应用管理与运维平台

最佳实践

文档版本 01

发布日期 2024-09-25

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 使用 ServiceStage 托管和治理天气预报微服务应用

1.1 使用 ServiceStage 托管天气预报微服务应用概述

天气预报微服务应用提供天气预报、紫外线和天气湿度展示等功能。本文通过天气预报应用，展示了微服务架构设计理念的应用场景，以及使用ServiceStage管理运行环境、构建应用和治理微服务的最佳实践。

天气预报应用由前端应用和后端应用组成。前端应用组件weathermapweb采用Node.js进行开发，实现前端应用发现后端应用。后端应用分别采用Java Chassis、Spring Cloud微服务开发框架实现，包括weather、forecast、fusionweather、weather-beta、edge-service等微服务组件。

其中：

- weathermapweb是一个基于Node.js语言开发的界面微服务。
- weather是提供指定城市当前的天气情况的微服务。
- forecast是提供指定城市未来几天天气情况预测的微服务。
- fusionweather是一个聚合微服务，通过访问weather和forecast服务，提供全方位的天气预报功能。
- weather-beta是weather微服务的新版本，新增了查询指定城市紫外线情况的功能。
- edge-service为所有其它微服务的统一入口。

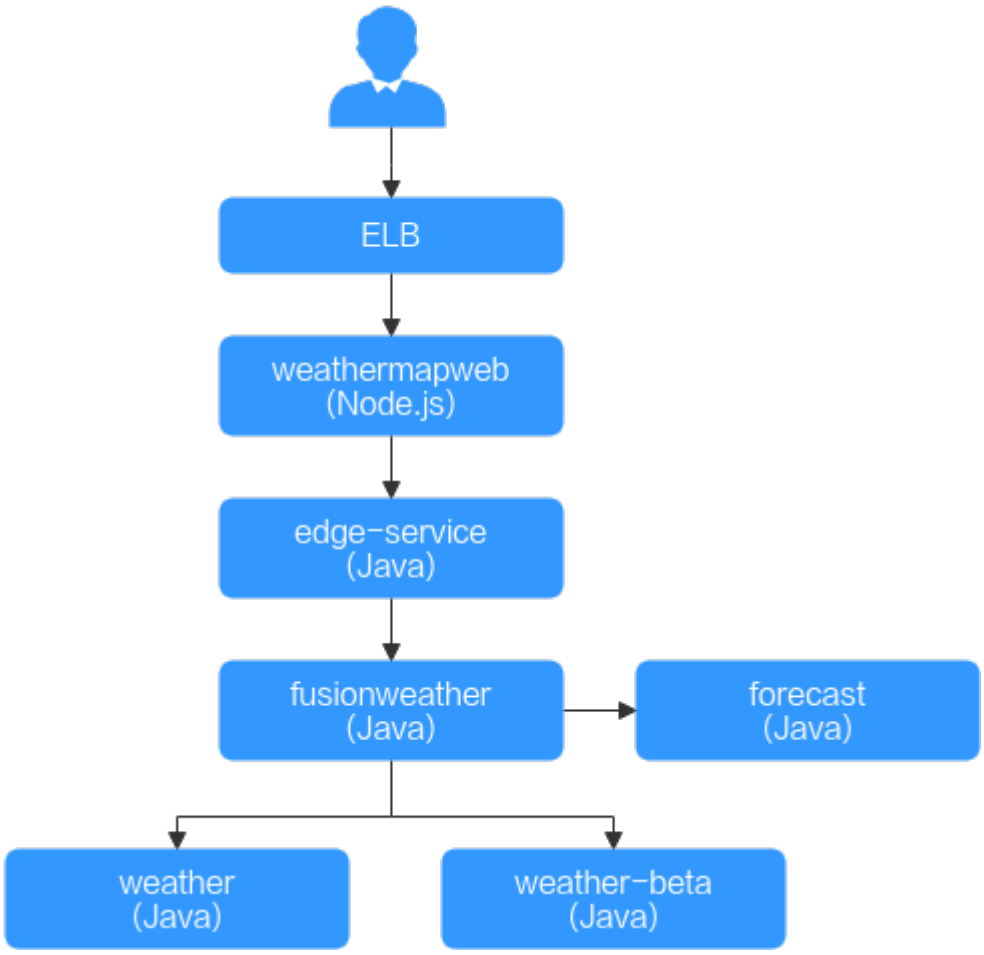
天气预报各个后端应用组件详细说明请参见**表1-1**。

表 1-1 天气预报组件说明

微服务开发框架	组件名称
Java Chassis	weather
	forecast
	fusionweather

微服务开发框架	组件名称
	weather-beta
	edge-service
	weathermapweb
Spring Cloud	weather
	forecast
	fusionweather
	weather-beta
	edge-service
	weathermapweb

天气预报的逻辑组网和调用关系图如下：



ServiceStage支持通过源码、软件包方式部署并接入Java Chassis、Spring Cloud微服务开发框架开发的微服务应用。

本最佳实践使用基于Java Chassis微服务开发框架开发的天气预报应用，提供了**使用源码部署天气预报微服务**和**使用软件包部署天气预报微服务**两种微服务应用部署方法，为您展示使用ServiceStage托管和治理微服务应用的能力。

1.2 使用源码部署天气预报微服务

1.2.1 源码部署前准备

资源准备

为了方便后续的操作，需要您提前准备好如下资源：

1. 创建一个虚拟私有云VPC，请参考**创建虚拟私有云和子网**。
2. 创建一个未开启安全认证的ServiceComb引擎专享版，请参考**创建微服务引擎**。

   ServiceComb引擎所在VPC为**1**所创建的VPC。如果VPC不一致，需正确配置VPC连通。
3. 创建一个CCE集群（如果只是试用场景，“集群管理规模”选择“50节点”，“高可用”选择“否”即可），请参考**购买集群**。
   - CCE集群所在VPC为**1**所创建的VPC。
   - 集群中至少包含1个规格为8vCPUs、16GB内存或者2个规格为4vCPUs、8GB内存的ECS节点，并且绑定弹性公网IP。为CCE集群添加节点，请参考**创建节点**。
   - CCE集群不能被其他环境绑定。
4. 本例基于ServiceStage绑定GitHub源码仓库，实现源码构建、归档、应用创建，需要先到**GitHub**官网注册账号。

Fork 天气预报源码

使用您的账号登录GitHub，并Fork天气预报源码。源码地址：**https://github.com/servicestage-demo/weathermap.git**。

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ServiceStage控制台。

步骤2 选择“持续交付 > 仓库授权 > 新建授权”，参考下表配置授权信息。

参数	说明
*授权名称	授权名称保持默认，创建之后不可更改。
*仓库授权	1. 选择GitHub代码仓库。 2. “授权方式”选择“OAuth”。 3. 单击“使用OAuth授权”，根据页面提示完成访问GitHub源码仓库授权。

