配置Gitlab账号密码，单击“添加”，保存配置。

步骤5 选择“系统管理 > Manage Credentials”，查看配置的凭据。

唯一标识在**配置流水线脚本**中会用到。

----结束

创建流水线任务

步骤1 在浏览器地址栏输入http://{安装Jekins的Linux虚拟机IP}:8080，登录Jenkins。

步骤2 单击“新建任务”。

步骤3 输入任务名称，本示例使用：test-demo，选择“流水线”，单击“确定”。

----结束

配置构建触发器

这里介绍两种构建方式：

1. 在Jenkins中手动触发构建，手动单击任务右边的“立即构建”从而触发流水线任务。
2. 通过Gitlab提交代码后自动触发Jenkins构建，这种方式同时支持手动触发构建。

此处以第二种方式为例。

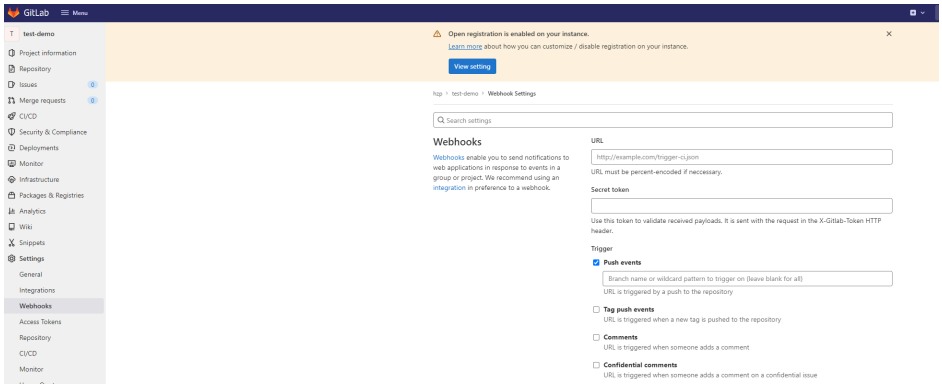
步骤1 配置Jenkins构建触发器。

1. 勾选“Build when a change is pushed to GitLab”，保存图示中的GitLab webhook URL（配置Gitlab webhook时需使用），然后单击右下角“高级”。
2. 选择“Filter branches by regex ”，配置指定分支变更后触发构建任务，示例中的分支名称为main，单击右下角“Generate”生成Secret token并保存，在配置Gitlab webhook时需使用。

步骤2 配置Gitlab webhook

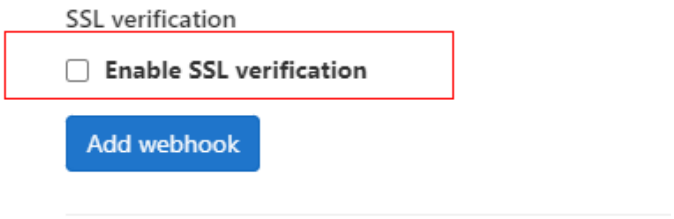
1. 登录Gitlab，进入代码仓库，示例中的仓库名称是“test-demo”。
2. 选择settings中的“Webhooks”，URL和Secret token填写**步骤1**获取到的GitLab webhook URL和Secret token。

图 3-10 webhooks 配置



3. 取消勾选SSL verification的“Enable SSL verification”，单击“Add webhook”。

图 3-11 完成 webhook 配置



----结束

配置流水线脚本

流水线脚本是构建时运行的构建命令，脚本参数说明见**表3-3**。

表 3-3 流水线脚本参数说明

参数	是否必须	参数类型	描述
git_url	是	String	Gitlab代码仓库地址。
credentials_id	是	String	使用账号密码配置的Gitlab凭据id，参考**创建Gitlab凭证**。
branch_name	是	String	Gitlab代码仓库分支名称。
maven	是	String	maven安装的可执行文件路径，示例：/root/app/maven/apache-maven-3.8.6/bin/mvn。
deploy_shell	是	String	deploy.sh脚本在Jenkins所在虚拟机上存放的路径，示例：/root/jar/deploy.sh，内容请参见**deploy.sh脚本说明**。
build_target_name	是	String	构建产物名称：软件包名称或镜像名称:版本号，通过执行脚本时传入该参数，软件包部署场景为软件包名称，镜像部署场景为构建出来的镜像名称:版本号。

步骤1 完成“构建触发器”配置之后，在“流水线”页签，在下拉框选择“Pipeline script”。

步骤2 配置流水线脚本，示例中使用的是构建jar包场景，脚本如下：

```
node {
    // 定义代码仓地址
    def git_url = 'http://100.**.**.207:8090/test/test-demo.git'
    // Gitlab凭据id
    def credentials_id = '133b7c9a-eb6a-4484-84b3-c3509ed63df8'
    // git代码仓库分支名称
```

```
    def branch_name = 'main'
    // maven安装的可执行文件路径
    def maven = '/root/app/maven/apache-maven-3.8.6/bin/mvn'
    // deploy.sh 脚本存放路径，需要设置可执行权限
    def deploy_shell = '/root/jar/deploy.sh'
    // 构建产物名称：软件包名称或镜像名称，必须参数，通过执行脚本时传入该参数
    def build_target_name = "cae-demo-1.0-SNAPSHOT.jar"

    stage('Clone sources') {
        git branch: branch_name, credentialsId: credentials_id, url: git_url
    }
    stage('Build') {
        // 构建jar包
        sh "'$maven' clean package -Dmaven.test.failure.ignore=true"
    }
    stage('deploy') {
        // 执行脚本，使用构建产物升级CAE组件，超时时间5分钟
sh "timeout 300s '$deploy_shell' '$build_target_name'"
    }
}
```

说明

- 流水线脚本运行时会调用deploy.sh，该脚本详细说明参见**deploy.sh脚本说明**。
- 设置脚本文件deploy.sh为可执行文件。

----结束

3.3.3 deploy.sh 脚本说明

使用场景说明

- 场景一：使用Jenkins构建生成的是软件包，如jar包，就使用脚本中的软件包部署场景，软件包部署会将构建出来的软件包上传到obs桶中，再使用新的jar包去升级CAE组件。
- 场景二：使用Jenkins构建生成的是镜像，就使用脚本中的镜像部署场景，镜像部署会将构建出来的镜像上传到swr镜像仓库中，再使用新的镜像去升级CAE组件。

脚本内容

```
#!/bin/bash

#------------ 必填参数 ------------------
# region名称
region=''
# 项目id
project_id=''
# 环境名称
environment_name=''
# 应用名称
application_name=''
# 组件名称
component_name=''
# 部署类型: 1. software 2. image    software表示软件包部署， image表示镜像部署
deploy_type=''

#------------ 软件包部署场景必填参数 ---------------
# obsutil安装的可执行文件绝对路径
obsutil=''
# 存放软件包的obs桶名称
bucket_name=''
# 软件包在obs桶中的存放目录，默认是根目录，目录需要以/结尾，如果obs桶中没有这个目录，会自动创建出该目录
bucket_dir='/'
```

```
#------------ 镜像部署场景必填参数 ------------------
# swr组织名称
swr_organization=''
# AK 用于登录swr镜像仓库
AK=''
# swr登录密钥 用于登录swr镜像仓库
login_secret=''

#------------ 外部传入参数，不需要填，参考配置流水线构建任务中的流水线脚本参数build_target_name
------------------
# 软件包部署场景：构建出来的软件包名称，由外部build_target_name参数传入
software_package=''
# 镜像部署场景：构建后生成的镜像：镜像名称:版本名称，由外部build_target_name参数传入
machine_image_name=''

#必须传入一个参数，是软件包名称或镜像名称
#参数数量为1时，设置软件包名称或镜像名称
if [ $# == 1 ]; then
    if [[ "$deploy_type" == "software" ]]; then
        software_package=$1
    elif [[ "$deploy_type" == "image" ]]; then
        machine_image_name=$1
    fi
else
    echo "The number of external input parameters is not 1. Configure the pipeline script parameter
build_target_name in the pipeline build task."
fi

# ----------------   以下代码不用改动   --------------------
enterprise_project_id='0'
environment_id=''
application_id=''
component_id=''

function splitVersion() {
    str=$1
    right=${str##*.}
    left=${str%.*}
    if [ ${#str} -eq ${#left} ]; then
        left=""
    fi
}

function addVersion() {
    v1=$1
    res=`echo $v1|awk '{print int($0)}'`
    if [ $res -gt 9998 ]; then
        res=0
    else
        res=$(($res+1))
    fi
}

# 升级版本号
# 版本号格式为：A.B.C或 A.B.C.D  A、B、C、D均为整数，每个整数不大于9999
# 示例：1.0.0 -> 1.0.1；1.0.9999 -> 1.1.0
function upgradeVersion() {
    left=$1
    start=0
    end=''
    res=0
    while ( [[ $res = 0 ]] && [ $start -lt 4 ] )
    do
        start=$(($start+1))
        splitVersion $left
        addVersion $right
        if [ $start -gt 1 ]; then
            end=`echo "$res.$end"`
```

```
        else
            end=`echo "$res$end"`
        fi
    done
    if [ ${#left} -ne 0 ]; then
        version="$left.$end"
    else
        version="$end"
    fi
}

#获取组件升级的版本号
function upgradeComponentVersion() {
    # 查询组件信息
    component_detials=`/usr/local/bin/hcloud CAE ShowComponent --cli-region="$region" --
project_id="$project_id" --X-Enterprise-Project-ID="$enterprise_project_id" --X-Environment-
ID="$environment_id" --application_id="$application_id" --component_id="$component_id"`

    # 获取组件当前的版本号
    version=`getJsonValuesByAwk  "$component_detials" "version" "defaultValue"`
    version=`echo "$version"  | cut -d  '"' -f  2`
    echo "current version: $version"
    if [[ "$version" == "defaultValue" ]]; then
        echo "get component detials and get component version error"
        echo "$component_detials"
        exit 126
    fi
    upgradeVersion $version
    echo "upgrade version: $version"
}

# 软件包部署使用场景
function obs_software_upgrade() {
    echo "upload jar to obs"
    # 软件包上传到obs的地址，格式：obs://{桶名}{jar包在obs桶中的存放目录}
    bucket_address="obs://$bucket_name$bucket_dir"
    # 上传到obs的jar包链接，格式：https://{桶名}.obs.{region}.myhuaweicloud.com/{jar包在obs桶中的存放目
录}/
    obs_jar_url="https://$bucket_name.obs.$region.myhuaweicloud.com$bucket_dir$software_package"
    echo "software_package url: $obs_jar_url"
    # 将项目下构建生成的jar包上传到obs
    obs_result=`"$obsutil" cp ./target/*.jar "$bucket_address"`
    if [ $? -ne 0 ]; then
        echo "obsutil upload software package error"
        echo "$obs_result"
        exit 126
    fi
    # 打印上传结果
    echo "$obs_result"

    # 升级组件
    echo "upgrade component"
    action_result=`/usr/local/bin/hcloud CAE ExecuteAction --cli-region="$region" --project_id="$project_id" --
X-Enterprise-Project-ID="$enterprise_project_id" --X-Environment-ID="$environment_id" --
application_id="$application_id" --component_id="$component_id" --api_version='v1' --kind='Action' --
metadata.name='upgrade' --metadata.annotations.version="$version" --spec.source.type='softwarePackage'
--spec.source.sub_type='BinObs' --spec.source.url="$obs_jar_url"`
}

# 生成秒级时间格式字符串，用于上传SWR镜像时，在版本后添加时间格式后缀
function generateTimeFormatString() {
    ms=$(date +%s)
    time=$(date -d @$ms "+%Y%m%d%H%M%S")
    echo $time
}

# 镜像部署使用场景
function swr_image_upgrade() {
```

```
    # 上传到swr的镜像仓库路径，格式为：[镜像仓库地址]/[组织名称]/[镜像名称:版本名称]
    swr_image_url="swr.$region.myhuaweicloud.com/$swr_organization/$machine_image_name"
    # swr 镜像仓库地址
    swr_url="swr.$region.myhuaweicloud.com"

    # swr镜像版本拼接时间格式字符串
    time_format_string=`generateTimeFormatString`
    swr_image_url="$swr_image_url.$time_format_string"

    echo "upload image to swr"
    docker tag "$machine_image_name" "$swr_image_url"
    login_result=`docker login -u "$region"@"$AK" -p "$login_secret" "$swr_url"`
    # 打印登录swr镜像仓库结果
    #echo "$login_result"
    push_result=`docker push "$swr_image_url"`
    if [ $? -ne 0 ]; then
        echo "docker push error"
        echo "$push_result"
        exit 126
    fi
    # 打印推送镜像结果
    #echo "$push_result"
    logout_result=`docker logout "$swr_url"`
    # 打印退出登录swr镜像仓库的结果
    #echo "$logout_result"
    # 清除所有的历史记录，历史记录中可能会存在swr登录密钥信息，可以使用该命令清理所有的历史记录
    #history -c
    echo "upgrade component"
    # 升级 image镜像组件
    action_result=`/usr/local/bin/hcloud CAE ExecuteAction --cli-region="$region" --project_id="$project_id" --
X-Enterprise-Project-ID="$enterprise_project_id" --X-Environment-ID="$environment_id" --
application_id="$application_id" --component_id="$component_id" --api_version='v1' --kind='Action' --
metadata.name='upgrade' --metadata.annotations.version="$version" --spec.source.type='image' --
spec.source.url="$swr_image_url" `
}

### 方法简要说明:
### 1. 是查找一个字符串：带双引号的key。如果没找到，则直接返回defaultValue。
### 2. 查找最近的冒号，找到后认为值的部分开始了。
### 3. 如果有多个同名key，只打印第一个value
### 4. params: json, key, defaultValue
function getJsonValuesByAwk() {
    awk -v json="$1" -v key="$2" -v defaultValue="$3" 'BEGIN{
        foundKeyCount = 0
        pos = match(json, "\""key"\"[ \\t]*?:[ \\t]*");
        if (pos == 0) {if (foundKeyCount == 0) {print defaultValue;} exit 0;}
        ++foundKeyCount;
        start = 0; stop = 0; layer = 0;
        for (i = pos + length(key) + 1; i <= length(json); ++i) {
            lastChar = substr(json, i - 1, 1)
            currChar = substr(json, i, 1)
            if (start <= 0) {
                if (lastChar == ":") {
                    start = currChar == " " ? i + 1: i;
                    if (currChar == "{" || currChar == "[") {
                        layer = 1;
                    }
                }
            } else {
                if (currChar == "{" || currChar == "[") {
                    ++layer;
                }
                if (currChar == "}" || currChar == "]") {
                    --layer;
                }
                if ((currChar == "," || currChar == "}" || currChar == "]") && layer <= 0) {
                    stop = currChar == "," ? i : i + 1 + layer;
                    break;
                }
```

```
            }
        }
        if (start <= 0 || stop <= 0 || start > length(json) || stop > length(json) || start >= stop) {
            if (foundKeyCount == 0) {print defaultValue;} exit 0;
        } else {
            print substr(json, start, stop-start);
        }
    }'
}

### 方法简要说明:
###    从list返回的结果中，找到key=value时的tagKey值
###    查找json字符串'"key":"value"'，找到后，以这个位置为结尾
###    查找{最后一次出现的位置lastIndex，再以lastIndex为起始点，查找tagKey，获取tagKey的值
###
### params: json, key, value, tagKey, defaultValue
function getJsonValuesByAwkWithConditions() {
    awk -v json="$1" -v key="$2" -v value="$3" -v tagKey="$4" -v defaultValue="$5" 'BEGIN{
        foundKeyCount = 0
        pos = match(json, "\""key"\"[ \\t]*?:[ \\t]*\""value"\"");
        if (pos == 0) {print defaultValue; exit 0;}
        str1 = substr(json, 0, pos)
        lastIndex = 0
        while (match(str1, "{")) {
            lastIndex = lastIndex + RSTART
            str1 = substr(str1, RSTART + RLENGTH)
        }
        if (lastIndex == 0) {print defaultValue; exit 0;}
        newStr = substr(json, lastIndex)
        pos1 = match(newStr, "\""tagKey"\"[ \\t]*?:[ \\t]*");
        if (pos1 == 0) {print defaultValue; exit 0;}
        ++foundKeyCount;
        start = 0; stop = 0; layer = 0;
        for (i = pos1 + length(tagKey) + 1; i <= length(newStr); ++i) {
            lastChar = substr(newStr, i - 1, 1)
            currChar = substr(newStr, i, 1)
            if (start <= 0) {
                if (lastChar == ":") {
                    start = currChar == " " ? i + 1: i;
                    if (currChar == "{" || currChar == "[") {
                        layer = 1;
                    }
                }
            } else {
                if (currChar == "{" || currChar == "[") {
                    ++layer;
                }
                if (currChar == "}" || currChar == "]") {
                    --layer;
                }
                if ((currChar == "," || currChar == "}" || currChar == "]") && layer <= 0) {
                    stop = currChar == "," ? i : i + 1 + layer;
                    break;
                }
            }
        }
        if (start <= 0 || stop <= 0 || start > length(newStr) || stop > length(newStr) || start >= stop) {
            if (foundKeyCount == 0) {print defaultValue;} exit 0;
        } else {
            print substr(newStr, start, stop-start);
        }
    }'
}

# 获取enterprise_project_id environment_id application_id component_id
function initParpare() {
    # 获取environment_id
    listEnvsResult=`/usr/local/bin/hcloud CAE ListEnvironments --cli-region="$region" --
project_id="$project_id"`
```

```
    environment_id=`getJsonValuesByAwkWithConditions  "$listEnvsResult" "name" "$environment_name"
"id" "defaultValue"`
    environment_id=`echo "$environment_id"  | cut -d  '"' -f  2`
    echo "environment_id: $environment_id"
    if [[ "$environment_id" == "defaultValue" ]]; then
        echo "list environments and get environment_id error"
        echo "$listEnvsResult"
        exit 126
    fi

    # 获取enterprise_project_id
    enterprise_project_id=`getJsonValuesByAwk  "$listEnvsResult" "enterprise_project_id" "defaultValue"`
    enterprise_project_id=`echo "$enterprise_project_id"  | cut -d  '"' -f  2`
    echo "enterprise_project_id: $enterprise_project_id"
    if [[ "$enterprise_project_id" == "defaultValue" ]]; then
        echo "get enterprise_project_id error"
        echo "$listEnvsResult"
        exit 126
    fi

    # 获取application_id
    listAppsResult=`/usr/local/bin/hcloud CAE ListApplications  --cli-region="$region" --
project_id="$project_id" --X-Environment-ID="$environment_id"`
    application_id=`getJsonValuesByAwkWithConditions  "$listAppsResult" "name" "$application_name" "id"
"defaultValue"`
    application_id=`echo "$application_id"  | cut -d  '"' -f  2`
    echo "application_id: $application_id"
    if [[ "$application_id" == "defaultValue" ]]; then
        echo "list applications and get application_id error"
        echo "$listAppsResult"
        exit 126
    fi

    # 获取component_id
    listComponentsResult=`/usr/local/bin/hcloud CAE ListComponents  --cli-region="$region" --
project_id="$project_id" --X-Environment-ID="$environment_id" --application_id="$application_id"`
    component_id=`getJsonValuesByAwkWithConditions  "$listComponentsResult" "name"
"$component_name" "id" "defaultValue"`
    component_id=`echo "$component_id"  | cut -d  '"' -f  2`
    echo "component_id: $component_id"
    if [[ "$component_id" == "defaultValue" ]]; then
        echo "list components and get component_id error"
        echo "$listComponentsResult"
        exit 126
    fi
}

# 每隔15秒查询一次job状态，直到job完成
function waitDeployFinish() {
    sleep 10s
    id="$1"
    leni=${#id}
    id=${id:1:leni-2}
    echo "job_id= $id"
    job_status=""
    while [[ "$job_status" != "success" ]]; do
        job_status_result=`/usr/local/bin/hcloud CAE ShowJob --cli-region="$region" --project_id="$project_id"
--job_id="$id" --X-Environment-ID="$environment_id"`
        job_status=`getJsonValuesByAwk  "$job_status_result" "status" "defaultValue"`
        if [[ "$job_status" == "defaultValue" ]]; then
            echo "ShowJob failed"
            echo "$job_status_result"
            return
        fi
        job_status=`echo "$job_status"  | cut -d  '"' -f  2`
        if [[ "$job_status" != "running" && "$job_status" != "success" ]]; then
            echo 'upgrade failed'
            echo "$job_status_result"
            return
```

```
        fi
        sleep 15s
    done
    echo 'upgrade success'
}

function deploy() {
    if [[ "$deploy_type" == "software" ]]; then
        obs_software_upgrade
    elif [[ "$deploy_type" == "image" ]]; then
        swr_image_upgrade
    else
        return
    fi

    # 打印升级组件的结果
    echo "upgrade result: $action_result"
    # 获取结果中的job_id
    job_id=`getJsonValuesByAwk  "$action_result" "job_id" "defaultValue"`
    if [[ "$job_id" == "defaultValue" ]]; then
        echo "upgrade failed"
        exit 126
    fi
    # 等待升级完成
    waitDeployFinish "$job_id"
}

function main() {
    initParpare
    upgradeComponentVersion
    deploy
}
main
```