----结束

创建组织

步骤1 选择“部署源管理 > 组织管理”。

步骤2 单击“创建组织”，在弹出的页面中填写“组织名称”，例如：org-test。

步骤3 单击“确定”。

图 1-1 创建组织

----结束

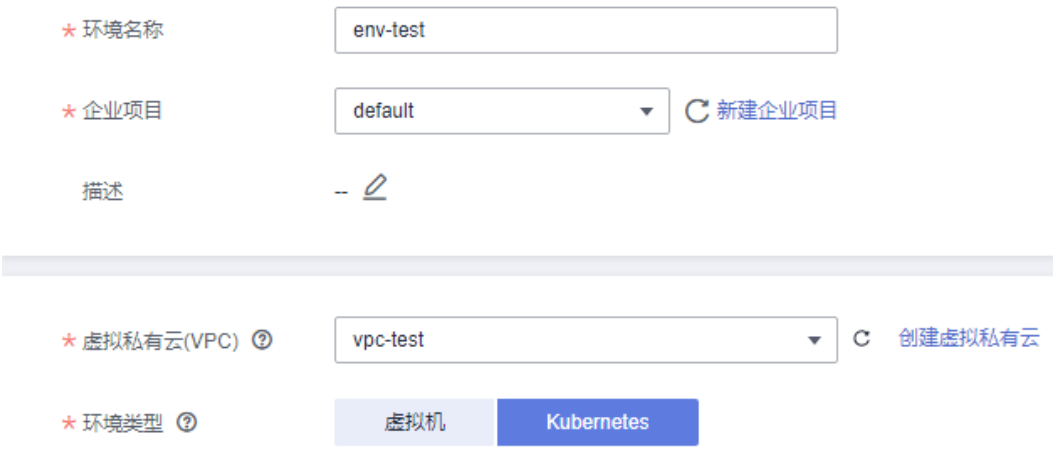
创建环境

步骤1 选择“环境管理 > 创建环境”，参照下表设置环境信息。

参数	参数说明
环境名称	输入环境名称（例如：env-test）。
企业项目	设置企业项目。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 已**开通企业项目**后可以使用。
描述	保持默认。
虚拟私有云(VPC)	选择**资源准备**中已准备好的虚拟私有云VPC。 **说明** 环境创建完成后，不支持修改VPC。

参数	参数说明
环境类型	选择Kubernetes。

图 1-2 设置环境信息



步骤2 单击“立即创建”，进入环境详情页面。

步骤3 在“资源”下左侧列表，选择“计算”资源类型下的“云容器引擎 CCE”，单击“立即绑定”。

步骤4 在弹出的对话框中，选择**资源准备**中已创建的CCE集群资源，单击“确定”。

步骤5 在“资源”下左侧列表，选择“中间件”资源类型下的“ServiceComb引擎”，单击“纳管资源”。

步骤6 在弹出的对话框中，勾选**资源准备**中已创建的ServiceComb引擎资源，单击“确定”。

----结束

创建应用

步骤1 单击左上角 < ，返回“环境管理”页面。

步骤2 选择“应用管理 > 创建应用”，设置应用基本信息。

1. “应用名称”：填写weathermap。

   说明

   如果应用列表中存在同名应用，请参考**如何处理当前环境下已存在同名的天气预报微服务应用?** 处理。

2. “企业项目”：企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。

   已**开通企业项目**后可以使用。

步骤3 单击“确定”，完成应用创建。

图 1-3 创建应用

----结束

1.2.2 源码部署微服务

业务场景

基于ServiceStage可以方便快捷的将微服务部署到容器（如CCE）、虚拟机（如ECS），同时支持源码部署、jar/war包部署或docker镜像包部署。同时，ServiceStage支持Java、PHP、Node.js、Python等多种编程语言应用的完全托管，包括部署、升级、回滚、启停和删除等。

本实践中使用了Java开发的后台组件和Node.js开发的前台组件。

用户故事

在本实践中，您可以通过容器部署的方式部署应用并将微服务实例注册到ServiceComb引擎中，weathermap应用需要创建以下组件：

1. 前台组件：weathermapweb，基于Node.js语言开发的界面。
2. 后台组件：weather、fusionweather、forecast、edge-service，基于Java语言开发。

微服务部署有以下几个操作过程：

1. **创建并部署后台应用组件**
2. **设置edge-service组件访问方式**
3. **创建并部署前台组件**
4. **确认部署结果**
5. **添加访问方式**
6. **访问应用**

创建并部署后台应用组件

此处需要创建并部署4个应用组件：weather、forecast、fusionweather、edge-service，对应后台构建任务生成的4个软件包。

步骤1 登录ServiceStage控制台。

步骤2 单击“应用管理”，进入“应用管理”页面。

步骤3 单击**创建应用**时创建的应用名称（例如：weathermap）“操作”栏的“新增组件”。

步骤4 在“基本信息”区域，参考下表设置必填组件基本信息，其余参数保持默认。

参数	说明
组件名称	输入对应的后台组件名称（例如：weather）。
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。
所属环境	选择**创建环境**时创建的环境（例如：env-test）。
所属应用	选择**创建应用**时创建的应用（例如：weathermap）。

步骤5 在“组件包”区域，参考下表设置必填组件包参数，其余参数保持默认。

参数	说明
技术栈	组件技术栈类型选择Java。
源码/软件包	1. 选择“源码仓库”。 2. 选择“GitHub”。 3. “授权信息”选择**设置GitHub仓库授权**时创建的授权信息。 4. “用户名/组织”选择**Fork天气预报源码**时登录您的GitHub使用的用户名。 5. “仓库名称”选择已Fork到您的GitHub下的天气预报源码仓库的名称，例如：weathermap。 6. “分支”选择“master”。

步骤6 在“构建”区域，设置必填构建参数。

1. “编译命令”：保持默认。
2. “Dockerfile地址”：参考下表设置。

组件名称	Dockerfile地址
weather	./weather/
forecast	./forecast/
fusionweather	./fusionweather/
edge-service	./edge-service/

3. “组织”：选择**创建组织**时创建的组织名称。
4. “构建环境”：选择“使用当前环境构建”。
5. 其余参数，保持默认。

图 1-4 设置构建参数

步骤7 单击“下一步”。

步骤8 在“资源”区域，参考下表设置各组件“实例数”，其余参数设置保持默认。

组件名称	实例数
weather	2
forecast	1
fusionweather	1
edge-service	1

步骤9 绑定ServiceComb引擎。

说明

- 组件部署以后，微服务会注册到绑定的ServiceComb引擎。
- 所有组件需要注册到同一个ServiceComb引擎，才能互相发现。

1. 选择“云服务配置 > 微服务引擎”。
2. 单击“绑定微服务引擎”。
3. 选择当前环境下已纳管的ServiceComb引擎专享版。
4. 单击“确定”。

步骤10 单击“创建并部署”。

----结束

设置 edge-service 组件访问方式

步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 单击**创建应用**时创建的应用名称（例如：weathermap），进入“应用概览”页。