脚本参数说明

表 3-4 脚本参数说明

参数	是否必须	参数类型	描述
region	是	String	项目名称。
project_id	是	String	项目id。
environment_name	是	String	组件的环境名称。
application_name	是	String	组件的应用名称。
component_name	是	String	组件名称。
deploy_type	是	String	部署类型: software或image。 ● software表示软件包部署。 ● image表示镜像部署。

参数	是否必须	参数类型	描述
obsutil	否	String	当使用软件包部署如jar包部署时为必须参数。上传jar包到obs的工具安装的绝对路径。 示例：/root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil。
bucket_name	否	String	当使用软件包部署时为必须参数。存放软件包的obs桶名称。
bucket_dir	否	String	当使用软件包部署时为必须参数。 软件包在obs桶中的存放目录，默认是根目录，目录需要以/结尾，如果obs桶中没有这个目录，会自动创建出该目录。如根目录“/”，根目录下面的test目录“/test/”。
swr_organization	否	String	当使用镜像部署时为必须参数。上传到swr的组织名称。
AK	否	String	当使用镜像部署时为必须参数。创建永久AK、SK中的AK参数，用于登录swr镜像仓库。
login_secret	否	String	当使用镜像部署时为必须参数。swr的登录密钥，用于登录swr镜像仓库。通过创建永久AK、SK获取的AK/SK，执行如下命令，返回的结果就是登录密钥： `printf "$AK" \| openssl dgst -binary -sha256 -hmac "$SK" | od -An -vtx1 | sed 's/[ \n]//g' | sed 'N;s/\n//'`

参数值获取

- 获取region、project_id值：
  登录CAE控制台，在右上角个人账号上，单击“我的凭证”，查看所属区域的项目和项目ID，即为对应的region和project_id值。

  图 3-12 我的凭证

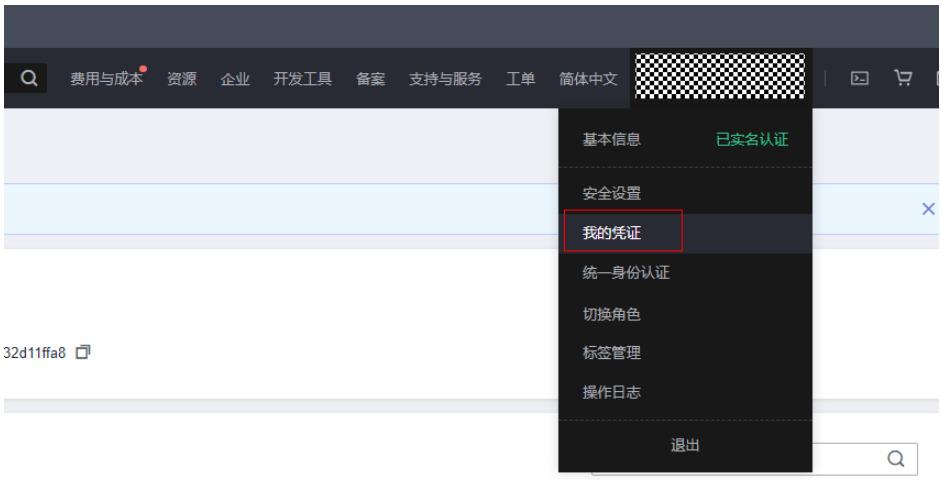


图 3-13 项目列表

- 获取组件所在的环境名称environment_name，应用名称application_name和组件名称component_name：

  登录CAE控制台，单击“组件列表”，找到目标组件，例如image组件，如图3-14所示。

图 3-14 查看组件信息

3.4 构建验证

3.4.1 手动构建验证

步骤1 在浏览器地址栏输入http://{安装Jekins的Linux虚拟机IP}:8080，登录Jenkins。

步骤2 左侧导航栏单击“我的视图”。

步骤3 选择对应的构建任务，单击构建任务名称进入详情界面，示例为“test-demo”。

步骤4 单击右侧 ，在左侧构建执行状态中会生成一个构建任务，单击任务编号，选择“Console Output”，查看构建输出日志。

步骤5 查看构建输出日志，构建出来的是jar包，部署成功。

图 3-15 构建输出日志

步骤6 登录CAE控制台。

步骤7 导航栏单击“组件列表”，查看对应组件的版本号已更新。

----结束

3.4.2 Gitlab 自动触发 Jenkins 构建

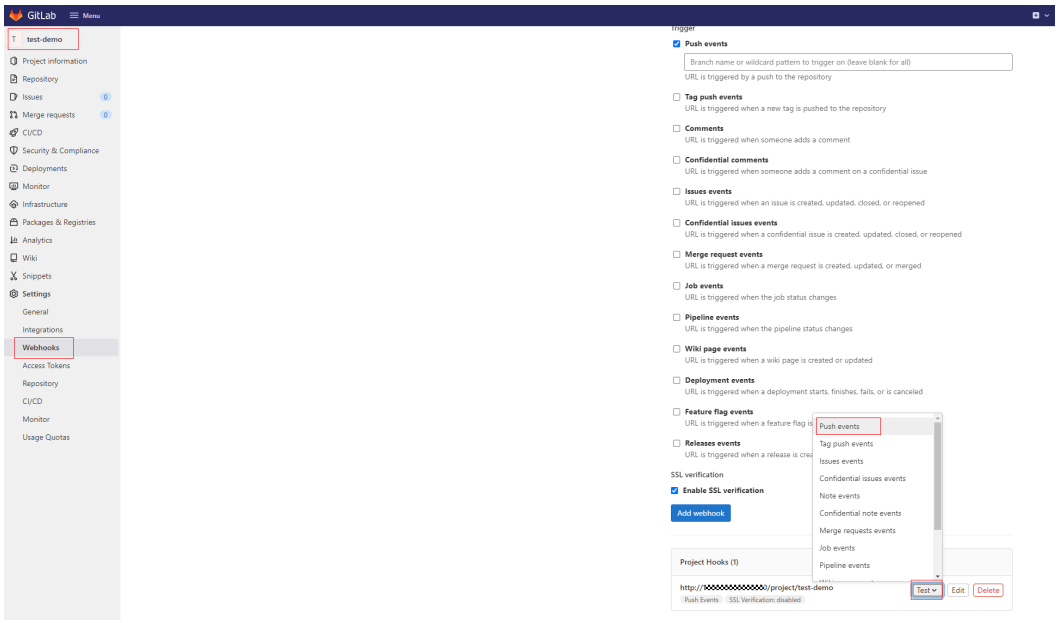
Gitlab触发Jenkins构建有以下两种方式：

- 通过配置好的Webhook来Push events，触发Jenkins构建任务。
- 修改构建配置指定分支的文件来Push events，触发Jenkins构建任务。

示例通过方式一来触发Jenkins构建。

步骤1 登录Gitlab，进入代码仓库，示例为“test-demo”，单击“Settings”，选择“Webhooks”，在右下角的“Test”下拉框，选择“Push events”，如图3-16所示。

图 3-16 Webhook 触发 Jenkins 构建



步骤2 登录Jenkins，左侧构建执行状态中可以看到已经触发的构建任务，单击构建任务编号，选择“Console Output”，查看构建输出日志。

步骤3 查看构建输出日志，构建出来的是jar包，部署成功，如图3-17。

图 3-17 查看构建输出日志

步骤4 登录CAE控制台。

步骤5 左侧导航栏单击“组件列表”，查看对应组件的版本号已更新。

----结束

3.5 附录

3.5.1 AK/SK 获取方法

步骤1 使用管理员账号登录CAE控制台。

步骤2 单击右上角的用户名，在下拉菜单选择“我的凭证”。

步骤3 在导航栏左侧选择“访问密钥”。

步骤4 单击“新增访问密钥”，通过身份认证后成功创建AK/SK。

步骤5 单击“立即下载”。

步骤6 下载成功后，在credentials文件中获取AK和SK信息：

- Access Key Id的值即为AK。
- Secret Access Key的值即为SK。

须知

- 每个用户仅允许保留2个有效的访问密钥。
- 为保证访问密钥的安全，访问密钥仅在初次生成时自动下载，后续不可再次通过控制台界面获取，请妥善保管访问密钥。

----结束

4 Jenkins 流水线支持多组件按照依赖顺序部署

4.1 概述

适用场景

使用jenkins来构建部署升级微服务组件，升级涉及多个微服务组件，每个组件有单独的流水线用于构建部署升级。但是组件之间存在依赖关系，必须等待所依赖的组件升级完成才能升级流水线，否则会有基本功能问题。

升级过程需要时刻关注所依赖的组件是否升级完成，才能进行下一个组件的升级任务。

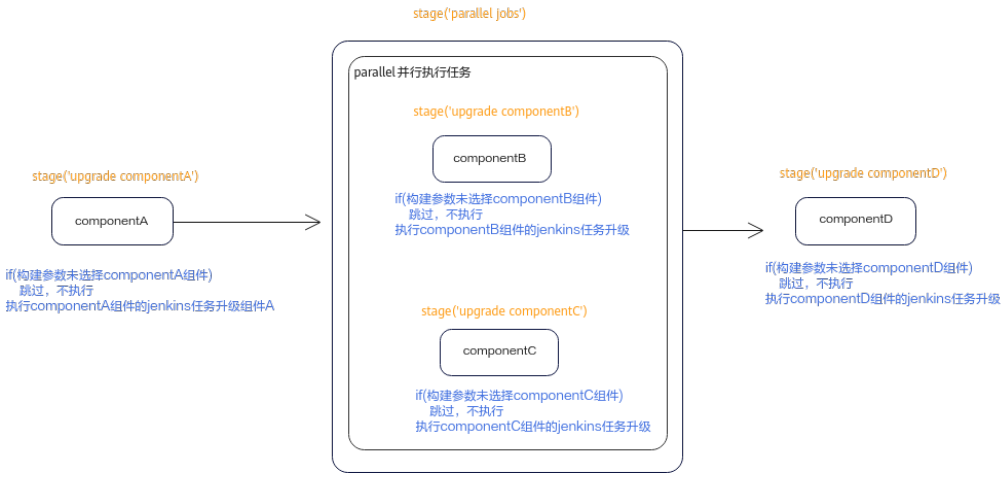
解决方案

新建一条jenkins流水线来编排多个组件，自动按照依赖顺序构建升级，并使用参数化构建的方式，同时支持单个或多个组件的升级。

下面以一个示例说明流水线执行流程。

有4个组件分别是componentA、componentB、componentC和componentD，其中componentD依赖componentB和componentC，componentB和componentC依赖componentA。

流水线执行流程：



4.2 前提条件

组件要求

每个待升级的组件已完成**配置流水线构建升级部署CAE**。

配置应用无损上线

升级多个组件时，需确保上一个组件已经就绪，升级才返回成功，再升级下一个组件。因此，您需要配置应用无损上线，具体操作请参考健康检查的**使用就绪探针保证升级时流量正常**章节进行配置。

若您的组件是SpringCloud应用，则请参考**Spring Cloud应用无损上线**。

安装 Jenkins 插件

步骤1 进入jenkins的系统管理页面，单击“插件管理”。

步骤2 在“插件管理”页面，单击“可选插件”，搜索“pipeline”和“extended choice parameter”两个插件进行安装，安装时选择不重启安装。

----结束

4.3 操作步骤

场景说明

当前示例包含4个组件，分别为componentA、componentB、componentC和componentD。

组件依赖关系为：componentD依赖componentB和componentC，componentB和componentC依赖componentA。

现在使用jenkins流水线来编排部署这4个组件，支持单个或多个组件升级而不用考虑依赖关系单个组件依次升级。

配置 jenkins 流水线

步骤1 进入jenkins的系统管理页面，单击“新建任务”。

步骤2 输入任务名称，选择任务类型为“流水线”。

步骤3 在“General”页面中，添加“扩展选项参数”。

步骤4 配置需要编排的组件。

表 4-1 配置构建参数

参数名称	描述	示例
Name	设置参数的名称，构建时选择的组件会存放在该变量名。	Model_Name
Description	设置构建参数的描述。	选择需要升级的组件
Parameter Type	选择基本参数类型。	Check Boxes
Number of Visible Items	设置流水线编排的组件数量。	4
Delimiter	设置组件之间的分隔符，设置为逗号（,）。	,
Value	设置流水线编排的组件名称，用逗号（,）分隔。	componentA,componentB,componentC,componentD

步骤5 选择“流水线”，在“流水线”页面进行脚本配置，脚本内容请参考流水线脚本。

步骤6 脚本配置完成后，单击“保存”。

----结束

流水线脚本

流水线脚本支持componentA、componentB、componentC和componentD，4个组件中的一个或多个同时升级，并且按照依赖顺序先升级componentA，再并行升级componentB和componentC，最后升级componentD。如果构建时没有勾选到的组件会跳过，不执行升级。

```
pipeline {
    agent any
    environment{
        //构建时选择的组件以‘,’隔开的格式存在变量Model_Name里
        //判断变量Model_Name里的可选组件是否存在，如果存在，把组件名赋值给对应变量，如果不存在，把字符串"error"赋值给对应变量。
        //用于判断后续组件是否需要构建
        _componentA="${sh(script:' echo $Model_Name| grep -w -o "componentA" || echo "error" ', returnStdout: true).trim()}"
        _componentB="${sh(script:' echo $Model_Name| grep -w -o "componentB" || echo "error" ', returnStdout: true).trim()}"
        _componentC="${sh(script:' echo $Model_Name| grep -w -o "componentC" || echo "error" ', returnStdout: true).trim()}"
```

```
      _componentD="${sh(script:' echo $Model_Name| grep -w -o "componentD" || echo "error" ',
returnStdout: true).trim()}"
   }

   stages {
      stage('Build componentA') {
         //when条件判断当环境变量"_componentA" = "componentA"时，表示构建时选择了该组件，执行此
stage，否则跳过此stage
         when { environment name: '_componentA', value: 'componentA' }
         steps {
            sh ''' echo "start to build componentA" '''
            script{
               //build(job: 'componentA') 表示执行componentA任务，这个任务是jenkins中创建的任务名称，用
于componentA组件升级的任务
               def componentBuild=build(job: 'componentA')
               //打印执行任务的结果
               println componentBuild.getResult()
            }
         }
      }
      stage('Build parallel jobs') {
         //failFast true 表示parallel并行的stage中只要有一个失败，其他并行任务也会中止
         //不加failFast true，parallel中失败的stage不会影响其他并行的stage执行，直到所有的并行任务执行完
成
         //failFast true //按需添加

         //parallel 表示里面的stage是并行执行的，只要其中有一个stage执行失败，则"Build parallel jobs"这个
stage是失败的，它后续的stage不会执行
         parallel {
            stage('Build componentB') {
               when { environment name: '_componentB', value: 'componentB' }
               steps {
                  sh ''' echo "start to build componentB" '''
                  script{
                     def componentBuild=build(job: 'componentB')
                     println componentBuild.getResult()
                   }
                }
            }
            stage('Build componentC') {
               when { environment name: '_componentC', value: 'componentC' }
               steps {
                  sh ''' echo "start to build componentC" '''
                  script{
                     def componentBuild=build(job: 'componentC')
                     println componentBuild.getResult()
                  }
               }
            }
         }
      }
      stage('Build componentD') {
         when { environment name: '_componentD', value: 'componentD' }
         steps {
            sh ''' echo "start to build componentD" '''
            script{
               def componentBuild=build(job: 'componentD')
               println componentBuild.getResult()
            }
         }
      }
   }
}
```

脚本说明

每个组件都需要在pipeline中的envionment获取组件参数，用于判断参数化构建时是否选中，执行下面脚本：


```
_{component_name}="${sh(script:' echo $Model_Name| grep -w -o "{component_name}" || echo "error" ', returnStdout: true).trim()}"
```

表 4-2 参数说明

参数名称	描述
Model_Name	**配置构建参数**中的Name。
{component_name}	**配置构建参数**中的Choose Source for Value中的value里面填写的组件名称之一。

每个组件都需要配置一个stage，用于组件的升级任务，如果是串行执行的，则按照先后顺序放置stage的位置，如果是并行执行的任务就放置在stage('Build parallel jobs')中的parallel中。

stage脚本：

```
stage('Build {component_name}') {

    when { environment name: '_{component_name}', value: '{component_name}' }
    steps {
      sh ''' echo "start to build {component_name}" '''
      script{
        def componentBuild=build(job: '{component_jenkins_task}')
        //打印执行任务的结果
        println componentBuild.getResult()
      }
    }
}
```

表 4-3 参数说明

参数	描述
{component_name}	**配置构建参数**中的Choose Source for Value中的value里面填写的组件名称之一。
{component_jenkins_task}	jenkins中创建的用于组件升级的任务名称。