步骤3 在“组件列表”，单击edge-service所在行“外部访问地址”列的“设置”，进入“访问方式”页面。

步骤4 单击“TCP/UDP路由配置”区域的“添加服务”，参考下表设置参数。

参数	说明
服务名称	保持默认。
访问方式	选择“公网访问”。
访问类型	选择“弹性IP”。
服务亲和	保持默认。
协议	选择TCP。
容器端口	填写3010。
访问端口	选择“自动生成”。

图 1-5 设置 edge-service 组件访问方式

步骤5 单击“确定”，生成访问地址。

----结束

创建并部署前台组件

步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 单击**创建应用**时创建的应用名称（例如：weathermap）“操作”栏的“新增组件”。

步骤3 在“基本信息”区域，参考下表设置必填组件基本信息，其余参数保持默认。

参数	说明
组件名称	输入前台组件的名称：weathermapweb。

参数	说明
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。
所属环境	选择**创建环境**时创建的环境（例如：env-test）。
所属应用	选择**创建应用**时创建的应用（例如：weathermap）。

步骤4 在“组件包”区域，参考下表设置必填组件包参数，其余参数保持默认。

参数	说明
技术栈	组件技术栈类型选择Node.js。
源码/软件包	1. 选择“源码仓库”。 2. 选择“GitHub”。 3. “授权信息”选择**设置GitHub仓库授权**时创建的授权信息。 4. “用户名/组织”选择**Fork天气预报源码**时登录您的GitHub使用的用户名。 5. “仓库名称”选择已Fork到您的GitHub下的天气预报源码仓库的名称，例如：weathermap。 6. “分支”选择“master”。

步骤5 在“构建”区域，设置必填构建参数。

1. “编译命令”：保持默认。
2. “Dockerfile地址”：参考下表设置。

组件名称	Dockerfile地址
weathermapweb	./weathermapweb/

3. “组织”：选择**创建组织**时创建的组织名称。
4. “构建环境”：选择“使用当前环境构建”。
5. 其余参数，保持默认。

步骤6 单击“下一步”，添加环境变量。

1. 选择“容器配置 > 环境变量”。
2. 单击“添加环境变量”，参考下表添加环境变量。

类型	变量名称	变量/变量引用
手动添加	SERVICE_ADDR	**设置edge-service组件访问方式**中生成的访问地址。

步骤7 单击“创建并部署”。

----结束

确认部署结果

步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 选择“微服务引擎 > 微服务目录”。

步骤3 在微服务引擎下拉列表选择部署了微服务应用的ServiceComb引擎。

步骤4 在“微服务列表”页签的“全部应用”下拉列表中选择**创建应用**时创建的应用名称（例如：weathermap）。

如果各微服务实例数如下表所示，则部署成功。

组件名称	实例数
weather	2
forecast	1
fusionweather	1
edge-service	1

----结束

添加访问方式

步骤1 单击“应用管理”。

步骤2 单击**创建应用**时创建的应用名称（例如：weathermap），进入“应用概览”页。

步骤3 在“组件列表”，单击weathermapweb所在行“外部访问地址”列的“设置”，进入“访问方式”页面。

步骤4 单击“TCP/UDP路由配置”区域的“添加服务”，参考下表设置参数。

参数	说明
服务名称	保持默认。
访问方式	选择“公网访问”。
访问类型	选择“弹性IP”。
服务亲和	保持默认。
协议	选择TCP。
容器端口	填写3000。
访问端口	选择“自动生成”。

图 1-6 添加访问方式

步骤5 单击“确定”，生成访问地址。

图 1-7 访问地址

----结束

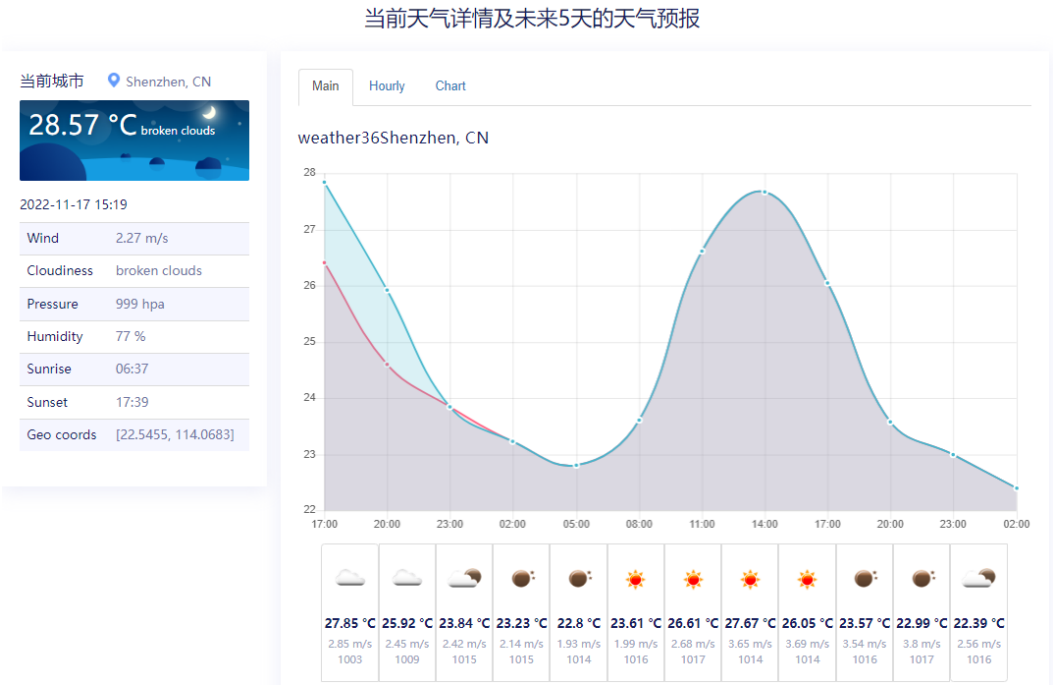
访问应用

步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 单击**创建应用**时创建的应用名称（例如：weathermap），进入“应用概览”页。

步骤3 在“组件列表”，单击weathermapweb所在行“外部访问地址”列的外部访问地址。出现以下页面表明天气预报微服务应用部署成功。

图 1-8 应用部署成功



说明

- 该数据为实时数据。
- 首次访问应用时，weather系统就绪需要一段时间。如果如上图所示页面没有出现，请持续刷新页面。

----结束

1.3 使用软件包部署天气预报微服务

1.3.1 软件包部署前准备

准备资源

为了方便后续的操作，需要您提前准备好如下资源：

1. 创建一个虚拟私有云VPC，请参考**创建虚拟私有云和子网**。
2. 创建一个未开启安全认证的2.4.0或以上版本的ServiceComb引擎专享版，请参考**创建微服务引擎**。

   ServiceComb引擎所在VPC为**1**所创建的VPC。如果VPC不一致，需正确配置VPC连通。
3. 创建一个CCE集群（如果只是试用场景，“集群管理规模”选择“50节点”，“高可用”选择“否”即可），请参考**购买集群**。
   - CCE集群所在VPC为**1**所创建的VPC。
   - 集群中至少包含1个规格为8vCPUs、16GB内存或者2个规格为4vCPUs、8GB内存的ECS节点，并且绑定弹性公网IP。为CCE集群添加节点，请参考**创建节点**。

- CCE集群不能被其他环境绑定。

4. 创建用于存储软件包的桶，请参考**创建桶**。

下载并上传天气预报组件软件包

步骤1 参考**表1-2**下载天气预报组件软件包到本地（本实践使用Java Chassis微服务开发框架开发的组件）。

表 1-2 天气预报组件软件包说明

组件微服务开发框架	组件名称	组件软件包名称	组件软件包下载说明
Java Chassis	weather	weather-1.0.0.jar	1. 进入**天气预报组件软件包仓库**。 2. 单击ServiceComb，进入使用Java Chassis微服务开发框架开发的天气预报组件软件包库。
	weather-beta	weather-beta-2.0.0.jar	
	forecast	forecast-1.0.0.jar	
	fusionweather	fusionweather-1.0.0.jar	
	edge-service	edge-service-1.0.0.jar	
	weathermapweb	weathermapweb.zip	
Spring Cloud	weather	weather-1.0.0.jar	1. 进入**天气预报组件软件包仓库**。 2. 单击Spring Cloud，进入使用Spring Cloud微服务开发框架开发的天气预报组件软件包库。
	weather-beta	weather-beta-2.0.0.jar	
	forecast	forecast-1.0.0.jar	
	fusionweather	fusionweather-1.0.0.jar	
	edge-service	edge-service-1.0.0.jar	
	weathermapweb	weathermapweb.zip	

步骤2 将下载到本地的天气预报组件软件包上传到**准备资源**中准备好的桶中备用。

上传软件包，请参考**上传对象**。

----结束

创建组织

步骤1 登录ServiceStage控制台。

步骤2 选择“部署源管理 > 组织管理”。

步骤3 单击“创建组织”，在弹出的页面中填写“组织名称”，例如：org-test。

步骤4 单击“确定”。

图 1-9 创建组织

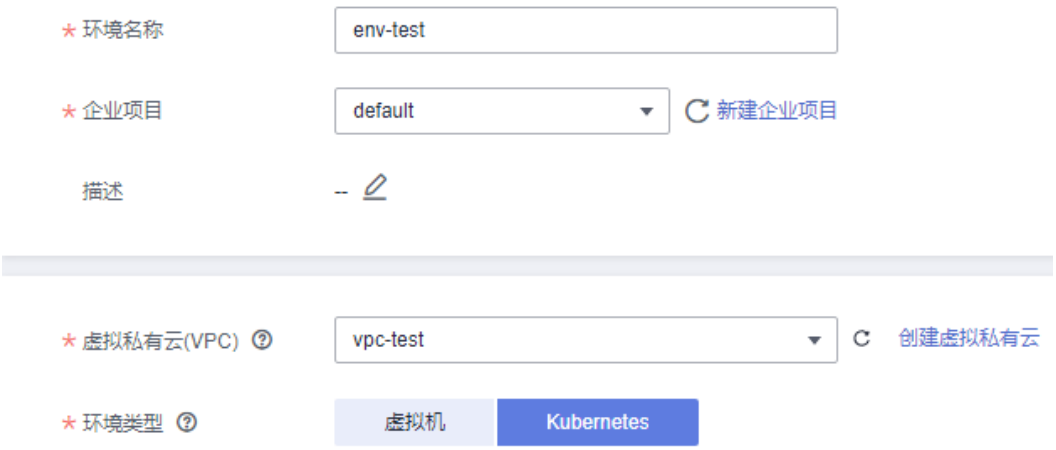
----结束

创建环境

步骤1 选择“环境管理 > 创建环境”，参照下表设置环境信息。

参数	参数说明
环境名称	输入环境名称（例如：env-test）。
企业项目	设置企业项目。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 已**开通企业项目**后可以使用。
描述	保持默认。
虚拟私有云(VPC)	选择**准备资源**中已准备好的虚拟私有云VPC。 **说明** • 环境创建完成后，不支持修改VPC。 • 选定VPC后，会加载该VPC下的资源供选择，不在该VPC下的资源无法选择。
环境类型	选择Kubernetes。

图 1-10 设置环境信息



步骤2 单击“立即创建”，进入环境详情页面。

步骤3 在“资源”下左侧列表，选择“计算”资源类型下的“云容器引擎 CCE”，单击“立即绑定”。

步骤4 在弹出的对话框中，选择**准备资源**中已创建的CCE集群资源，单击“确定”。

步骤5 在“资源”下左侧列表，选择“中间件”资源类型下的“ServiceComb引擎”，单击“纳管资源”。

步骤6 在弹出的对话框中，勾选**准备资源**中已创建的ServiceComb引擎资源，单击“确定”。

----结束

创建应用

步骤1 单击左上角 < ，返回“环境管理”页面。

步骤2 选择“应用管理 > 创建应用”，设置应用基本信息。

1. “应用名称”：填写weathermap。

   说明

   如果应用列表中存在同名应用，请参考**如何处理当前环境下已存在同名的天气预报微服务应用?** 处理。

2. “企业项目”：企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。

   已**开通企业项目**后可以使用。

步骤3 单击“确定”，完成应用创建。

图 1-11 创建应用

----结束

1.3.2 软件包部署微服务

业务场景

基于ServiceStage可以方便快捷的将微服务部署到容器（如CCE）、虚拟机（如ECS），同时支持源码部署、jar/war包部署或docker镜像包部署。同时，ServiceStage支持Java、PHP、Node.js、Python等多种编程语言应用的完全托管，包括部署、升级、回滚、启停和删除等。

本实践中使用了Java开发的后台组件和Node.js开发的前台组件。

用户故事

在本实践中，您可以通过容器部署的方式部署应用并将微服务实例注册到ServiceComb引擎中，weathermap应用需要创建并部署以下组件：

1. 前台组件：weathermapweb，基于Node.js语言开发的界面。
2. 后台组件：weather、fusionweather、forecast、edge-service，基于Java语言开发。

微服务部署有以下几个操作过程：

1. **创建并部署后台应用组件**
2. **设置edge-service组件访问方式**
3. **创建并部署前台组件**
4. **确认部署结果**
5. **添加访问方式**
6. **访问应用**