使用流水线升级多个组件

步骤1 选择流水线，单击“参数化构建”。

步骤2 选择要升级的组件，可以选择一个或多个同时升级，本示例选择全部组件同时升级，单击“开始构建”。

步骤3 可在构建历史中查看流水线任务执行结果。

步骤4 流水线日志如下所示，可以看出先升级componentA，再并行升级componentB和componentC，最后升级componentD。

```
Started by user admin
Running in Durability level: MAX_SURVIVABILITY
```



```
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /root/.jenkins/workspace/pipeline-arrange
[Pipeline] {
[Pipeline] sh
+ echo componentA,componentB,componentC,componentD
+ grep -w -o componentA
[Pipeline] sh
+ echo componentA,componentB,componentC,componentD
+ grep -w -o componentD
[Pipeline] sh
+ echo componentA,componentB,componentC,componentD
+ grep -w -o componentB
[Pipeline] sh
+ echo componentA,componentB,componentC,componentD
+ grep -w -o componentC
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Build componentA)
[Pipeline] sh
+ echo 'start to build componentA'
start to build componentA
[Pipeline] script
[Pipeline] {
[Pipeline] build
Scheduling project: componentA
Starting building: componentA #13
[Pipeline] echo
SUCCESS
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build parallel jobs)
[Pipeline] parallel
[Pipeline] { (Branch: Build componentB)
[Pipeline] { (Branch: Build componentC)
[Pipeline] stage
[Pipeline] { (Build componentB)
[Pipeline] stage
[Pipeline] { (Build componentC)
[Pipeline] sh
[Pipeline] sh
+ echo 'start to build componentB'
start to build componentB
[Pipeline] script
+ echo 'start to build componentC'
start to build componentC
[Pipeline] {
[Pipeline] script
[Pipeline] {
[Pipeline] build
Scheduling project: componentB
[Pipeline] build
Scheduling project: componentC
Starting building: componentB #12
Starting building: componentC #12
[Pipeline] echo
SUCCESS
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] echo
SUCCESS
[Pipeline] }
```

```
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // parallel
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build componentD)
[Pipeline] sh
+ echo 'start to build componentD'
start to build componentD
[Pipeline] script
[Pipeline] {
[Pipeline] build
Scheduling project: componentD
Starting building: componentD #10
[Pipeline] echo
SUCCESS
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

步骤5 **登录CAE控制台**，在组件列表查看组件状态，“最近一次变更状态/时间”列显示状态为“升级成功”，表示组件升级成功。

----结束

5 SpringCloud 应用部署到 CAE 自动对接 Nacos 引擎

5.1 概述

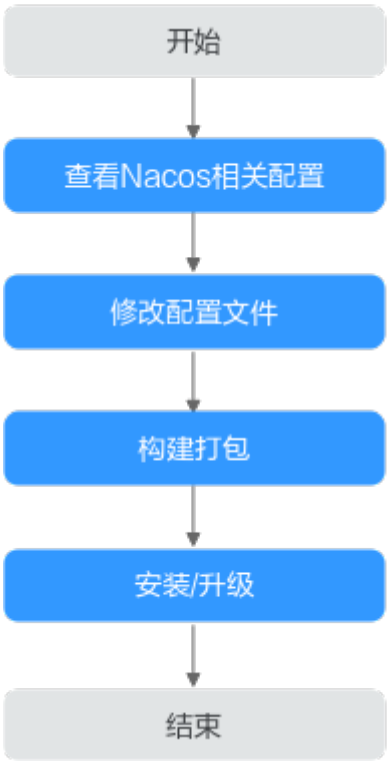
应用场景

用户可以将业务代码打包部署到CAE，只需在部署的组件配置中添加Nacos引擎配置，即可用自动化对接Nacos引擎，无需手动修改业务代码配置文件，避免人为误操作。

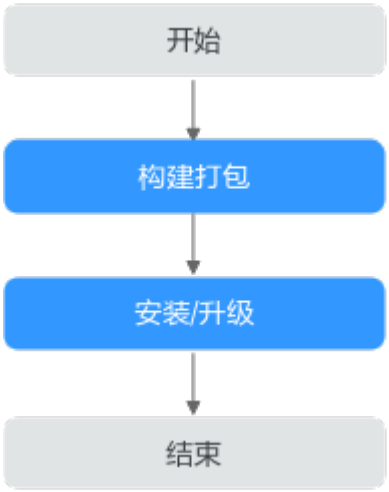
本章节通过CAE源码部署能力自动化部署一个provider服务和一个consumer服务，帮助您体验自动接入Nacos引擎。

流程优化对比

当前对接Nacos流程：



该方案对接Nacos流程：



原理说明

在Spring Cloud应用中，系统环境变量的优先级高于配置文件。因此，如果在系统环境变量中配置了参数，那么系统会优先使用环境变量中的参数。为了实现自动对接Nacos引擎，CAE会自动在组件中添加SPRING_CLOUD_NACOS_DISCOVERY_SERVERADDR和SPRING_CLOUD_NACOS_CONFIG_SERVERADDR环境变量。这样，即使在配置文件中没有配置这些参数，系统也能够正确地连接到Nacos引擎。

须知

如果用户在外部配置了比环境变量优先级更高的nacos注册发现参数，可能会导致nacos对接异常。例如，如果在启动命令中添加了--spring.cloud.nacos.discovery.server-addr=nacos.com，这会覆盖环境变量中的相应配置，从而导致组件对接nacos异常无法正常工作。因此，在配置nacos参数时，需要注意优先级的问题，以确保nacos能够正确地对接。

5.2 前提条件

- 已**创建环境**和**应用**。
- 已创建Nacos引擎，具体操作请参考**创建Nacos引擎**。
  Nacos引擎所在虚拟私有云和子网同CAE环境的VPC一致。
- 登录GitHub，将Nacos源码fork到您的仓库下。源码地址：**https://github.com/nacos-group/nacos-examples**。

5.3 操作步骤

源码仓库授权

步骤1 登录CAE控制台。

步骤2 选择“系统设置”。

步骤3 在“系统设置”页面，单击“源码仓库授权”模块的“编辑”，进入“已授权源码仓库”页面。

步骤4 单击“新建授权”，进入“新建授权”页面。

1. 单击“GitHub”。
2. 授权名称输入：nacos-github。
3. 授权方式选择“OAuth”。
4. 单击“使用OAuth授权”。

步骤5 在“服务声明”弹框中，勾选“我已知晓本服务的源码构建功能收集上述信息，并同意授权对其的收集、使用行为。”。单击“确认”完成授权。

步骤6 授权完成后，进入GitHub登录页面。

步骤7 输入用户GitHub的用户名或邮箱和密码，单击“Sign in”，等待OAuth认证完成，返回CAE页面。

步骤8 在CAE“系统设置”页面提示框单击“确认”，完成源码仓库授权。

----结束

创建并部署 nacos-provider 组件

步骤1 在导航栏中选择“组件列表”。

步骤2 在页面上方，下拉选择**前提条件**中已创建的应用和环境，单击“新增组件”。

步骤3 参考**表5-1**配置组件信息。

表 5-1 nacos-provider 组件基本信息

参数	说明
组件名称	新建组件的名称。本实践输入名称为“nacos-provider”。
版本号	组件的版本号。 本实践版本号为1.0.0。
实例规格	选择实例规格，例如：0.5core、1GiB。
实例数量	输入实例数为1。
代码源	选择“源码仓库 > GitHub”，配置授权信息。 ● 授权信息：nacos-github。 ● 用户名/组织：用户授权使用的用户名/邮箱。 ● 仓库名称：nacos-example。 ● 分支：master。
语言/运行时	选择“Java17”。
自定义构建	1. 选择“使用自定义命令”。 2. 小黑框内输入： `cd ./nacos-spring-cloud-example/nacos-spring-cloud-discovery-example/nacos-spring-cloud-provider-example && mvn clean package`
Dockerfile地址	输入“./”。

步骤4 单击“配置组件”。

步骤5 在“组件配置”页面，单击“微服务引擎CSE”模块的“配置”，进入“微服务引擎CSE配置”页面。

步骤6 单击“注册配置中心”，选择**前提条件**中您已创建的Nacos引擎。

步骤7 单击“保存”，完成对该组件微服务引擎CSE配置。

步骤8 在“组件配置”页面，单击“环境变量”模块的“编辑”，对nacos-provider组件进行配置。

步骤9 单击“新增环境变量”，输入变量名称及变量。

- 变量名称：JAVA_TOOL_OPTIONS。
- 变量/变量引用：--add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED。

步骤10 单击“确定”，完成对该组件环境变量配置。

步骤11 在“组件配置”页面，单击“配置并部署组件”。

步骤12 确认配置信息无误后，单击“确定”，等待组件部署完成。

----结束

创建并部署 nacos-consumer 组件

步骤1 在导航栏中选择“组件列表”，返回组件列表页面。

步骤2 在页面上方，下拉选择**前提条件**中已创建的应用和环境，单击“新增组件”。

步骤3 参考**表5-2**配置组件信息。

表 5-2 nacos-consumer 组件基本信息

参数	说明
组件名称	新建组件的名称。本实践输入名称为“nacos-consumer”。
版本号	组件的版本号。 本实践版本号为1.0.0。
实例规格	选择实例规格，例如：0.5core、1GiB。
实例数量	输入实例数为1。
代码源	选择“源码仓库 > GitHub”，配置授权信息。 ● 授权信息：nacos-github。 ● 用户名/组织：用户授权使用的用户名/邮箱。 ● 仓库名称：nacos-example。 ● 分支：master。
语言/运行时	选择“Java17”。
自定义构建	1. 选择“使用自定义命令”。 2. 小黑框内输入： `cd ./nacos-spring-cloud-example/nacos-spring-cloud-discovery-example/nacos-spring-cloud-consumer-example && mvn clean package`
Dockerfile地址	输入“./”。

步骤4 单击“配置组件”。

步骤5 在“组件配置”页面，单击“微服务引擎CSE”模块的“配置”，进入“微服务引擎CSE配置”页面。

步骤6 单击“注册配置中心”，选择**前提条件**中您已创建的Nacos引擎。

说明

nacos-consumer组件和nacos-provider组件需绑定同一个Nacos引擎。

步骤7 单击“保存”，完成对该组件微服务引擎CSE配置。

步骤8 在“组件配置”页面，单击“环境变量”模块的“编辑”，对nacos-consumer组件进行配置。

步骤9 单击“新增环境变量”，输入变量名称及变量。

- 变量名称：JAVA_TOOL_OPTIONS。

- 变量/变量引用：--add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED。

步骤10 单击“确定”，完成对该组件环境变量配置。

步骤11 在“组件配置”页面，单击“访问方式”模块的“编辑”，进入“访问方式配置”页面。

步骤12 在“负载均衡配置”页签下，单击“添加负载均衡配置”并设置参数。

- 协议：选择“TCP”。
- 监听端口：输入“8080”。
- 访问端口：自定义访问端口。本实践设置为“14688”。

步骤13 单击“确定”，完成对该组件访问方式配置。

步骤14 在“组件配置”页面，单击“配置并部署组件”，等待组件部署完成。

步骤15 确认配置信息无误后，单击“确定”，等待组件部署完成。

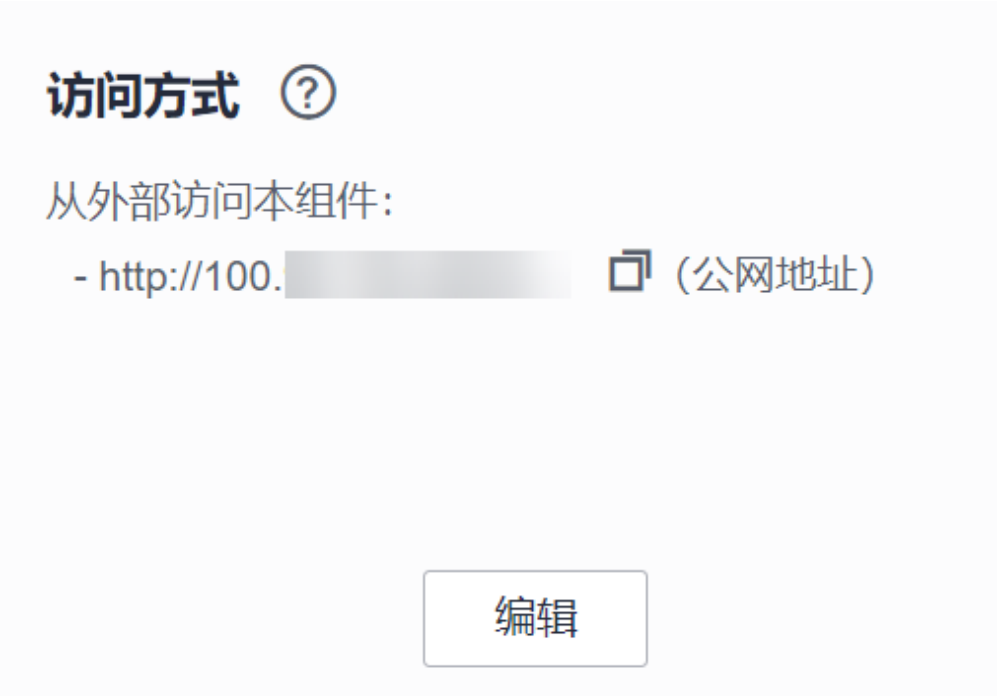
----结束

调用微服务

步骤1 在左侧导航栏中选择“组件列表”，进入“组件列表”页面。

步骤2 选择**nacos-consumer**组件，在“访问地址”列单击，获取自动生成的外网访问地址。

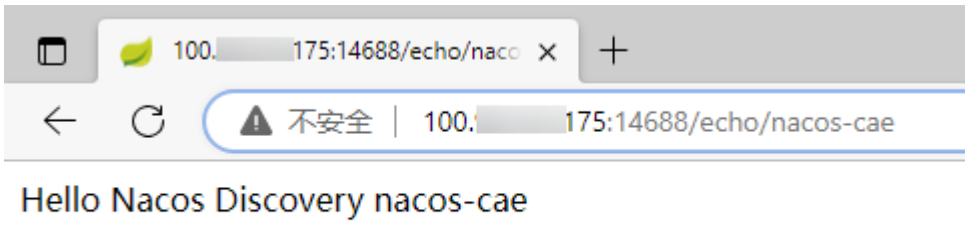
图 5-1 获取外网访问地址



步骤3 在浏览器输入访问地址：http://外网访问ip地址:访问端口/echo/{string}，外网访问ip地址、访问端口为**步骤2**中获取的，{string}为自定义字符串。例如：http://100.**.**.175:14688/echo/nacos-cae。

如果出现如下图所示欢迎页面，表示Nacos调用成功。

图 5-2 调用微服务引擎



----结束

6 Spring Cloud 应用无损上线

概述

在组件运维过程中，不可避免要进行升级、重启、扩容等操作，在这些操作中，无损上线是常见的要求，本文介绍如何配置Spring Cloud无损上线。

前提条件

- 已**创建环境**。
- 已**创建应用**。
- 已**创建并部署组件**，确保您的组件为Spring Cloud。

操作步骤

步骤1 Spring Cloud启用spring-boot-starter-actuator。

1. 在组件对应的源码的pom.xml中添加spring-boot-starter-actuator依赖（建议使用2.3.0及以上版本）。

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
   <version>{您使用的spring boot版本}</version>
</dependency>
```

2. 修改application.properties，添加如下配置：

表 6-1 application.properties 配置

spring boot版本	配置
2.3.0/2.3.1	management.health.probes.enabled=true
>= 2.3.2	management.endpoint.health.probes.enabled=true management.health.livenessState.enabled=true management.health.readinessState.enabled=true

3. 更新代码。
   - 若您的组件使用源码部署，请将修改的源码更新至源码仓库。

- 若您的组件使用软件包部署，请将新代码打包为软件包，并将新的软件包上传至软件包仓库。

4. 使用新的源码或软件包，**升级组件**。

步骤2 配置组件健康检查。

1. 登录CAE控制台。选择“组件配置”。
2. 在“组件配置”页面上方的下拉框中选择待操作的组件。
3. 参考**配置健康检查**，配置就绪探针，具体参数如下：

表 6-2 就绪探针参数

配置内容	值
检查方式	HTTP请求检查。
端口	您的组件的实际监听端口。
路径	- spring boot 2.3及以上版本：/actuator/health/readiness。 - spring boot 2.3以下版本：/actuator/health。
协议	您的组件的实际协议。
检测周期	10
延迟时间	0
超时时间	1
成功阈值	1
最大失败次数	3

4. 升级组件，具体操作请参考**升级组件**。

步骤3 验证配置。

1. 选择“组件列表”，若操作的组件状态为“运行中”，则进入下一步，否则配置失败。
2. 选择“组件事件”，在页面上方的下拉框中选择操作的组件。若存在“组件实例健康”的事件，则配置成功，否则配置失败。

----结束

7 健康检查

7.1 概述

健康检查用于检测您的应用实例是否正常工作，是用来保障业务正常运行的一种机制。CAE提供三种健康检查机制：存活探针、就绪探针和启动探针。

- 存活探针用于检测应用程序是否存活。如果检测实例异常，k8s将会删除当前运行的实例并重新检测，直到检测返回实例正常。

  须知

  - 单独使用存活探针时，如出现网络波动或程序启动过慢的情况，会导致实例持续被重启，且实例一直处于未就绪的状态。

    有如下解决方案：

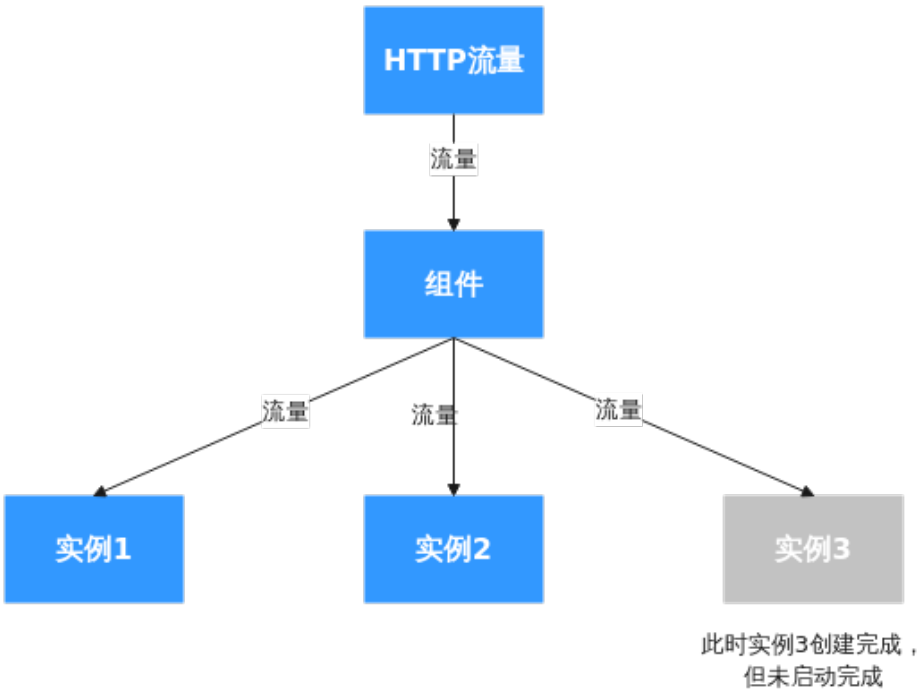
    - 与启动探针配合使用，具体操作请参见**启动探针与存活探针配合工作**。
    - 使用时将“最大失败次数”调大，增加容错率，并增大“延迟时间”，保证程序在启动后再接受存活探针的检测。
  - 检查成功：对于在健康检查中设置的请求返回状态码200。
  - 检查失败：对于在健康检查中设置的请求返回状态码非200，且连续失败次数达到设置的“最大失败次数”。

- 就绪探针用于检测应用是否完成启动，并准备好开始接受请求。如果检查到实例已经健康，则进行流量的切换。

  例如：以组件的实例从2个伸缩到3个为例，说明设置就绪探针前后对比。

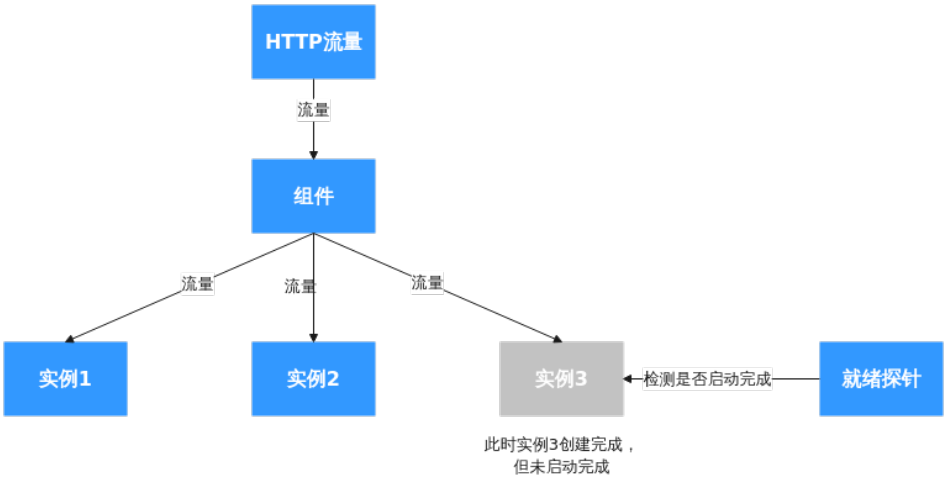
  a. 未设置就绪探针时，实例已创建完成，但由于程序原因，并未准备好开始接受流量，此时如**图7-1**所示，部分流量仍然会进入实例3中。

图 7-1 未配置就绪探针



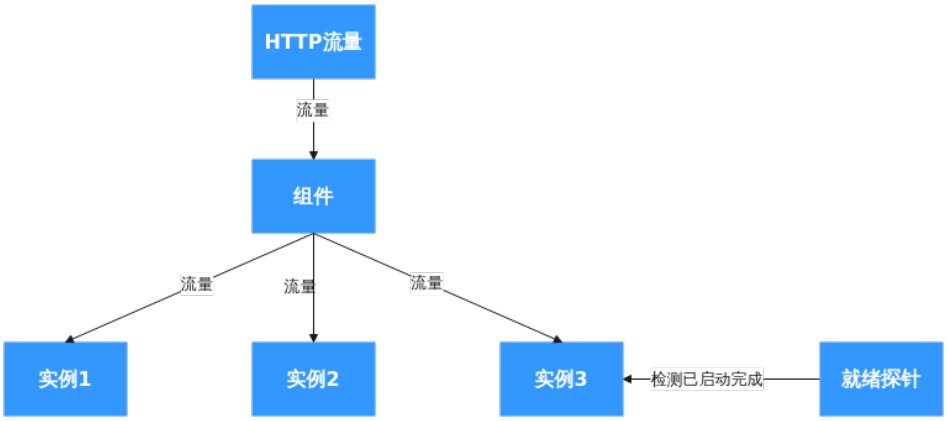
b. 设置就绪探针后，就绪探针会进行检测，发现实例3并未启动完成或就绪，则使实例3不接收流量，保证所有流量都会流入健康的实例1和实例2中。