创建并部署后台应用组件

需要分别创建并部署4个应用组件：weather、forecast、fusionweather、edge-service，对应后台构建任务生成的4个软件包。

步骤1 登录ServiceStage控制台。

步骤2 单击“应用管理”，进入应用列表。

步骤3 单击**创建应用**时创建的应用名称（例如：weathermap）“操作”栏的“新增组件”。

步骤4 在“基本信息”区域，参考下表设置必填组件基本信息，其余参数保持默认。

参数	说明
组件名称	输入对应的后台组件名称（例如：weather）。
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。
所属环境	选择**创建环境**时创建的环境（例如：env-test）。
所属应用	选择**创建应用**时创建的应用（例如：weathermap）。

步骤5 在“组件包”区域，参考下表设置必填组件包参数，其余参数保持默认。

参数	说明
技术栈	组件技术栈类型选择Java。
上传方式	单击“选择软件包”，参考**表1-2**选择对应组件已上传的软件包。

步骤6 在“构建”区域，设置必填构建参数，其余参数保持默认。

参数	说明
组织	组织用于管理组件构建生成的镜像。 选择**创建组织**时创建的组织名称。
构建环境	选择“使用当前环境构建”，使用组件所属的部署环境中的CCE集群进行镜像构建。 当前环境CCE集群的master节点和node节点的CPU架构必须保持一致，否则会导致组件构建失败。

步骤7 单击“下一步”。

步骤8 “资源”区域，参考下表设置各组件“实例数”，其余参数设置保持默认。

组件名称	实例数
weather	2
forecast	1
fusionweather	1
edge-service	1

步骤9 绑定ServiceComb引擎。

说明

- 组件部署以后，微服务会注册到设置的ServiceComb引擎。
- 所有组件需要注册到同一个ServiceComb引擎，才能互相发现。

1. 选择“云服务配置 > 微服务引擎”。
2. 单击“绑定微服务引擎”。
3. 选择当前环境下已纳管的ServiceComb引擎。
4. 单击“确定”。

步骤10 （可选）选择“容器配置 > 环境变量 > 添加环境变量”，参考下表分别为weather、forecast、fusionweather组件添加环境变量。

类型	变量名称	变量/变量引用
手动添加	MOCK_ENABLED	设置为false。 • true：准备资源创建的CCE集群中的ECS节点如果没有绑定弹性公网IP或者不能访问公网时，需设置该参数值为true。则应用所用到的天气数据为模拟数据。 • false：准备资源创建的CCE集群中的ECS节点如果已绑定弹性公网IP且能访问公网时，需设置该参数值为false或者不设置该参数。则应用所用到的天气数据为实时数据。

步骤11 单击“创建并部署”。

----结束

设置 edge-service 组件访问方式

步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 单击创建应用时创建的应用名称（例如：weathermap），进入“应用概览”页。

步骤3 在“组件列表”，单击edge-service所在行“外部访问地址”列的“设置”，进入“访问方式”页面。

步骤4 单击“TCP/UDP路由配置”区域的“添加服务”，参考下表设置参数。

参数	说明
服务名称	保持默认。
访问方式	选择“公网访问”。
访问类型	选择“弹性IP”。
服务亲和	保持默认。

参数	说明
协议	选择TCP。
容器端口	填写3010。
访问端口	选择“自动生成”。

图 1-12 设置 edge-service 组件访问方式

步骤5 单击“确定”，生成访问地址。

----结束

创建并部署前台组件

步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 单击**创建应用**时创建的应用名称（例如：weathermap）“操作”栏的“新增组件”。

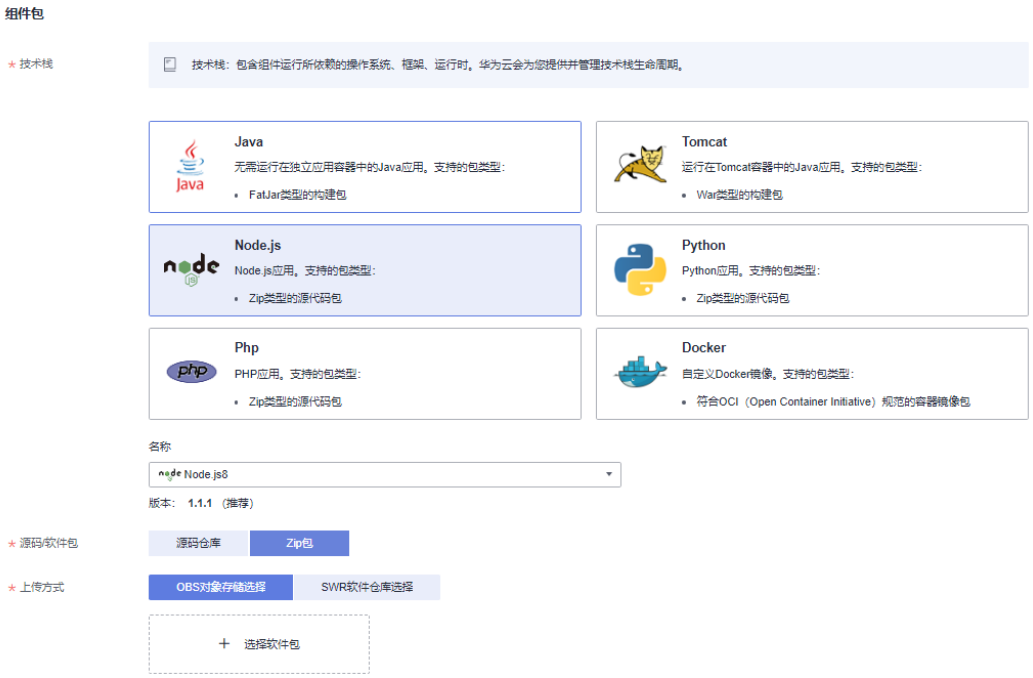
步骤3 在“基本信息”区域，参考下表设置必填组件基本信息，其余参数保持默认。

参数	说明
组件名称	输入前台组件的名称：weathermapweb。
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。
所属环境	选择**创建环境**时创建的环境（例如：env-test）。
所属应用	选择**创建应用**时创建的应用（例如：weathermap）。

步骤4 在“组件包”区域，参考下表设置必填组件包参数，其余参数保持默认。

参数	说明
技术栈	组件技术栈类型选择Node.js。
上传方式	单击“选择软件包”，参考**表1-2**选择weathermapweb组件已上传的软件包。

图 1-13 设置前台组件包参数



步骤5 在“构建”区域，参考下表设置必填构建参数，其余参数保持默。

参数	说明
组织	组织用于管理组件构建生成的镜像。 选择**创建组织**时创建的组织名称。
构建环境	选择“使用当前环境构建”，使用组件所属的部署环境中的CCE集群进行镜像构建。 当前环境CCE集群的master节点和node节点的CPU架构必须保持一致，否则会导致组件构建失败。

步骤6 单击“下一步”，添加环境变量。

1. 选择“容器配置 > 环境变量”。
2. 单击“添加环境变量”，参考下表添加环境变量。

类型	变量名称	变量/变量引用
手动添加	SERVICE_ADDR	设置edge-service组件访问方式中生成的访问地址。

步骤7 单击“创建并部署”。

----结束

确认部署结果

步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 选择“微服务引擎 > 微服务目录”。

步骤3 在微服务引擎下拉列表选择部署了微服务应用的ServiceComb引擎。

步骤4 在“微服务列表”页签的“全部应用”下拉列表中选择创建应用时创建的应用名称（例如：weathermap）。

如果各微服务实例数如下表所示，则部署成功。

组件名称	实例数
weather	2
forecast	1
fusionweather	1
edge-service	1

----结束

添加访问方式

步骤1 单击“应用管理”。

步骤2 单击创建应用时创建的应用名称（例如：weathermap），进入“应用概览”页。

步骤3 在“组件列表”，单击weathermapweb所在行“外部访问地址”列的“设置”，进入“访问方式”页面。

步骤4 单击“TCP/UDP路由配置”区域的“添加服务”，参考下表设置参数。

参数	说明
服务名称	保持默认。
访问方式	选择“公网访问”。
访问类型	选择“弹性IP”。
服务亲和	保持默认。

参数	说明
协议	选择TCP。
容器端口	填写3000。
访问端口	选择“自动生成”。

图 1-14 添加访问方式

步骤5 单击“确定”，生成访问地址。

图 1-15 访问地址

----结束

访问应用

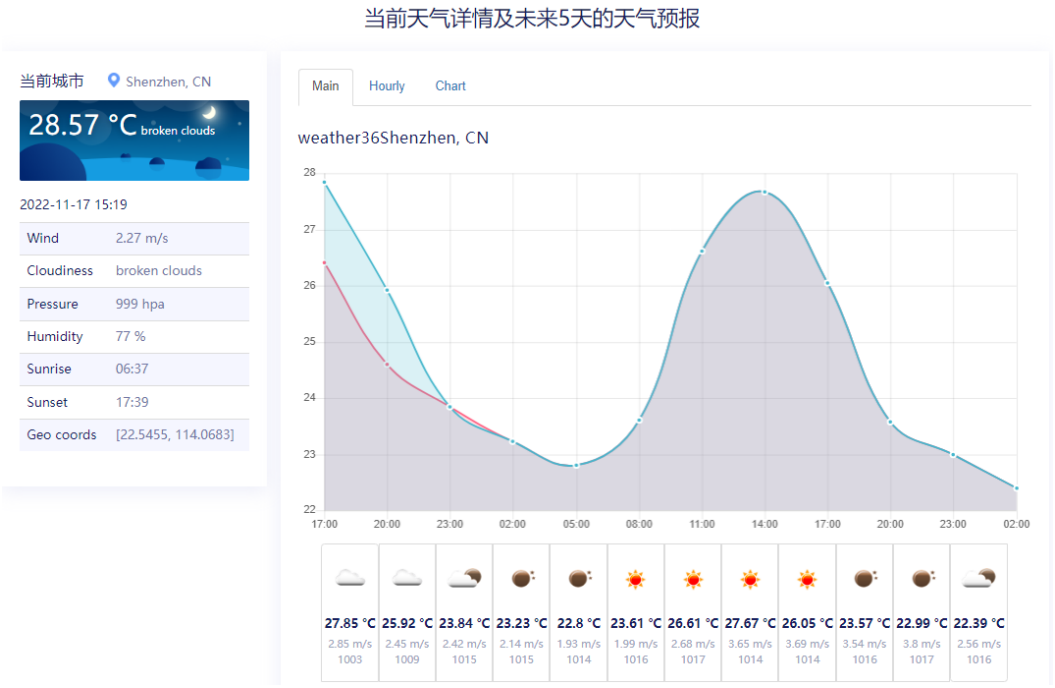
步骤1 单击左上角 < ，返回“应用管理”页面。

步骤2 单击**创建应用**时创建的应用名称（例如：weathermap），进入“应用概览”页。

步骤3 在“组件列表”，单击weathermapweb所在行“外部访问地址”列的外部访问地址。

出现以下页面表明天气预报微服务应用部署成功。

图 1-16 应用部署成功



说明

- 该数据为实时数据。
- 首次访问应用时，weather系统就绪需要一段时间。如果如上图所示页面没有出现，请持续刷新页面。

----结束

1.4 微服务日常运维

业务场景

ServiceStage支持应用监控、事件、告警、日志和调用链诊断，内置AI能力，实现轻松运维。

用户故事

在实际的使用场景中，用户可以通过图形化指标数据和阈值告警等能力，实时监控应用运行情况，同时结合性能管理与日志策略，快速定位应用的运行问题，分析性能瓶颈等。

操作步骤

步骤1 登录ServiceStage控制台。

步骤2 单击“应用管理”。

步骤3 单击应用名称（例如：weathermap），进入“应用概览”页。

步骤4 在组件列表，单击组件名称，进入组件“概览”页面。

参考**组件运维**，执行日常运维工作。

----结束

1.5 灰度发布

weather-beta是weather的新版本，提供了紫外线查询功能。升级weather-beta，需要先将少部分请求引流到新版本做功能验证，功能验证正常的情况下，再下线老版本。在升级过程中，需要保证客户的请求不能出现中断，在部署新版本的过程中不给新版本导流，在下线老版本前已经将老版本的流量全部切走。

ServiceStage提供了灰度发布功能，可以达到上述目的。

本章节演示通过使用ServiceStage的灰度发布功能部署weather服务的新版本weather-beta。

步骤1 登录ServiceStage控制台。

步骤2 单击“应用管理”。

步骤3 单击应用名称（例如：weathermap），进入“应用概览”页。

步骤4 在“组件列表”，单击weather组件名称，进入组件概览页。

步骤5 在页面右上角，单击“升级”。

步骤6 选择“灰度发布”，单击“下一步”。

步骤7 根据部署天气预报微服务时的组件部署方式，设置灰度版本配置信息。

- **源码部署方式**，参考下表设置必填参数，其余参数保持默认。

参数	说明
编译命令	选择“使用默认命令或脚本”。
Dockerfile地址	输入： `./weather-beta/`
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。

- **软件包部署方式**，参考下表设置必填参数，其余参数保持默认。

参数	说明
上传方式	1. 鼠标移动到weather-1.0.0.jar软件包名称上。 2. 单击  。 3. 选择**下载并上传天气预报组件软件包**时已上传的weather-beta-2.0.0.jar软件包。

参数	说明
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。

步骤8 参考下表设置组件灰度策略配置信息，其余参数保持默认。

参数	说明
部署架构	1. 单击“选择”。 2. 选择“类型二：注册到微服务中心（微服务B实现灰度）”。 3. 单击“确定”。
灰度策略	选择“基于流量比例”。
选择流量比例	• 灰度流量比例：设置为50%，即引入到新版本的流量比例为50%。 • 当前流量比例：自动调整为50%，即引入到当前版本的流量比例为50%。
灰度实例新增模式	选择“金丝雀（先增后减）”。
首批灰度实例数量	设置为1。
剩余实例部署批次	设置为1。