图 7-2 就绪探针检测健康前，流量不放通



c. 当就绪探针检查到实例3为健康时，则放通流量，停止本次检测。

图 7-3 就绪探针健康后，放通流量



- 启动探针在运行时会禁用其他探测（就绪探针以及存活探针），如果启动探针检查失败，则实例会被重启。

  启动探针建议搭配存活探针一起使用。

7.2 启动探针与存活探针配合工作

前提条件

- 已**创建环境**。
- 已**创建应用**。
- 已**创建并部署组件**，此示例中所用组件为demo-frontend。

操作步骤

步骤1 登录CAE控制台。选择“组件配置”。

步骤2 在“组件配置”页面上方的下拉框中选择待操作的组件。

图 7-4 选择待操作的组件

步骤3 如图7-5和图7-6所示分别配置启动探针与存活探针，并生效配置，具体操作步骤请参考**配置健康检查**。

图 7-5 配置存活探针

图 7-6 配置启动探针

启动探针会在实例创建后10秒钟进行检查，每5秒检测一次，且连续失败5次后进行容器重启的操作。在启动探针检测到实例健康后，存活探针将进行启动检测，可以避免程序启动过慢，导致实例一直重启。

须知

请保证程序启动时间在“延迟时间”+ “检测周期”* “最大失败次数”秒内能启动，否则启动探针会一直重启实例。如果不确定程序启动时间，建议调大“最大失败次数”以及“延迟时间”。

如图7-6所示，程序需要在10 + 5 * 5 = 35秒内启动。

步骤4 单击“组件事件”，在“组件事件”页面中可以看到启动探针探测一次失败后，程序成功启动，切换到了存活探针，并检测实例健康。

图 7-7 查看事件信息

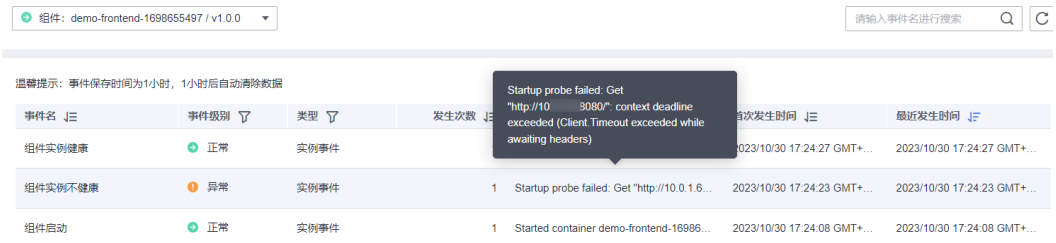


图 7-8 启动探针探测情况

----结束

7.3 使用就绪探针保证升级时流量正常

前提条件

- 已**创建环境**。
- 已**创建应用**。
- 已**创建并部署组件**，此示例中所用组件为demo-frontend。
- 已配置**访问方式**并生效配置。

操作步骤

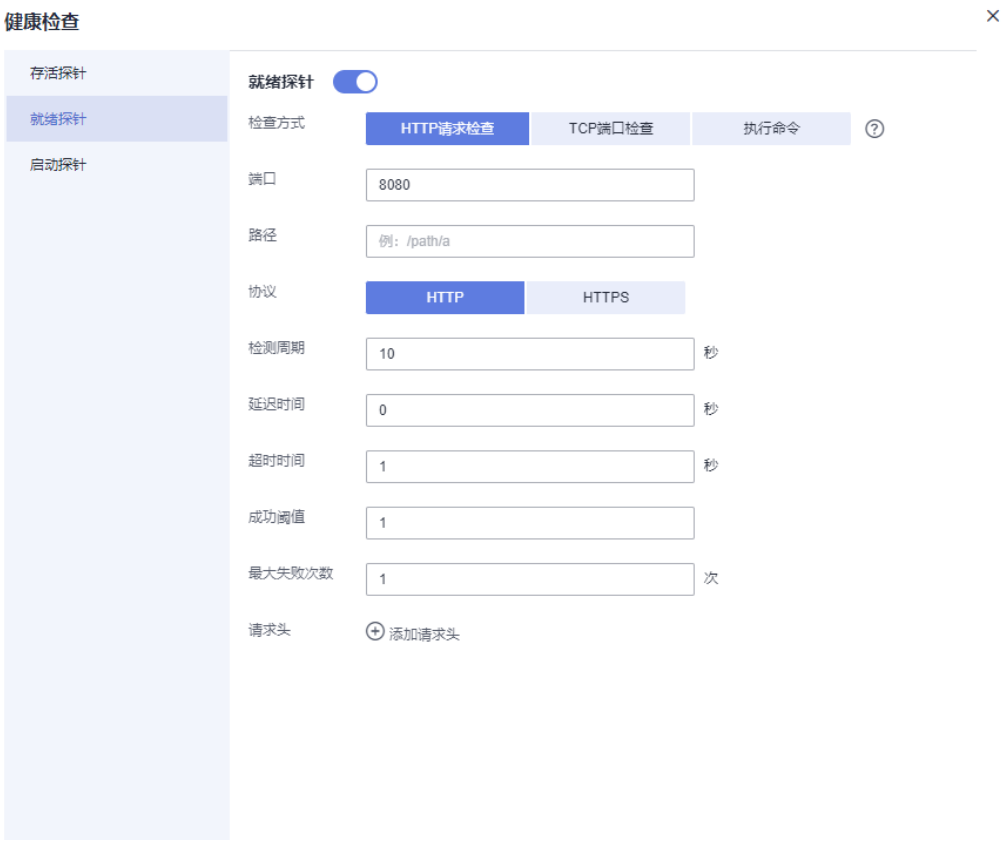
步骤1 登录CAE控制台。选择“组件配置”。

步骤2 在“组件配置”页面上方的下拉框中选择待操作的组件。

图 7-9 选择待操作的组件

步骤3 如图7-10所示配置就绪探针，并生效配置，具体操作步骤请参考**配置健康检查**。

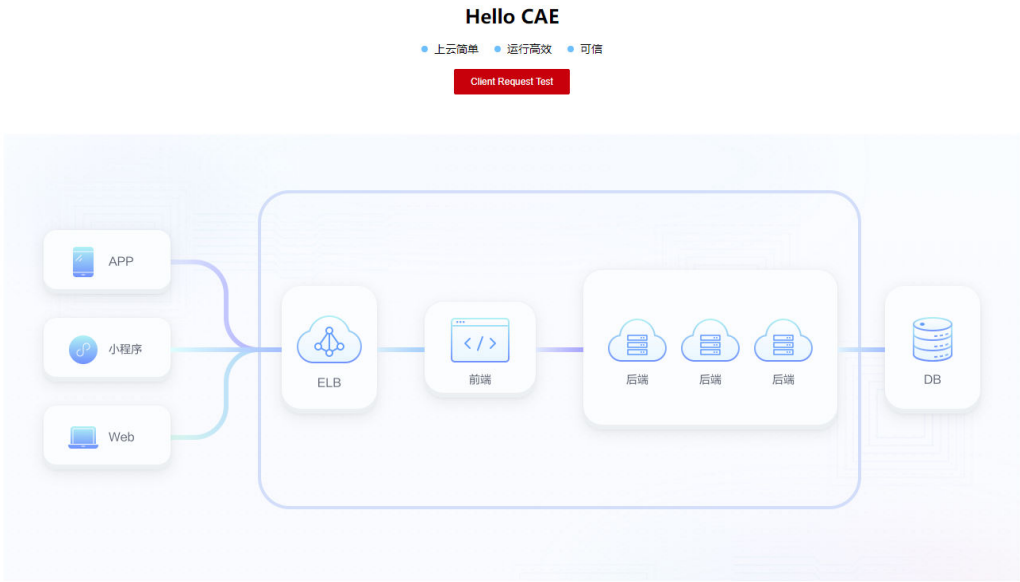
图 7-10 配置就绪探针



步骤4 对组件进行升级操作，具体操作请参考**升级组件**。

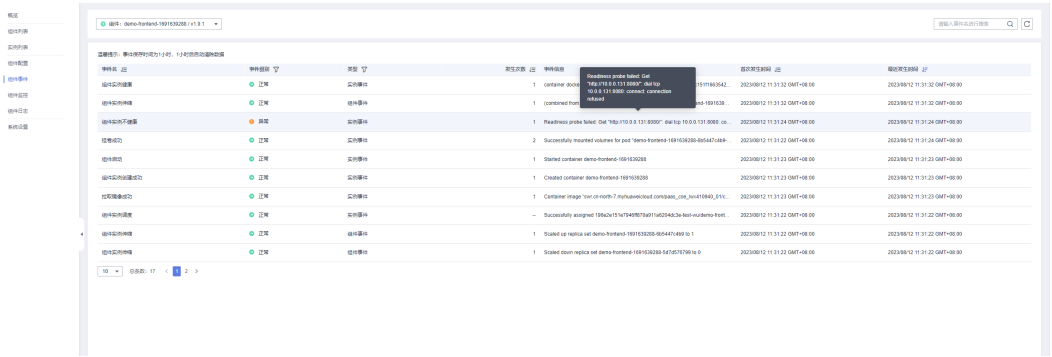
步骤5 升级时，在左侧导航栏中选择“组件列表”，进入“组件列表”页面，单击对应组件“访问地址”列的ip地址，查看应用页面。如图7-11所示服务并无中断。

图 7-11 访问应用页面



步骤6 在“组件事件”页面，查看组件事件，可以看到实例存在不健康的请求，此时检测到新实例还未准备好进行流量切换，则继续使用旧实例提供服务。

图 7-12 查看组件事件



----结束

8 生命周期管理

8.1 概述

生命周期管理是用于在特定阶段执行调用的方法。CAE提供两种生命周期管理：启动后处理、停止前处理。

- 启动后处理：组件实例启动后立即触发启动后事件，但是不确保对应的handler是否能在容器的EntryPoint之前执行。

  说明

  只有启动后处理函数执行完毕，组件实例的状态才会变成Running。因此当命令设置为死循环时，CAE组件状态将无法正常变成运行中。

- 停止前处理：组件实例停止前，会发送停止前事件。

  说明

  组件实例结束前会立即发送停止前事件，除非实例宽限期限超时，组件实例会一直阻塞等待停止前函数执行完毕。

8.2 利用启动后处理写文件

前提条件

- 已**创建环境**。
- 已**创建应用**。
- 已**创建并部署组件**，此示例中所用组件为nginx组件。

操作步骤

步骤1 登录CAE控制台。选择“组件配置”。

步骤2 在“组件配置”页面上方的下拉框中选择待操作的组件。

图 8-1 选择组件

步骤3 如图8-2所示配置启动后处理，具体操作请参考**配置生命周期**。

分别输入以下命令：

```
/bin/bash
-c
echo 'Hello, postStart' > /lifecycle.txt
```

图 8-2 配置生命周期

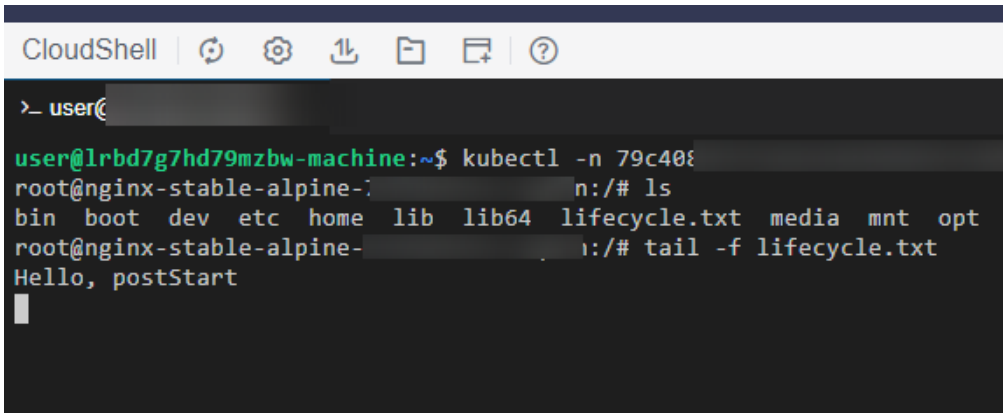
步骤4 单击“确定”，保存配置。

步骤5 单击页面上方“生效配置”。在右侧弹框中确认配置信息，并单击“确定”，使配置生效。

步骤6 使用CloudShell登录容器，查看文件内容是否生效。

1. 在左侧导航栏中选择“实例列表”。
2. 在“实例列表”页面上方的下拉框中选择环境、应用及待操作的组件。
3. 选择运行中的实例，在“操作”列单击“远程登录”。进入容器内部，查看文件内容是否生效。
4. 查看文件内容，确认是否已写入成功。
```
tail -f lifecycle.txt
```

图 8-3 登录容器查看文件内容



----结束

8.3 利用停止前处理优雅关闭 Nginx

如果容器碰到问题被系统关闭，停止前处理可以帮助您的主程序在关闭前执行必要的清理任务。

前提条件

- 已**创建环境**。
- 已**创建应用**。
- 已**创建并部署组件**，此示例中所用组件为nginx组件。

操作步骤

步骤1 登录CAE控制台。选择“组件配置”。

步骤2 在“组件配置”页面上方的下拉框中选择待操作的组件。

图 8-4 选择组件

步骤3 如图8-5所示配置停止前处理，具体操作请参考**配置生命周期**。

分别输入以下命令：

```
/bin/bash
-c
nginx -s quit;while killall -0 nginx;do sleep 1;done
```

图 8-5 配置生命周期

步骤4 单击“确定”，保存配置。

步骤5 单击页面上方“生效配置”。在右侧弹框中确认配置信息，并单击“确定”，使配置生效。

----结束

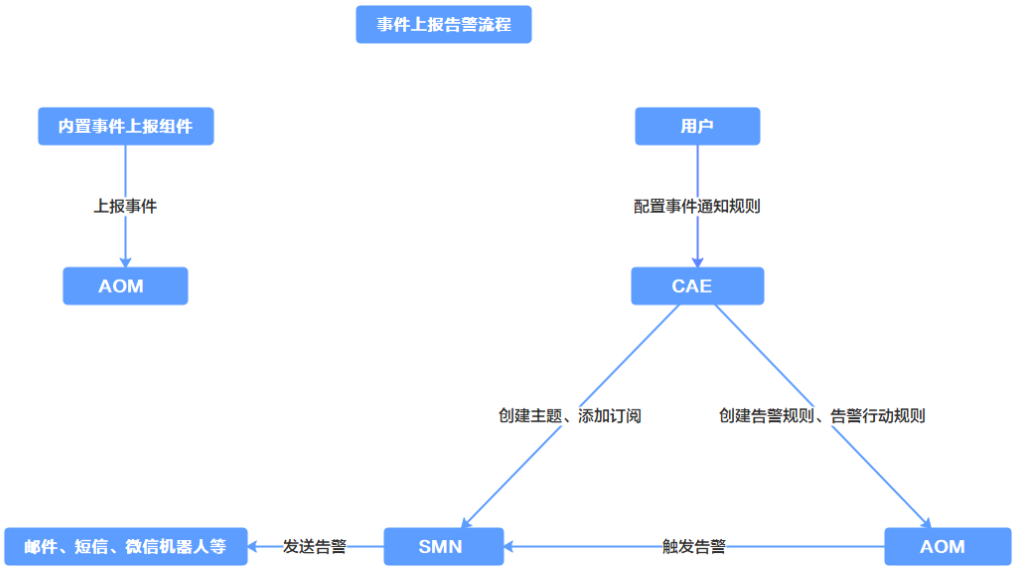
9 发送事件告警到企业微信

概述

CAE支持在实例调度成功/失败，健康检查成功/失败，拉取镜像成功/失败，卷挂载成功/失败，容器启动成功/失败时发送通知。通过设置事件通知规则，可以帮助您及时了解组件运行时的状态，快速定位问题。配置事件通知规则依赖AOM服务和SMN服务，组件实例事件上报AOM服务，您可以通过“AOM > 告警管理 > 告警列表 > 事件”查看上报的事件，SMN服务是实际的告警消息的发送方。

CAE默认上报组件实例事件到AOM，用户在配置事件通知规则之后，由CAE在SMN服务创建主题、添加订阅者，在AOM服务创建告警规则、告警行动规则，完成整个事件上报告警流程的配置。

图 9-1 事件上报告警流程



前提条件

SMN服务支持添加企业微信群消息、钉钉群消息、飞书群消息的订阅，目前，这些功能处于公测期间，您需先申请开通SMN服务的公测资格。

- 已开通企业微信，并创建了群机器人。
- 已在华为云**提交工单**申请SMN服务的公测资格，并已开通公测。

操作步骤

步骤1 **登录CAE控制台**，选择“系统设置”。

步骤2 单击“事件通知规则”模块中的“编辑”，进入“事件通知规则配置”页面。

步骤3 单击“创建事件通知规则”，参考**表9-1**配置基本信息。

表 9-1 配置基本信息说明

配置项	配置项说明
事件通知规则名称	输入事件通知规则名称。例如：container-Initiate。 由英文字母、数字、中划线和下划线组成，并以英文字母开头和结尾，长度为1-64个字符。
触发事件	在下拉框中选择事件通知的触发事件。例如：容器启动成功。
生效组件范围	选择环境内所有组件。
触发策略	触发方式选择“立即触发”。

步骤4 选择通知方式为“企业微信机器人”。

填写企业微信机器人终端地址：请输入以https://qyapi.weixin.qq.com/cgi-bin/webhook/send开头的webhook地址。

说明

获取企业微信订阅终端参考**企业微信机器人如何获取订阅终端**。

图 9-2 配置事件通知

步骤5 单击“确定”，完成事件通知规则创建。

步骤6 在组件列表中选择待操作组件，单击操作列“更多 > 重启”。

等待组件重启成功后，您可以登录AOM控制台查看事件列表或在企业微信群接收到告警消息。

- 登录**AOM控制台**查看CAE上报的事件列表，
  a. 选择“告警管理 > 告警列表 > 事件”。
  b. 事件级别勾选“全选”。
  c. 选择筛选条件为“事件源：CAE”，单击 按钮。
- 在企业微信群接收到告警信息。