图 1-17 设置组件灰度策略配置信息

步骤9 单击“升级”。

等待组件状态由“升级/回滚中”转换为“灰度发布中”，表示已成功完成组件灰度发布。

灰度发布成功后，会给weather微服务接入的ServiceComb引擎下发“servicecomb.routeRule.weather”配置项。

您可以在“微服务引擎 > 配置管理”下查看到该配置项。

步骤10 确认灰度版本工作正常。

访问应用，多次刷新天气预报页面，可以看到界面会根据灰度策略，周期性的显示当前版本界面和灰度版本界面。

图 1-18 当前版本界面（无紫外线数据）

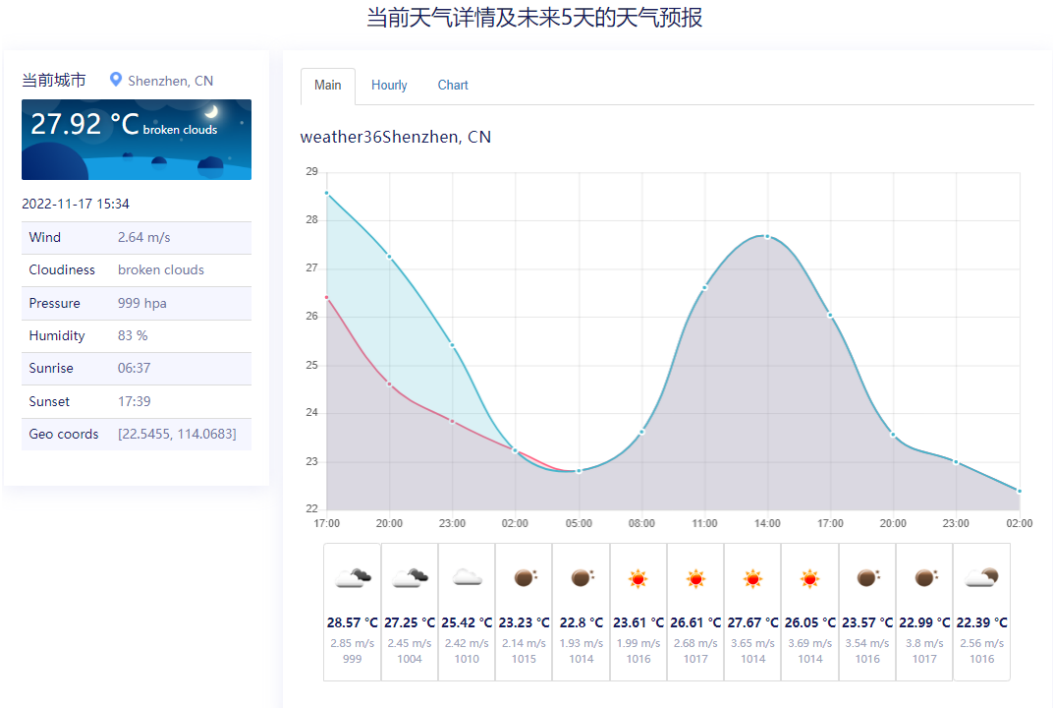
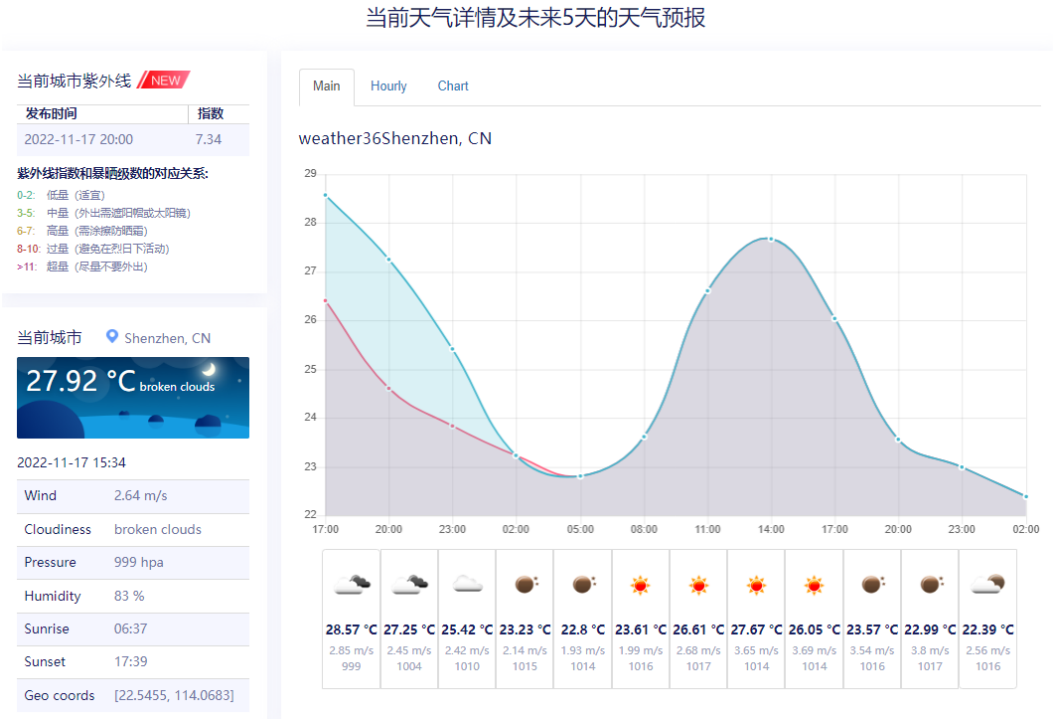


图 1-19 灰度版本界面（有紫外线数据）



----结束

1.6 微服务治理

业务场景

ServiceComb引擎提供负载均衡、降级、限流、容错、熔断、错误注入、黑白名单等治理策略。

用户故事

用户可以根据实际的业务场景提前配置相应的治理策略，灵活应对业务需求变化，保障应用的稳定运行。

降级：在本实践中，假设前台请求剧增，导致系统响应缓慢甚至可能崩溃，在这样的场景下，我们可以在fusionweather对forecast使用降级策略，对forecast进行降级处理，只请求比较重要的实时天气weather的数据，保障重要业务功能的正常运行，等流量洪峰过去再进行复原。