----结束

10 对接软件开发生产线 CodeArts 流水线自动升级到 CAE

10.1 概述

CAE目前提供了Codearts商业插件，可对接Codearts流水线自动升级组件到CAE，此章节将根据不同场景为您介绍“CAE升级插件”的使用方法。

目前支持从Codearts构建为镜像，再部署到CAE，也可以通过Codearts构建为软件包上传至Codearts的软件发布库或OBS中，再部署至CAE。

前提条件

- 已开通软件开发生产线CodeArts服务，已创建项目并托管代码。
- 已**创建组件**。

10.2 流水线构建软件包上传到 Codearts 软件发布库后升级 CAE 组件

创建编译构建任务

步骤1 进入**CodeArts控制台**，单击右上角“立即使用”。

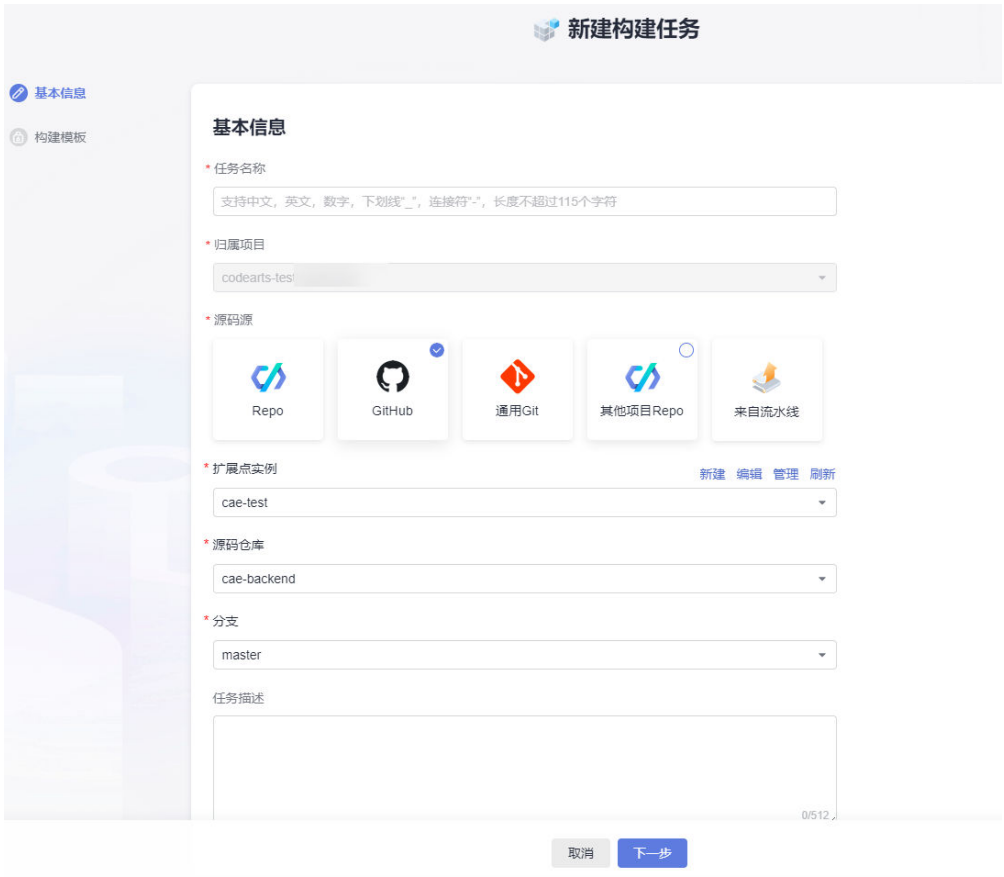
步骤2 在菜单栏选择“服务 > 编译构建”。

图 10-1 选择服务

步骤3 在编译构建页面单击“新建任务”，在基本信息中填入“任务名称”，并选择“归属项目”、“源码源”、“扩展点实例”、“源码仓库”和“分支”，单击“下一步”。

图 10-2 新建任务

图 10-3 配置基本信息



步骤4 选择构建使用的模板（此示例使用的为java代码，故此步选择Maven模板）后，单击“下一步”。

图 10-4 选择构建模板

步骤5 在“参数设置”页面，“自定义参数”中单击“新建参数”，并开启运行时设置。

- 名称：输入release_version。
- 值：取值使用时间戳变量：${TIMESTAMP}。

图 10-5 添加自定义参数

步骤6 进入“构建步骤”页面后，选择“上传软件包到软件发布库”，“发布版本号”选择使用变量“${release_version}”作为软件包存储的目录。

配置固定的包名，存储在软件发布库中，单击“新建并执行”。

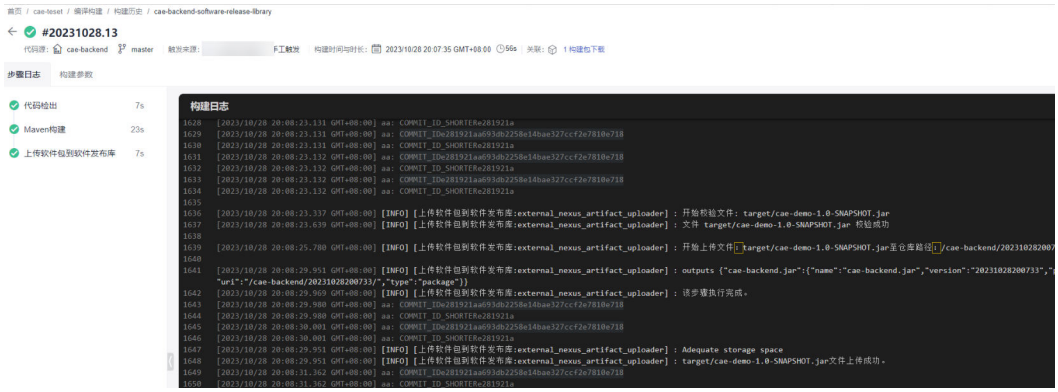
图 10-6 上传软件包

步骤7 确认运行时参数后，单击“确定”。

图 10-7 配置运行时参数

步骤8 查看构建日志，构建成功并上传软件包到软件发布库中。

图 10-8 查看构建日志



步骤9 在“制品仓库 > 软件发布库”中查看上传的软件包，目录结构为：项目名称-包名-构建时间戳-软件包名称。

图 10-9 查看软件包

----结束

创建流水线升级 CAE 组件

创建Codearts流水线：

步骤1 返回Codearts控制台首页。

步骤2 在菜单栏选择“服务 > 流水线”，在“流水线”页面单击“新建流水线”。

图 10-10 选择流水线

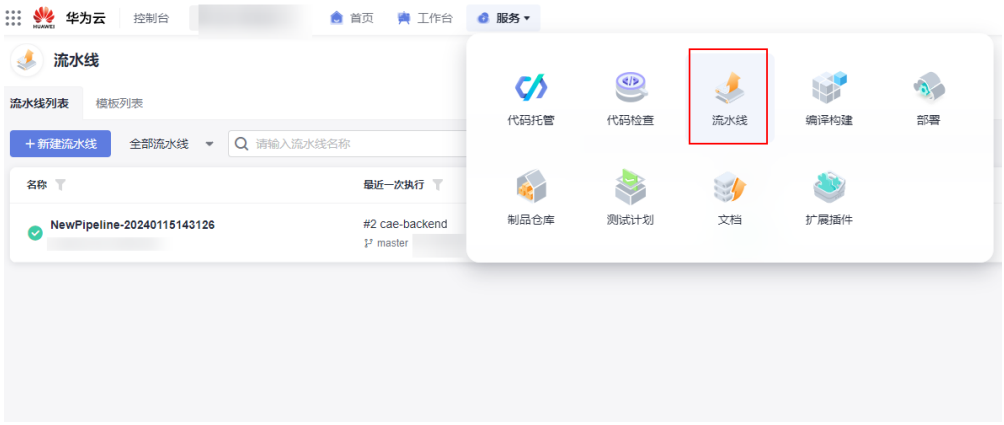


图 10-11 新建流水线

步骤3 在“新建流水线”页面，输入“名称”并选择“流水线源”、“代码仓”和“默认分支”，并单击“下一步”。

图 10-12 配置流水线基本信息

步骤4 模板选择“全部 > 空模板”，单击“确定”。

图 10-13 选择模板

步骤5 进入“任务编排”页面后，单击“新建阶段”。单击 将“阶段_1”改名为“构建”，“阶段_2”改名为“部署”。

图 10-14 新建阶段

步骤6 新建构建任务。

- 在“构建”阶段，单击“新建任务”。
  a. 进入“新建任务”页面，单击“构建”，选择“Build构建”任务，并单击“添加”。

图 10-15 新建任务

图 10-16 添加“Build 构建”任务

b. 选择前一步创建的“编译构建任务”和“仓库”，将“release_version”设置为流水线参数，单击“确定”。

- 在“部署”阶段，单击“新建任务”。

a. 在搜索框中输入“CAE”，并单击 。选择“CAE发布”插件，单击“添加”。

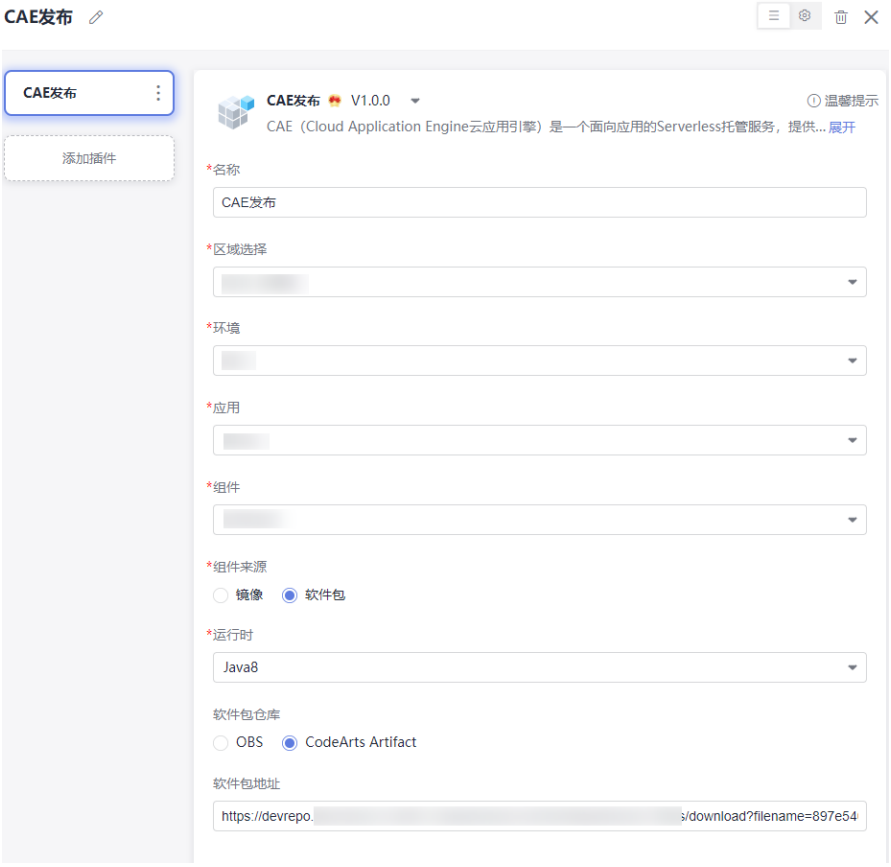
图 10-17 选择插件

b. 配置插件参数后，单击“确定”。

- 区域选择：待部署的区域，与当前CodeArts在同一个Region。
- 环境：选择组件所属环境。
- 应用：选择组件所属应用。
- 组件：选择待升级的组件（需要选择在CAE中使用软件包部署的组件）。
- 组件来源：选择“软件包”。
- 运行时：选择软件包对应的运行时。
- 软件包仓库：选择“CodeArts Artifact”。

- 软件包地址：取值见**参数说明**。

图 10-18 配置插件参数



步骤7 单击“保存并执行”，进入执行配置页面，确认执行配置信息无误后，单击“执行”。

图 10-19 执行任务

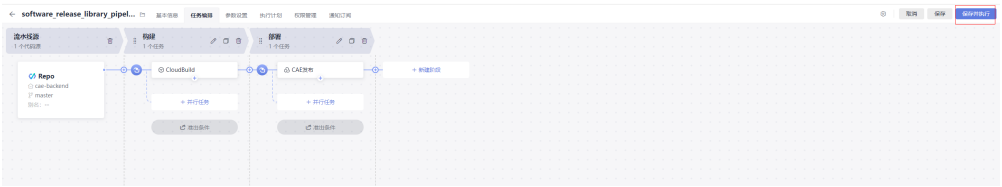


图 10-20 确认执行配置信息

步骤8 查看流水线执行结果。

图 10-21 查看流水线执行结果

步骤9 登录CAE控制台，在组件列表查看组件状态，“最近一次变更状态/时间”列显示状态为“升级成功”，则表示Codearts流水线执行升级成功。

----结束

参数说明

表 10-1 参数说明

参数名称	是否必须	参数类型	描述
release_version	是	String	发布版本号，取值使用时间戳变量：${TIMESTAMP}。
软件包地址	是	String	编译构建出来的软件包上传到软件发布库中的地址。 例如：复制**图10-9**中下载地址，下载地址为https://devrepo.devcloud.cn-east-3.huawei.com/DevRepoServer/v1/files/download?filename=39d25a3cc6ee48678533020abcfbf941/cae-backend/20231028200733/cae-backend.jar，将软件包的目录发布版本号20231028200733改为${release_version}，则软件包地址的值为https://devrepo.devcloud.cn-east-3.huawei.com/DevRepoServer/v1/files/download?filename=39d25a3cc6ee48678533020abcfbf941/cae-backend/${release_version}/cae-backend.jar。

10.3 流水线构建软件包上传到 obs 桶后升级 CAE 组件

创建编译构建任务

创建代码仓库的构建任务：

步骤1 进入**CodeArts控制台**，单击右上角“立即使用”。

步骤2 在菜单栏选择“服务 > 编译构建”。

图 10-22 选择服务

步骤3 在“编译构建”页面单击“新建任务”，在基本信息中填入“任务名称”，并选择“归属项目”、“源码源”、“源码仓库”和“分支”，单击“下一步”。

图 10-23 配置构建基本信息

步骤4 选择构建使用的模板（此示例使用的为java代码，故此步选择Maven模板）后，单击“下一步”。

图 10-24 选择构建模板

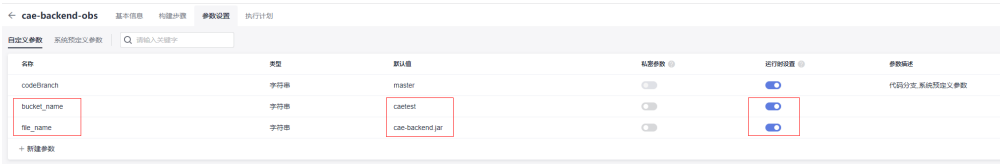
步骤5 进入“构建步骤”页面后，选择“添加步骤”，单击“文件上传”，选择“上传文件到OBS”，单击“添加”。

图 10-25 文件上传

步骤6 进入“参数设置”页面，参考图10-26配置构建参数和默认值，并开启运行时设置。bucket_name：上传到obs的桶名称，如caetest。

- file_name：上传到obs桶中的软件包名称，如cae-backend.jar。

图 10-26 配置构建参数



步骤7 进入“构建步骤”页面，删除“上传软件包到软件发布库”步骤。

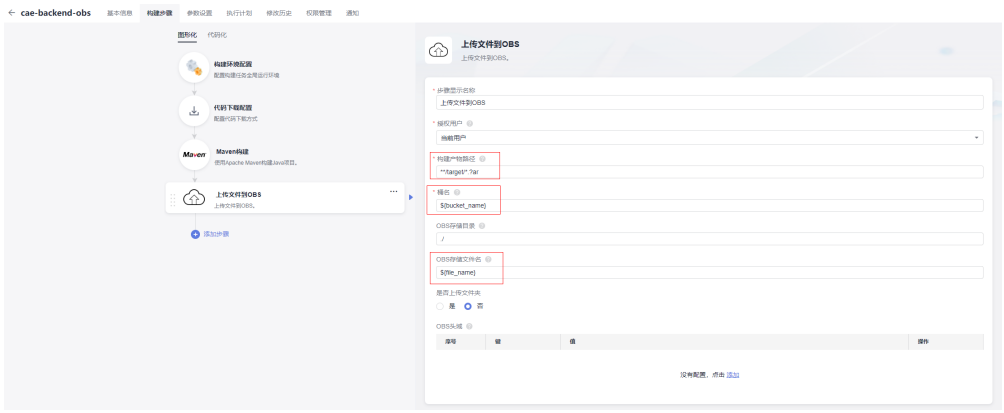
图 10-27 删除“上传软件包到软件发布库”步骤

步骤8 在弹框中单击“确定”，完成删除。

步骤9 配置“上传文件到OBS”参数。

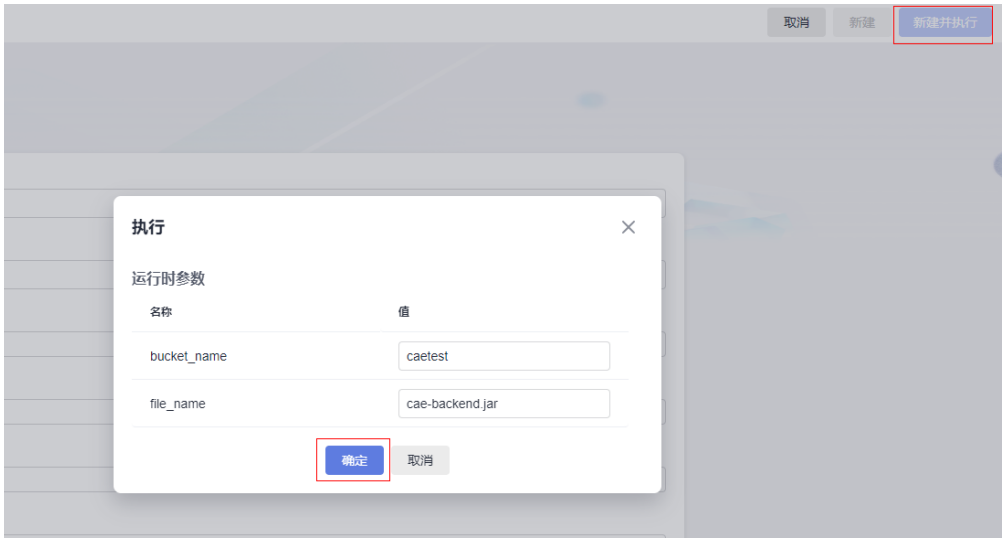
- “配置构建产物路径”为“**/target/*.?ar”。
- “桶名”使用参数“${bucket_name}”。
- “OBS存储目录”使用默认配置。
- “OBS存储文件名”使用参数“${file_name}”。

图 10-28 配置“上传文件到 OBS”参数



步骤10 单击右上角“新建并执行”，确认运行时参数无误后，单击“确定”。

图 10-29 执行任务



步骤11 查看构建日志，等待构建成功并成功上传软件包到OBS中。

步骤12 登录OBS控制台，在OBS桶中查看软件包已上传成功。

图 10-30 查看 OBS 桶

----结束

创建流水线使用 OBS 软件包升级 CAE 组件

创建Codearts流水线：

步骤1 返回Codearts控制台首页。

步骤2 在菜单栏选择“服务 > 流水线”，在“流水线”页面单击“新建流水线”。

图 10-31 选择服务

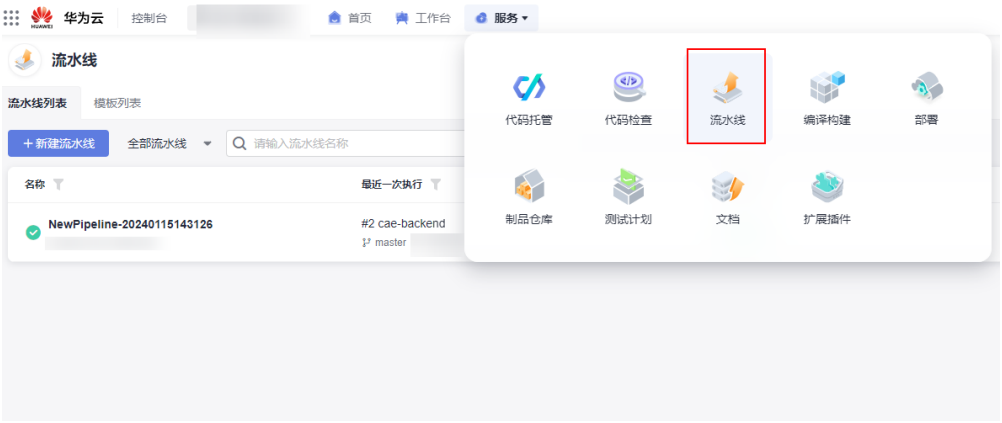
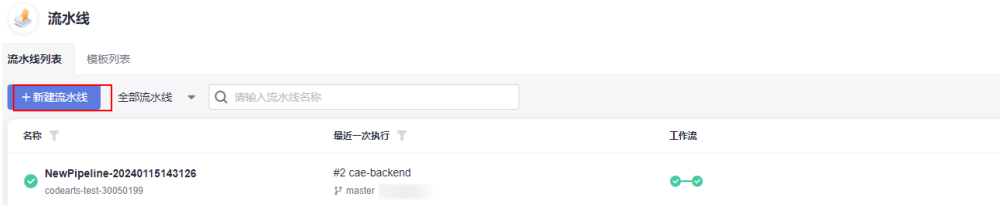


图 10-32 新建流水线



步骤3 在“新建流水线”界面，输入“名称”并选择“流水线源”、“代码仓”和“默认分支”，单击“下一步”。

图 10-33 配置流水线基本信息

步骤4 模板选择“全部 > 空模板”，单击“确定”。

图 10-34 选择模板

步骤5 进入“任务编排”页面后，单击“新建阶段”。单击 将“阶段_1”改名为“构建”，“阶段_2”改名为“部署”。

图 10-35 新建阶段

步骤6 新建构建任务。

- 在“构建”阶段，单击“新建任务”。
  a. 进入“新建任务”页面，单击“构建”，选择“Build构建”任务，并单击“添加”。

图 10-36 新建任务

图 10-37 添加“Build 构建”任务

b. 选择前一步创建的“编译构建任务”和“仓库”，并将创建编译构建任务**步骤6**配置的“bucket_name”和“file_name”设置为流水线参数，单击“确定”。

- 在“部署”阶段，单击“新建任务”。

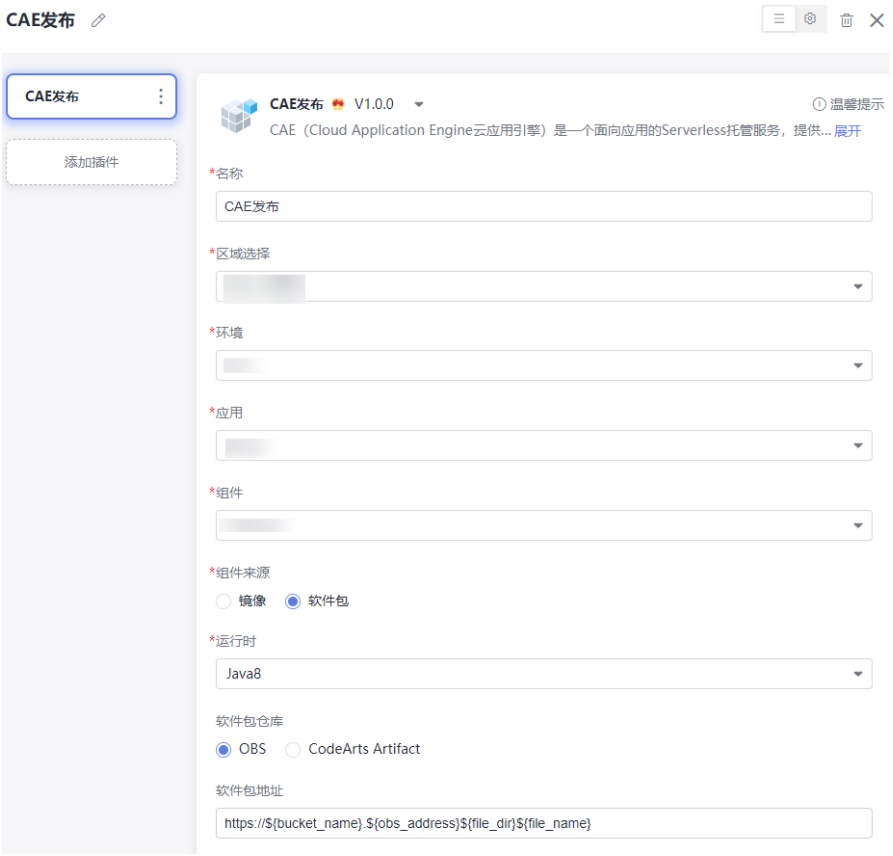
a. 在搜索框中输入“CAE”，并单击 。选择“CAE发布”插件，单击“添加”。

图 10-38 添加部署任务

b. 配置插件参数后，单击“确定”。

- 区域选择：待部署的区域，与当前CodeArts在同一个Region。
- 环境：选择组件所属环境。
- 应用：选择组件所属应用。
- 组件：选择待升级的组件（需要选择在CAE中使用软件包部署的组件）。
- 组件来源：选择“软件包”。
- 运行时：选择软件包对应的运行时。
- 软件包仓库：选择“OBS”。
- 软件包地址：取值见**参数说明**。

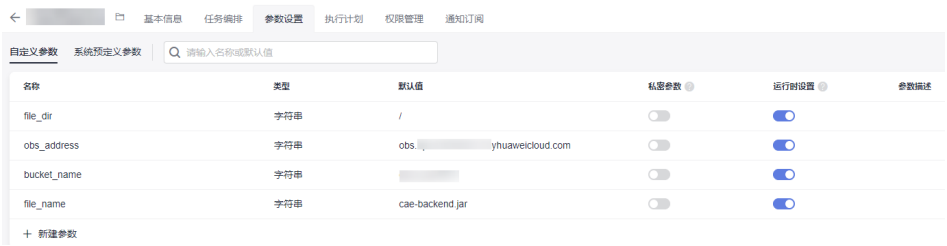
图 10-39 配置插件参数



c. 进入“参数设置”页面，参考**参数说明**添加自定义参数。

- bucket_name：obs桶名称，如caetest。
- file_name：上传到obs桶中的软件包名称，如cae-backend.jar。
- file_dir：软件包上传到obs桶中的存放目录，如根目录是“/”，根目录下面的test目录“/test/”。
- obs_address：为OBS桶的地址，名称格式为：obs.区域项目名称.myhuaweicloud.com。

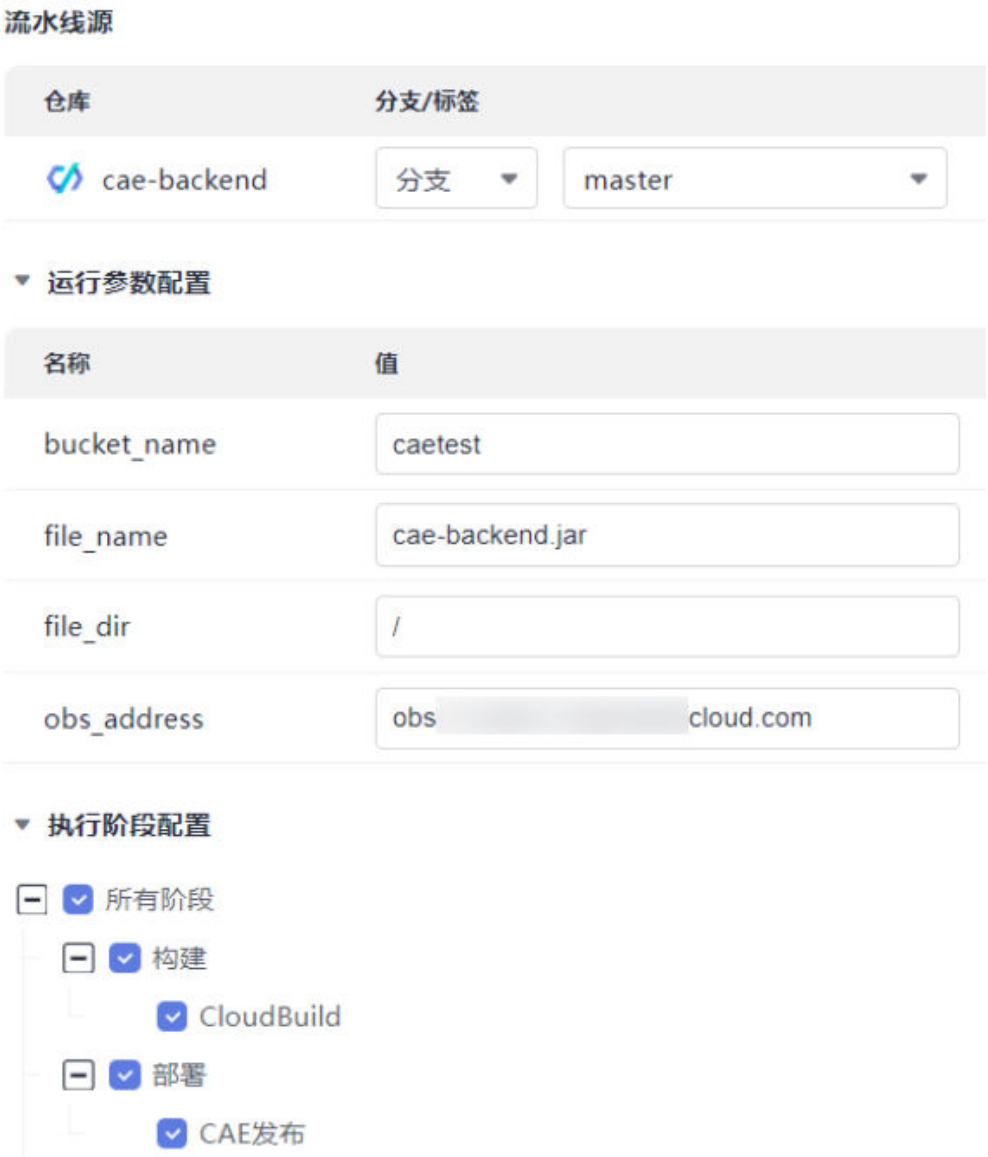
图 10-40 配置自定义参数



步骤7 单击“保存并执行”，进入执行配置页面，确认执行配置信息无误后，单击“执行”。

图 10-41 执行任务

图 10-42 确认流水线配置信息



步骤8 自动跳转至流水线详情页面，查看流水线执行结果。

图 10-43 查看流水线执行结果

步骤9 登录CAE控制台，在组件列表查看组件状态，“最近一次变更状态/时间”列显示状态为“升级成功”，则表示Codearts流水线执行升级成功。

图 10-44 查看组件状态

----结束

参数说明

表 10-2 参数说明

参数名称	是否必须	参数类型	描述
bucket_name	是	String	obs桶名称，CAE发布插件中选择的区域和obs桶所在的区域保持一致。
file_name	是	String	上传到obs桶中的软件包名称。
file_dir	是	String	软件包上传到obs桶中的存放目录，目录需要以/结尾，如果obs桶中没有这个目录，会自动创建出该目录。如根目录是“/”，根目录下面的test目录“/test/”。
obs_address	是	String	值的格式为：obs.区域项目名称.myhuaweicloud.com，区域项目名称就是region，参考参数值获取。
软件包地址	是	String	上传到obs的软件包地址，值填写：https://${bucket_name}.${obs_address}${file_dir}${file_name}。

10.4 流水线构建镜像上传到 swr 镜像仓库后升级 CAE 组件

创建编译构建任务

创建代码仓库的构建任务

步骤1 进入CodeArts控制台，单击右上角“立即使用”。

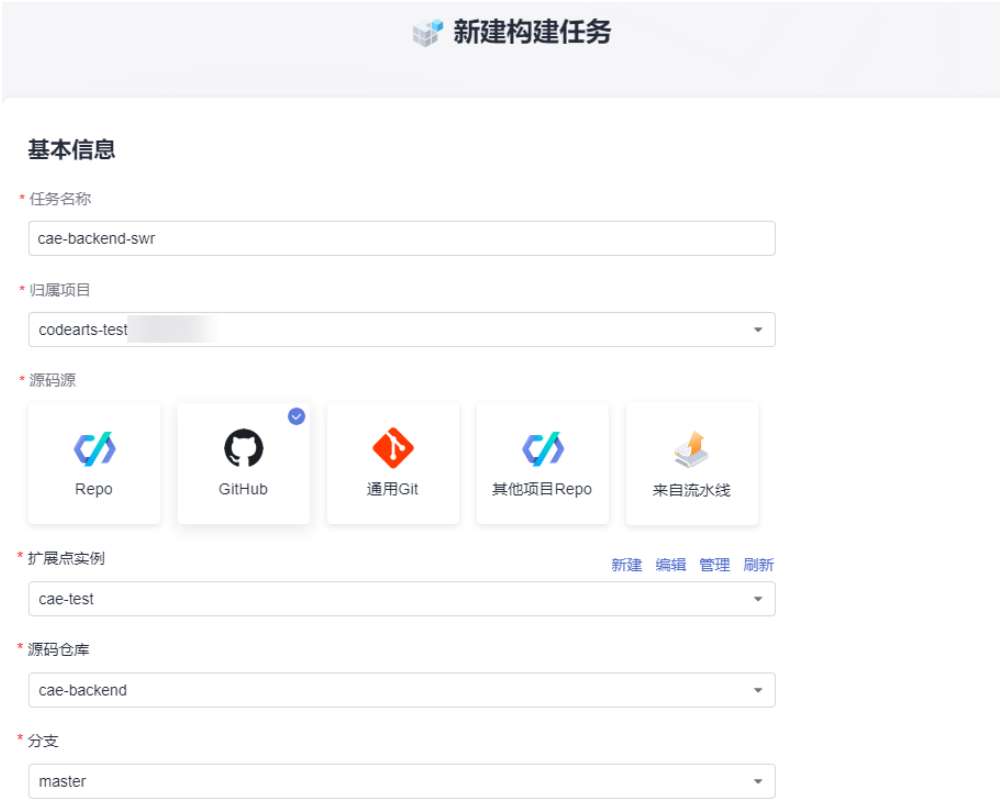
步骤2 在菜单栏选择“服务 > 编译构建”。

图 10-45 选择服务

步骤3 在编译构建页面单击“新建任务”，在基本信息中填入“任务名称”，并选择“归属项目”、“源码源”、“源码仓库”和“分支”，单击“下一步”。

图 10-46 新建构建任务

图 10-47 配置构建任务基本信息



步骤4 选择构建使用的模板（此示例使用的为java代码，故此步选择Maven模板）后，单击“下一步”。

图 10-48 选择构建模板

步骤5 进入“构建步骤”页面后，删除“上传软件包到软件发布库”步骤。

图 10-49 删除“上传软件包到软件发布库”步骤

步骤6 选择“添加步骤”，单击“文件上传”，选择“制作镜像并推送到SWR仓库”，单击“添加”。

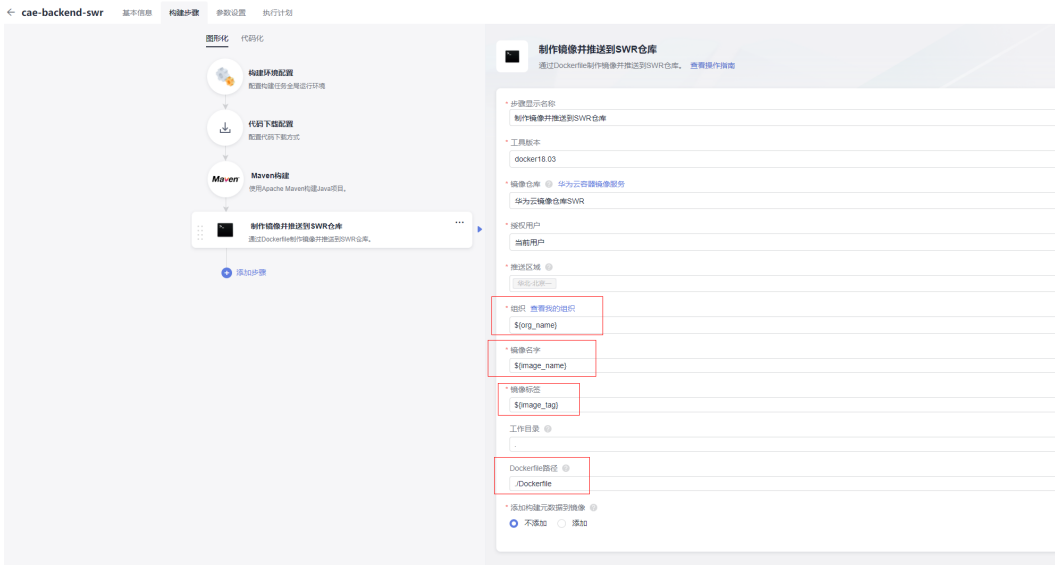
图 10-50 添加步骤

步骤7 进入“参数设置”页面，参考图10-51配置构建参数和默认值，并开启运行时设置。“org_name”为上传到SWR镜像仓库的组织，“image_name”为上传到SWR镜像仓库的镜像名称，“image_tag”为镜像标签或镜像版本。

图 10-51 配置自定义参数

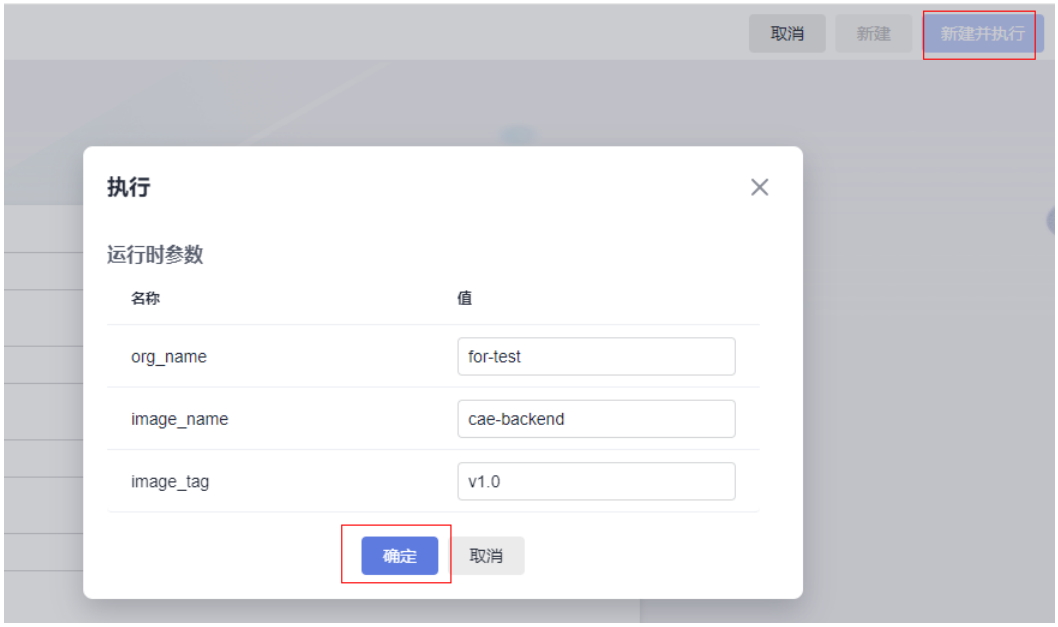
步骤8 进入“构建步骤”页面，配置“制作镜像并推送到SWR仓库”步骤，配置组织使用参数“${org_name}”，镜像名字使用参数“${image_name}”，镜像标签使用参数“${image_tag}”，工作目录使用默认值，Dockerfile路径按照代码仓中的路径来，示例中的Dockerfile在代码仓根目录。

图 10-52 配置“制作镜像并推送到 SWR 仓库”步骤



步骤9 单击右上角“新建并执行”，确认运行时参数无误后，单击“确定”。

图 10-53 执行构建任务



步骤10 在“构建历史”页面选择构建任务查看构建日志。

步骤11 登录SWR控制台，在“我的镜像”中查看镜像已上传成功。

----结束

创建流水线构建后升级 CAE 组件

创建codearts流水线：

步骤1 返回Codearts控制台首页。

步骤2 在菜单栏选择“服务 > 流水线”，在“流水线”页面单击“新建流水线”。

图 10-54 选择服务

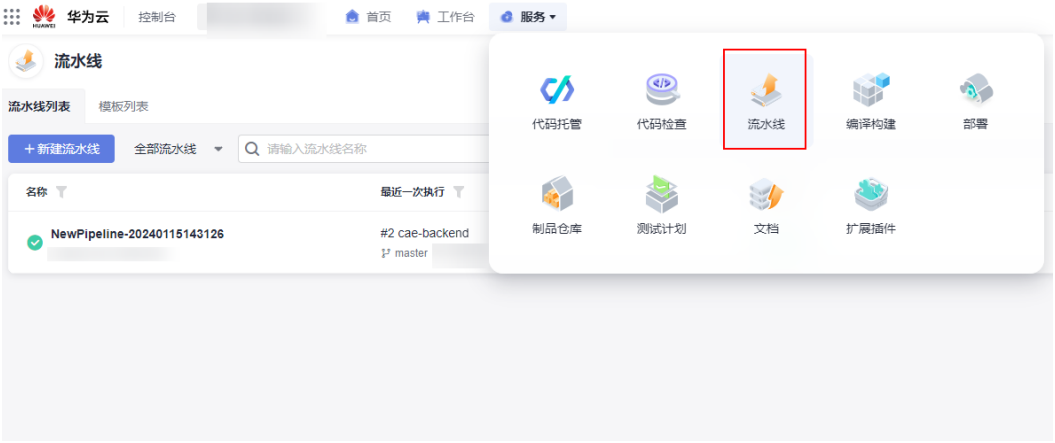
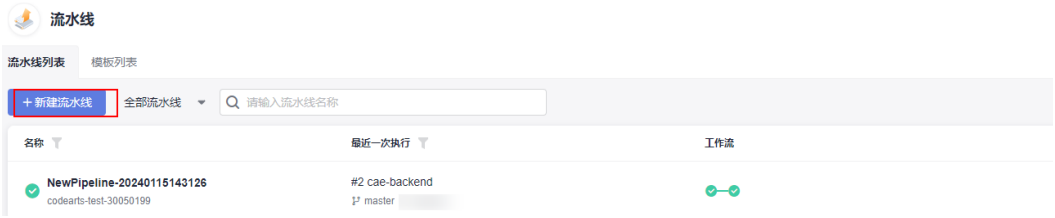


图 10-55 新建流水线



步骤3 在“新建流水线”界面，输入“名称”并选择“流水线源”、“代码仓”和“默认分支”，单击“下一步”。

图 10-56 配置流水线基本信息

步骤4 模板选择“全部 > 空模板”，单击“确定”。

图 10-57 选择模板

步骤5 进入“任务编排”页面后，单击“新建阶段”。单击 将“阶段_1”改名为“构建”，“阶段_2”改名为“部署”。

图 10-58 新建流水线阶段

步骤6 新建构建任务。

- 在“构建”阶段，单击“新建任务”。
  a. 在“新建任务”中，单击“构建”，选择“Build构建”任务，单击“添加”。

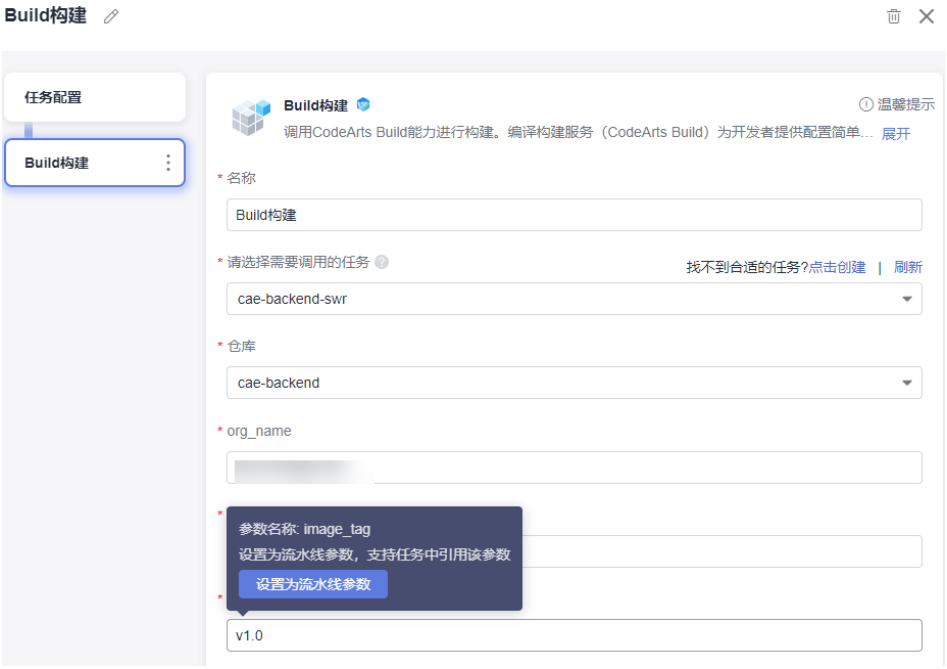
图 10-59 新建构建任务

图 10-60 添加“Build 构建”任务

b. 选择前一步创建的“编译构建任务”和“仓库”，并将“org_name”、“image_name”和“image_tag”都设置为流水线参数，单击“确定”。

图 10-61 image_name

图 10-62 image_tag



c. 进入“参数设置”页面，添加自定义参数，如图所示，取值见**参数说明**。

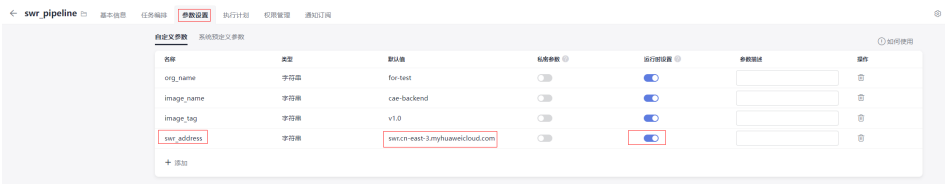
“org_name”：上传到SWR镜像仓库的组织，如for-test。

“image_name”：上传到SWR镜像仓库的镜像名称，如cae-backend。

“image_tag”：镜像标签或镜像版本，如v1.0。

“swr_address”：SWR镜像仓库地址，格式：swr.区域项目名称.myhuaweicloud.com。

图 10-63 添加自定义参数



- 在“部署”阶段，单击“新建任务”。

a. 在搜索框中输入“CAE”，并单击 。选择“CAE发布”插件，单击“添加”。

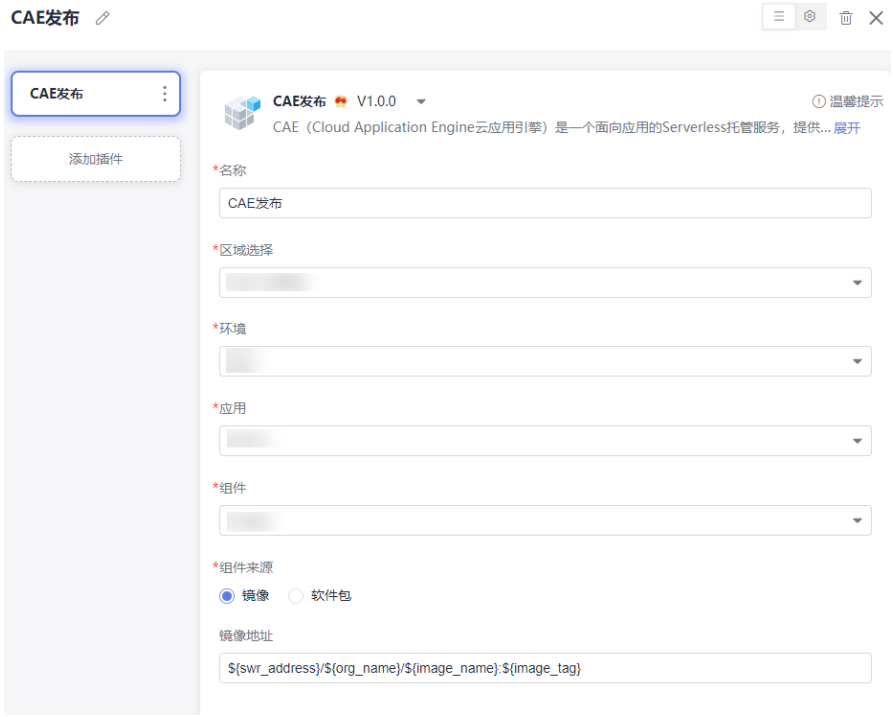
图 10-64 添加部署插件

b. 配置插件参数完成后，单击“确定”。

- 区域选择：待部署的区域，与当前CodeArts在同一个Region。

- 环境：选择组件所属环境。
- 应用：选择组件所属应用。
- 组件：选择待升级的组件（需要选择在CAE中使用镜像部署的组件）。
- 组件来源：选择“镜像”。
- 镜像地址：取值见**参数说明**。

图 10-65 配置插件参数



c. 单击“保存并执行”，进入执行配置页面，确认执行配置信息无误后，单击“执行”。

步骤7 查看流水线执行结果。

图 10-66 查看流水线执行结果

步骤8 **登录CAE控制台**，在组件列表查看组件状态，“最近一次变更状态/时间”列显示状态为“升级成功”，则表示Codearts流水线执行升级成功。

图 10-67 查看组件状态

----结束

参数说明

表 10-3 参数说明

参数名称	是否必须	参数类型	描述
org_name	是	String	镜像上传到SWR的组织名称，可以登录**SWR控制台**查看。
image_name	是	String	上传到SWR的镜像名称。
image_tag	是	String	上传到SWR的镜像版本。
swr_address	是	String	值的格式：swr.区域项目名称.myhuaweicloud.com，区域项目名称就是region，参考**参数值获取**。
镜像地址	是	String	上传到SWR的镜像地址，值填写：${swr_address}/${org_name}/${image_name}:${image_tag}。

11 通过配置 PromQL 实现自定义弹性伸缩

假设有一个名为my_component的组件，组件所处环境为my_environment，所处应用为my_application。

假设该组件提供自定义指标 http_requests_total，表示http请求总量，本文以该指标为例，介绍如何使用PromQL。

查询指标

使用以下PromQL语句，查询最新一条指标数据：

```
http_requests_total{environment_name="my_environment",application_name="my_application",component_name="my_component"}
```

若查询最近5分钟的所有指标数据：

```
http_requests_total{environment_name="my_environment",application_name="my_application",component_name="my_component"}[5m]
```

处理查询到的指标

查询指标中查询的指标数据通常有多条，例如组件有多个实例，则指标数据也有多条，或查询了一段时间的指标数据，该段时间内采集多条数据。

伸缩策略中的PromQL必须返回单个值，因此需要处理查询得到的指标数据，以得到单个值，示例如下：

查询最新一条指标，并求平均值，得到所有实例的http请求总数的平均值：

```
avg(http_requests_total{environment_name="my_environment",application_name="my_application",component_name="my_component"})
```

查询最近5分钟的所有指标数据，获取变化值（即增长值），并求平均，得到5分钟内平均每个实例的http请求增长数：

```
avg(delta(http_requests_total{environment_name="my_environment",application_name="my_application",component_name="my_component"}[5m]))
```

更多请参考**PromQL官方文档**和**PromQL官方示例**。

12 配置 CAE 对接 DEW，帮助应用从 DEW 获取加密凭据

12.1 概述

应用场景

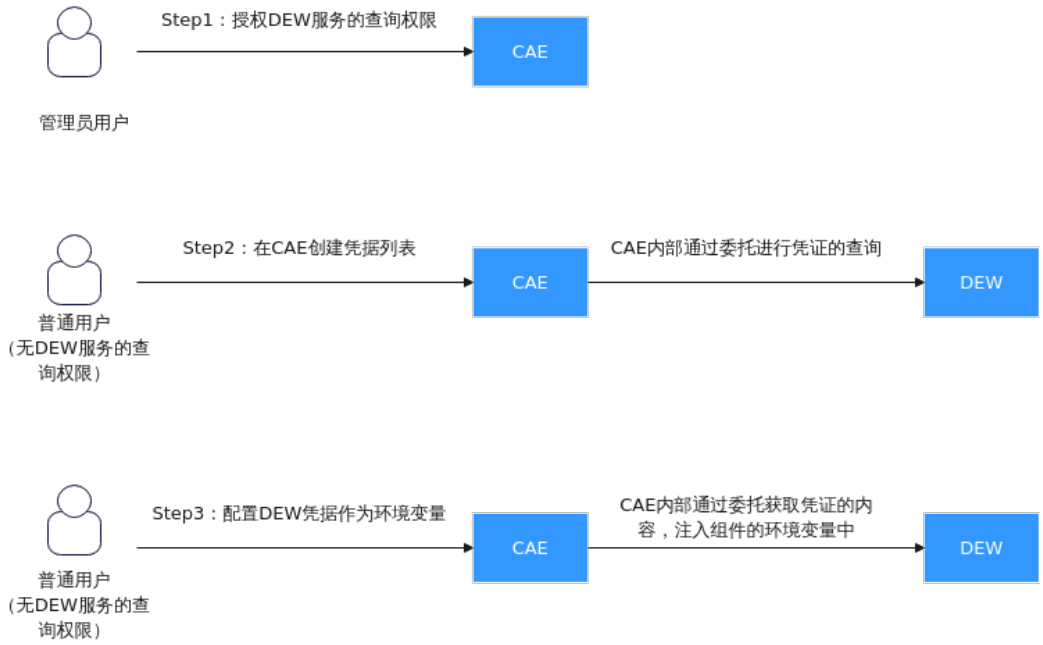
CAE提供对接DEW帮助应用从DEW获取凭据的功能，简化应用与DEW对接的复杂度，并提升安全性。

当您的代码里需要注入访问密钥、SSH Key等凭据作为环境变量时，可以通过配置DEW凭据来保证密钥的安全性。

如何创建及管理凭据，请参考**凭据管理**。

方案架构

图 12-1 凭据注入环境变量的流程图



约束限制

此功能需要您授权KMS CMKFullAccess以及CSMS ReadOnlyAccess到cae_trust委托中，具体操作请参考**给用户组授权**。

在启用此功能前需要使用管理员用户进行DEW授权。

12.2 配置凭据，通过环境变量注入

本实践通过添加对应DEW凭据，以环境变量方式注入到组件内，帮您实现数据保护。

创建 DEW 凭据

步骤1 登录DEW控制台。

步骤2 在左侧导航树中，选择“凭据管理”，进入“凭据管理”页面。

步骤3 单击“创建凭据”，在弹框中参考**表12-1**填写参数。

表 12-1 凭据配置参数说明

参数名称	参数说明
凭据类型	选择通用凭据。
凭据名称	填写创建凭据的名称。本实例凭证名称为db。

参数名称	参数说明
企业项目	创建凭据时，凭据绑定企业项目ID。 本实例选择default。
设置凭据值	选择明文，输入123456。
描述信息	本实践无须填写。
KMS加密	可选择默认密钥“csms/default”
关联事件	本实践暂不关联。

步骤4 单击“下一步”，通用凭据不支持设置轮转周期，再次单击“下一步”，确认创建的凭据信息。

步骤5 单击“确定”，凭据创建完成。

用户可在凭据列表查看已完成创建的凭据，凭据默认状态为“启用”。

----结束

添加凭据配置

步骤1 登录CAE控制台。

步骤2 在左侧导航栏中选择“系统设置”。

如未授权KMS CMKFullAccess和CSMS ReadonlyAccess，请使用管理员账号进行委托授权。

图 12-2 授权委托

步骤3 单击“凭据配置”模块中的“编辑”，进入“凭据配置”页面。

步骤4 单击“创建凭据配置”，在弹框中选择**创建DEW凭据**中已创建的凭据以及所需版本。

图 12-3 新增凭据

步骤5 单击“确定”，完成凭据配置。

----结束

配置环境变量

步骤1 在左侧导航栏中选择“组件配置”。

步骤2 在“组件配置”界面，选择需要配置的组件。

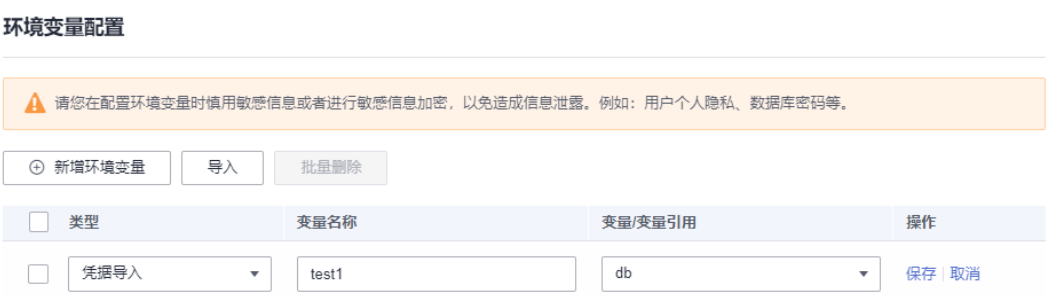
步骤3 单击“环境变量”模块中的“编辑”，进入“环境变量配置”页面。

步骤4 单击“新增环境变量”，考表12-2进行参数配置。

表 12-2 环境变量配置

参数	参数说明
类型	选择“凭据导入”。
变量名称	环境变量的名称，例如test1。 名称必须唯一，不可重复。
变量/变量引用	在下拉框中选择添加凭据配置中已创建的凭据配置。

图 12-4 配置环境变量



步骤5 在“操作”列单击“保存”，并在“环境变量配置”页面单击“确定”，完成环境变量配置添加。

步骤6 单击页面上方“生效配置”。

在右侧弹框中确认配置信息，并单击“确定”，使配置生效。

----结束

查看配置生效

步骤1 在左侧导航栏中选择“实例列表”。

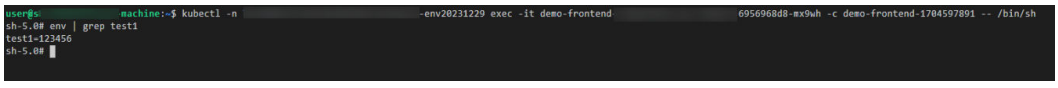
步骤2 在“实例列表”页面上方的下拉框中选择环境、应用及待操作的组件。

步骤3 选择待操作实例，在“操作”列单击“远程登录”。

步骤4 查看具体环境变量，与DEW凭据中设置的凭据一致。

图 12-5 DEW 服务中的凭据值

图 12-6 远程登录里的环境变量



----结束

12.3 屏蔽子账户读取密钥的权限，实现密钥保护

当您使用多账号管理CAE环境时，可以根据企业的业务组织，在您的华为云云服务账号中，给企业中不同职能部门的员工创建子账号，并根据职能设置不同的访问权限，以达到用户之间的权限隔离。

例如：分别为开发及测试创建开发账号与测试账号，测试人员无需感知敏感信息，为保护敏感信息，您可以屏蔽测试账号获取敏感信息的权限。

本最佳实践指导您通过屏蔽子账号DEW服务的所有权限以及CAE远程登录的权限，来控制子用户读取密钥。

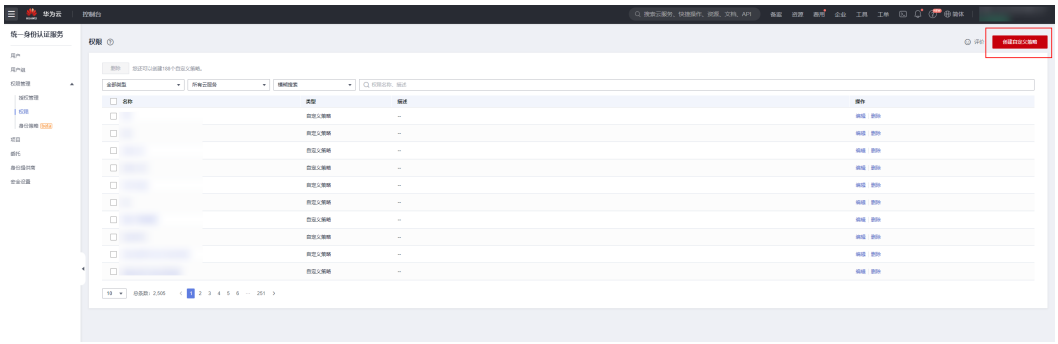
创建自定义策略

步骤1 登录“统一认证服务”控制台。

步骤2 在左侧导航栏中选择“权限管理 > 权限”。

步骤3 单击“创建自定义策略”，进入“创建自定义策略”页面。

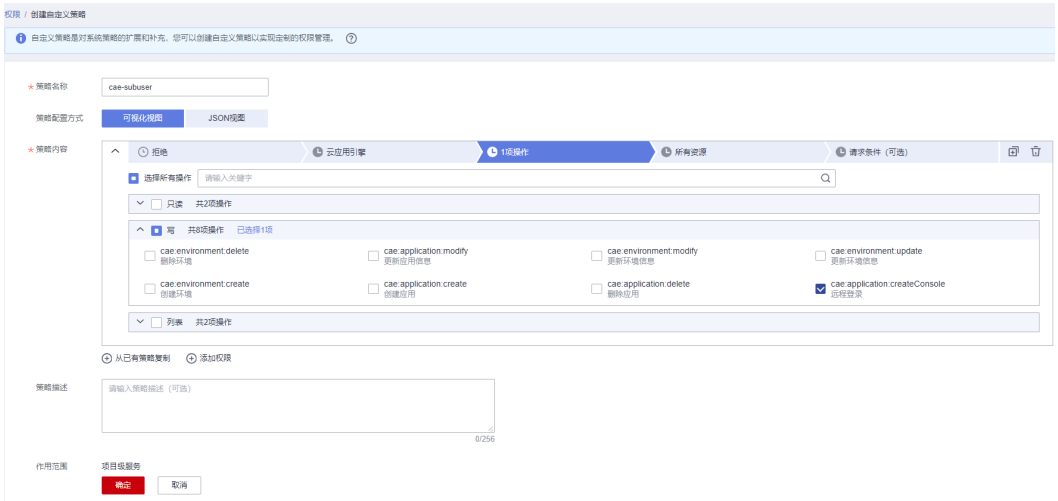
图 12-7 进入统一认证服务页面



步骤4 配置自定义策略。

- 策略名称：填写cae-subuser。
- 策略内容：选择“拒绝”。
- 服务：选择“云应用引擎”。
- 操作：勾选“cae:application:createConsole”权限。

图 12-8 创建自定义策略



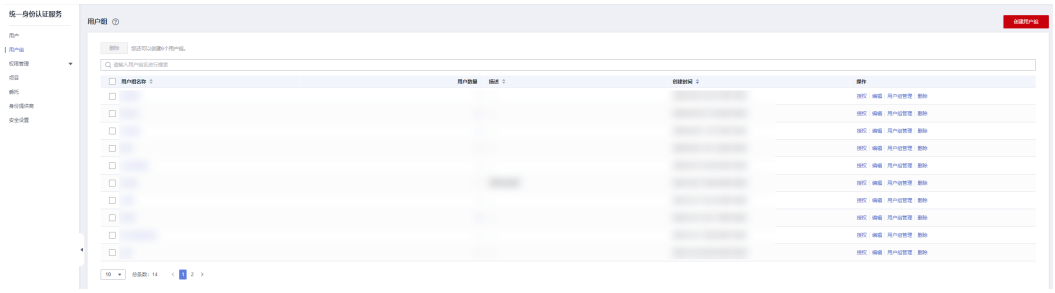
步骤5 单击“确定”，完成自定义策略创建。

----结束

创建用户组并授权

步骤1 在左侧导航栏中选择“用户组”，单击“创建用户组”。

图 12-9 创建用户组

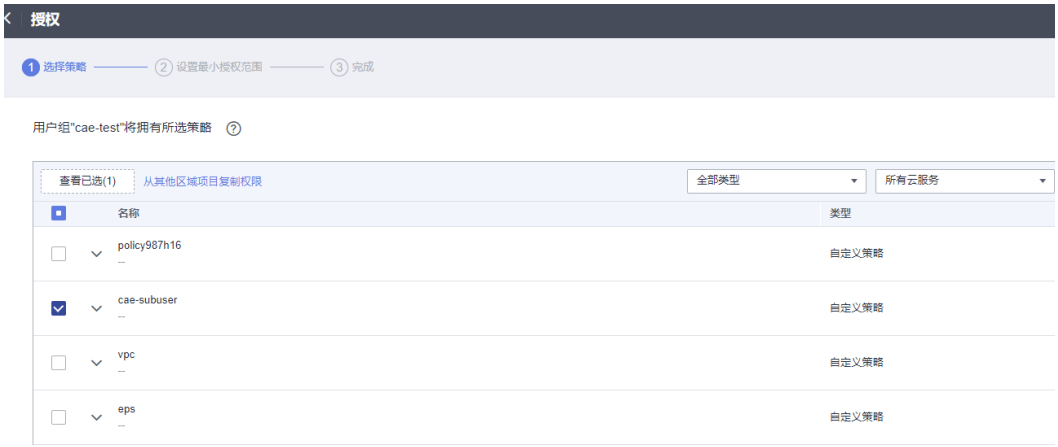


步骤2 输入“用户组名称”，例如cae-test，单击“确定”，完成用户组创建。

步骤3 在用户组列表中单击创建完成的用户组名称“cae-test”，进入用户组概览页面。

步骤4 在“授权记录”页签中单击“授权”，选择**创建自定义策略**中创建的自定义策略。

图 12-10 授权自定义策略



步骤5 单击“下一步”，选择授权范围方案为“所有资源”。

步骤6 单击“确定”，完成授权。

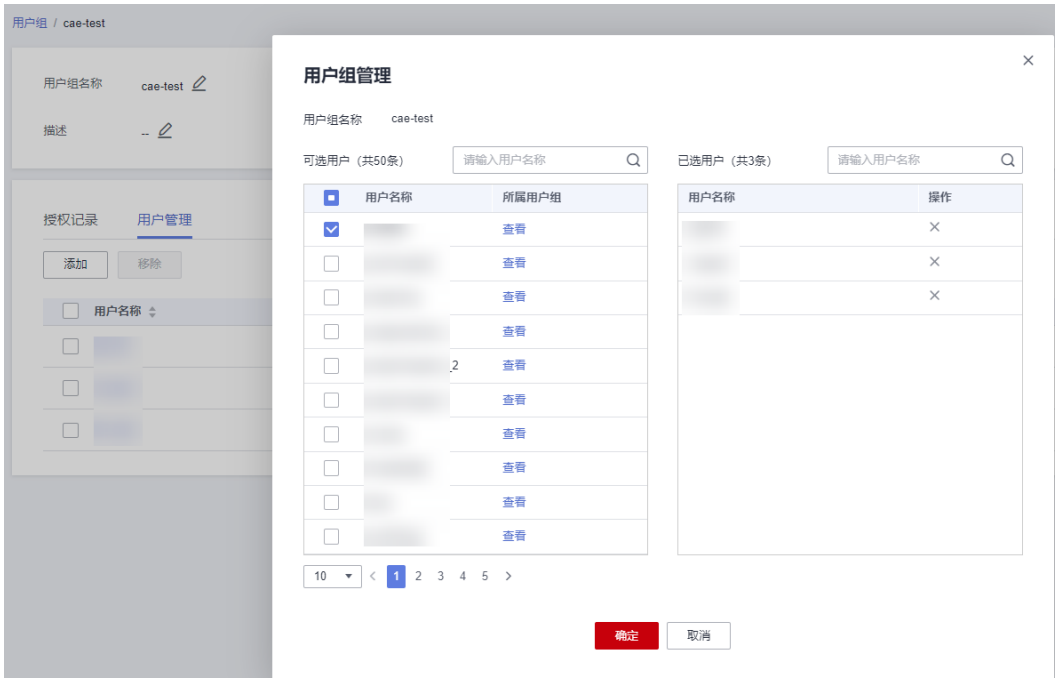
待授权完成后，单击“完成”，返回用户组概览页面。

图 12-11 完成用户组授权

步骤7 单击“用户管理”，进入“用户组管理”页面。

步骤8 在用户列表中勾选需要控制权限的子用户，单击“确定”。

图 12-12 配置用户管理



----结束

验证子账号权限

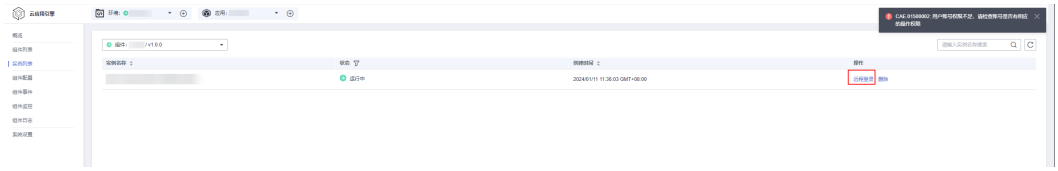
步骤1 使用**步骤8**的子用户账号登录CAE控制台。

步骤2 进行**添加凭据配置**、**配置环境变量**等操作，可正常使用。

步骤3 在左侧导航栏中选择“实例列表”。

步骤4 选择待操作实例，在“操作”列单击“远程登录”，无法查看凭据具体内容。

图 12-13 当前子用户无远程登录的权限



步骤5 单击左上角 ☰，在服务列表单击“数据加密控制台 DEW”，切换至DEW控制台。

步骤6 在左侧导航栏中选择“凭据管理 > 凭据列表”，无法查看凭据具体内容。

图 12-14 当前子用户无 DEW 的委托权限

----结束

13 ASP.NET Core 应用部署到 CAE

CAE源码部署支持Docker运行时，您可以自行配置Dockerfile文件，在Dockerfile中安装构建环境，定义构建命令，以此支持更多编程语言的项目在CAE部署。

本文以ASP.NET Core应用为例。

前提条件

已**创建环境**和**创建应用**。

Fork 示例源码

使用您的账号登录GitHub，并Fork示例源码仓库到个人账号。

源码地址：**https://github.com/Azure-Samples/dotnetcore-docs-hello-world**。

Dockerfile 解析

示例仓库中的Dockerfile文件。

```
# 指定基础镜像为mcr.microsoft.com/dotnet/sdk:7.0，该基础镜像作为ASP.NET Core项目的构建环境
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
# 指定工作目录
WORKDIR /source

# 从构建主机上复制源码到基础镜像，并指定工作目录和构建命令
COPY . ./dotnetcore-docs-hello-world/
WORKDIR /source/dotnetcore-docs-hello-world
RUN dotnet publish -c release -o /app

# 指定新的基础镜像为mcr.microsoft.com/dotnet/aspnet:7.0，并将第二步生成的构建产物复制到新的基础镜像中
FROM mcr.microsoft.com/dotnet/aspnet:7.0
WORKDIR /app
COPY --from=build /app ./

# 设置环境变量PORT，并声明容器端口为80
ENV PORT 80
EXPOSE 80

# 指定容器启动命令
ENTRYPOINT ["dotnet", "dotnetcoresample.dll"]
```

设置 GitHub 仓库授权

设置GitHub仓库授权，使构建工程、应用组件等可以使用授权信息访问GitHub源码仓库。

步骤1 登录CAE控制台。

步骤2 在左侧导航栏中选择“系统设置”。

步骤3 单击“源码仓库授权”模块中的“编辑”，进入“已授权源码仓库”页面。

步骤4 单击“新增授权”，参考表13-1选择需要的源码仓库，并配置参数。

表 13-1 授权参数说明

参数	说明
授权名称	授权名称，创建之后不可更改
仓库类型	支持以下官方仓库类型： 1. 选择GitHub代码仓库。 2. 选择“授权方式”。 – OAuth：请单击“使用OAuth授权”，根据页面提示完成授权创建。 – 私人令牌：输入已获取到的私人令牌。

步骤5 单击“确认”，完成GitHub代码仓库授权。

图 13-1 授权 GitHub 代码仓库

----结束

创建及配置组件

步骤1 在左侧导航栏选择“组件列表”。

步骤2 在页面上方，下拉选择已创建的应用和环境，单击“新增组件”。

步骤3 参考表13-2设置组件信息。

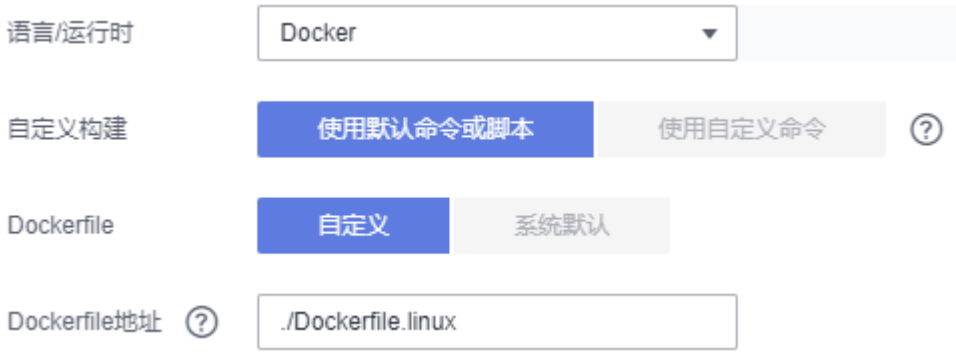
表 13-2 组件基本信息

参数	说明
组件名称	新建组件的名称。本实践输入名称为“test-dotnet”。
版本号	组件的版本号。 本实践版本号为1.0.0。
实例规格	选择实例规格，例如：0.5core、1GiB。
实例数量	输入实例数为1。
代码源	选择“源码仓库 > GitHub”。选择设置GitHub仓库授权中创建好的授权，选择示例代码仓。

图 13-2 配置基本信息

步骤4 语言/运行时选择“Docker”，Dockerfile地址填写./Dockerfile.linux。

图 13-3 配置 Dockerfile



步骤5 单击“配置组件”，跳转至“组件配置”页面。

步骤6 找到“访问方式”，单击“编辑”。

步骤7 在“从环境外部访问本组件”页面，选择“负载均衡配置”，并单击“添加负载均衡配置”。

参考图13-4配置负载均衡。

图 13-4 配置负载均衡

步骤8 单击“确定”，完成负载均衡配置。

步骤9 单击组件配置页面上方的“配置并部署组件”，在右侧弹框中单击“确定”，待部署执行完成后，配置生效。

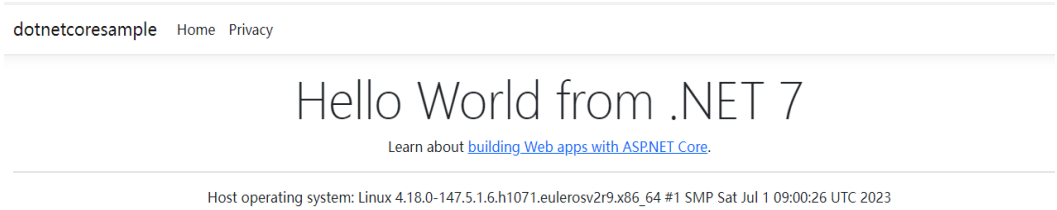
----结束

查看应用

步骤1 在左侧导航栏，选择组件列表，返回组件列表页面。

步骤2 单击“test-dotnet”组件“访问地址”列的公网访问地址。

步骤3 您还可以访问应用的静态页面。



----结束

14 通过企业路由打通网络

CAE提供CAE环境访问租户侧VPC的能力，本章最佳实践将指导您在关闭公网后，如何通过企业路由打通网络。

步骤一：创建企业路由并添加连接

创建企业路由器，并在企业路由中添加连接，本案例的vpc-wc为CAE环境所属VPC（网段为192.168.0.0/16），vpc-10为租户侧VPC（网段为10.0.0.0/16）。

步骤1 进入**企业路由器控制台**。

步骤2 单击“创建企业路由器”，在创建企业路由器页面，按照如下参数示例进行设置，参数的具体含义请参考**创建企业路由器**。

表 14-1 创建企业路由器配置参数

参数	本案例示例
区域	亚太-新加坡
可用区	可用区3
企业项目	default
名称	er-878b
ASN	64513
默认路由表关联	开启
默认路由表传播	开启
自动接受共享连接	关闭
标签	-
描述	-

步骤3 设置完成后，单击“立即创建 > 提交”。

步骤4 在企业路由器页面，单击刚才创建的企业路由器的名称“er-878b”，选择“连接”页签。

步骤5 在“连接”页签下，单击“添加连接”，在添加连接页面，按照**表14-2**和**表14-3**参数示例进行设置，参数的具体含义请参考**在企业路由器中添加VPC连接**。

表 14-2 添加连接至租户侧 VPC

参数	本案例示例
名称	er-attach-374d
连接类型	虚拟私有云（VPC）
连接资源	• 虚拟私有云：vpc-10 • 子网：wtr-test(10.0.1.0/24)
配置连接侧路由	开启
高级配置-描述	-
高级配置-标签	-

表 14-3 添加连接至 CAE 环境所属 VPC

参数	本案例示例
名称	er-attach-f855
连接类型	虚拟私有云（VPC）
连接资源	• 虚拟私有云：vpc-wc • 子网：wtr-test(10.0.1.0/24)
配置连接侧路由	开启
高级配置-描述	-
高级配置-标签	-

步骤6 设置完成后，单击“立即创建”。

----结束

步骤二：创建 NAT 网关

购买公网NAT网关，其中虚拟私有云为CAE环境所属VPC，在本案例中，CAE环境所属VPC为vpc-wc。

步骤1 进入**公网NAT网关控制台**。

步骤2 单击“购买公网NAT网关”，在购买公网NAT网关页面，按照如下参数示例进行设置，参数的具体含义请参考**购买公网NAT网关**。

表 14-4 购买公网 NAT 网关配置参数

参数	本案例示例
区域	华北-北京四
计费模式	请您按照业务需要选择计费模式。
规格	请您按照业务需要选择公网NAT网关的规格。
名称	nat-7a60
虚拟私有云	vpc-wc
子网	cae-subnet
企业项目	default
高级配置-描述	-
高级配置-标签	-
购买时长	请您按照业务需要选择购买时长。

步骤3 设置完成后，单击“下一步 > 去支付”，支付后完成创建公网NAT网关。

步骤4 在公网NAT网关列表，找到刚才创建的网关“nat-7a60”，单击右侧的“设置规则”。

步骤5 在SNAT规则页签中，单击“添加SNAT规则”，在添加SNAT规则页面，按照如下参数示例进行设置，参数的具体含义请参考**添加SNAT规则**。

表 14-5 添加 SNAT 规则配置参数

参数	本案例示例
NAT网关名称	nat-7a60
使用场景	虚拟私有云
网段	cae-subnet（192.168.0.0/24）
公网IP类型	100.95.154.131
监控	-
描述	-

步骤6 设置完成后，单击“确定”。

----结束

步骤三：租户侧创建 ECS 节点

租户侧创建的ECS节点，所属虚拟私有云为vpc-10，该节点弹性公网IP为100.93.0.237，私有IP为10.0.0.80。

步骤1 进入**弹性云服务器控制台**。

步骤2 单击“购买弹性云服务器”，在购买弹性云服务器页面，请您按照业务需要设置购买参数，参数的具体含义请参考**购买ECS指南**。

说明

在**网络**设置模块，设置“虚拟私有云”为租户侧VPC，在本案例中，租户侧VPC为vpc-10，该节点弹性公网IP为100.93.0.237，私有IP为10.0.0.80。

步骤3 设置完成后，单击“立即购买”。

----结束

步骤四：创建路由表和路由

在CAE所属VPC（vpc-wc）的路由表中添加路由，其中10.0.0.0/16为私网网段，100.93.0.0/16为公网网段。

步骤1 进入**企业路由器控制台**。

步骤2 单击刚才创建的企业路由器的名称“er-878b”，选择“路由表”页签。

步骤3 单击“创建路由表”，按照如下参数示例进行设置，参数的具体含义请参考**创建路由表**。

表 14-6 创建路由表配置参数

参数	本案例示例
名称	rtb-vpc-wc
描述	-
标签	-

步骤4 设置完成后，单击“确定”。

步骤5 在路由表页面，选择刚才创建的路由表“rtb-vpc-wc”，在“路由”页签下，单击“创建路由”，您需要按照如下两个参数示例**表14-7**和**表14-8**进行设置，参数的具体含义请参考**创建静态路由**。

表 14-7 创建路由配置参数（下一跳为企业路由器）

参数	本案例示例
目的地址	10.0.0.0/16
黑洞路由	不开启
连接类型	虚拟私有云（VPC）
下一跳	er-878b
描述	-

表 14-8 创建路由配置参数（下一跳为 NAT 网关）

参数	本案例示例
目的地址	100.93.0.0/16
黑洞路由	不开启
连接类型	虚拟私有云（VPC）
下一跳	nat-7a60
描述	-

步骤6 设置完成后，单击“确定”。

----结束

步骤四：在 CAE 环境添加目的访问网络地址

步骤1 进入云应用引擎控制台。

步骤2 在左侧导航栏选择“系统设置”。

步骤3 找到“系统网络配置”，单击“编辑”。

步骤4 单击“添加目的网络地址”，勾选下一跳为NAT网关的目标网络地址，单击“确定”，完成访问公网路由配置。

步骤5 再单击“添加目的网络地址”，勾选下一跳为企业路由器的目的网络地址，单击“确定”，完成访问私网路由配置。

----结束