体验微服务降级

ServiceStage支持从界面上设置按微服务或接口粒度降级。以对forecast微服务降级为例，操作步骤如下。

步骤1 登录ServiceStage控制台。

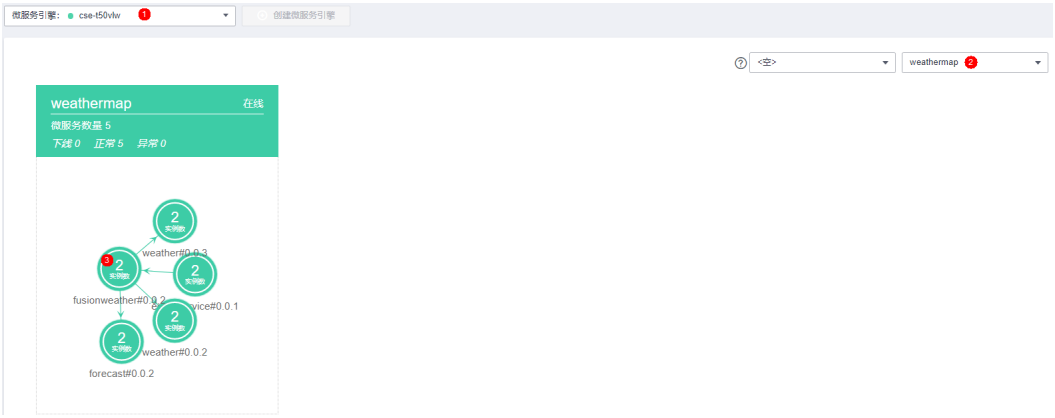
步骤2 选择“微服务引擎 > 微服务治理”。

步骤3 在微服务引擎下拉列表，选择部署了天气预报组件的ServiceComb引擎。

步骤4 在“全部应用”下拉列表选择weathermap应用名称。

步骤5 在应用卡片上双击fusionweather微服务名称，进入微服务治理页面。

图 1-20 进入微服务治理页面流程



步骤6 配置降级策略。

1. 选择“降级”。
2. 单击“新增”。
3. “降级对象”选择“forecast”。
4. “降级策略”设置为“开启”。
5. 单击“确定”。

图 1-21 配置降级策略

步骤7 查看效果。

访问应用，右侧天气预测部分显示空白。

图 1-22 微服务降级后效果

步骤8 单击，删除降级策略，以免对后续体验造成影响。

图 1-23 删除策略

----结束

1.7 常见问题

1.7.1 如何处理当前环境下已存在同名的天气预报微服务应用?

问题描述

登录ServiceStage控制台，创建指定名称的天气预报应用（例如：weathermap）时，因为应用列表中已存在同名应用，系统报“SVCSTG.00100458: 应用名已经被使用”错误提示。

解决方法

步骤1 创建应用时，“应用名称”输入唯一的应用名称，例如：weathermap_test。

步骤2 单击**步骤1**已创建的天气预报应用名称（例如：weathermap_test），进入“应用概览”页。

步骤3 单击“环境变量”，在下拉列表选择应用组件部署环境（例如：env-test）。

步骤4 单击“添加环境变量”，设置环境变量。

1. “变量名称”根据微服务组件采用的技术，参考下表设置。

微服务组件采用技术	变量名称
Java Chassis	servicecomb_service_application
Spring Cloud	spring_cloud_servicecomb_discovery_appName

2. 对应的“变量/变量引用”设置为**步骤1**已创建的应用名称，例如：weathermap_test。

步骤5 单击“提交”，完成应用环境变量的设置。

----结束

2 开启 ServiceComb 引擎专享版安全认证

概述

ServiceComb引擎专享版支持基于RBAC（Role-Based Access Control，基于角色的访问控制）策略的安全认证，并支持开启/关闭安全认证。

引擎开启了安全认证之后，要求所有连接该引擎的微服务都要配置安全认证账号和密码。否则，微服务将注册失败，导致业务受损。

适用场景

本章节介绍未开启安全认证的ServiceComb引擎专享版，开启安全认证并确保已接入引擎的微服务组件业务不受影响，即如何平滑开启安全认证。

操作步骤

步骤1 升级微服务组件使用的SDK版本。

开启安全认证功能，需要使用的SDK支持安全认证功能。如果当前的微服务组件使用的SDK版本低于要求的版本（Spring Cloud Huawei需要1.6.1及以上版本、Java Chassis需要2.3.5及以上版本），需要对当前的微服务进行SDK版本升级。

步骤2 配置微服务组件安全认证参数。

ServiceComb引擎开启安全认证前，需要对已连接到该引擎的微服务组件配置安全认证参数。配置安全认证参数是通过配置安全认证账号和密码的方式触发，具体方法如下：

- Spring Cloud微服务组件配置安全认证账号名和密码

表 2-1 Spring Cloud 微服务组件配置安全认证账号名和密码

配置文件配置方式	环境变量注入方式
为微服务的“bootstrap.yml”文件增加以下配置，若已配置请忽略。 spring: cloud: servicecomb: credentials: account: name: test #安全认证账号名，请结合用户实际值配置 password: mima #安全认证账号密码，请结合用户实际值配置 cipher: default	添加如下环境变量，请参考**手动添加应用环境变量**。 • spring_cloud_servicecomb_credentials_account_name，安全认证账号名，请结合用户实际值配置。 • spring_cloud_servicecomb_credentials_account_password，安全认证账号密码，请结合用户实际值配置。

说明

- 用户密码password默认为明文存储，无法保证安全。建议您对密码进行加密存储，请参考**自定义实现加密存储算法**。
- ServiceComb引擎没有开启安全认证时，如果连接到当前ServiceComb引擎的微服务组件配置了安全认证参数，微服务组件的正常业务功能不受影响。

- Java Chassis微服务组件配置安全认证账号名和密码

表 2-2 Java Chassis 微服务组件配置安全认证账号名和密码

配置文件配置方式	环境变量注入方式
为微服务的“microservice.yml”文件增加以下配置，若已配置请忽略。 servicecomb: credentials: rbac.enabled: true #是否开启安全认证，请结合用户实际值配置 cipher: default account: name: test #安全认证账号名，请结合用户实际配置 password: mima #安全认证账号密码，请结合用户实际配置 cipher: default	添加如下环境变量，请参考**手动添加应用环境变量**。 – servicecomb_credentials_rbac_enabled，是否开启安全认证，请结合用户实际值配置：true，开启安全认证；false，不开启安全认证。 – servicecomb_credentials_account_name，安全认证账号名，请结合用户实际值配置。 – servicecomb_credentials_account_password，安全认证账号密码，请结合用户实际值配置。

步骤3 开启ServiceComb引擎专享版安全认证，请参考**开启安全认证**。

说明

开启安全认证后，接入该引擎的微服务组件如果没有配置安全认证参数，或者微服务组件配置的安全认证账号和密码不正确，会导致该微服务组件心跳失败，服务被迫下线。

----结束

3 ServiceComb 引擎仪表盘中的数据通过 ServiceStage 对接到 AOM

背景信息

接入ServiceComb引擎的Java Chassis应用，在ServiceComb引擎仪表盘上的实时监控数据默认保留5分钟。如果需要持久化存储历史监控数据用于后续查询分析，可以使用ServiceStage的自定义指标监控功能，将微服务显示到ServiceComb引擎仪表盘中的数据对接到AOM。

本章节以软件包部署应用为例，指导您完成将ServiceComb引擎仪表盘中的数据通过ServiceStage对接到AOM。

操作步骤

步骤1 添加依赖。

在开发环境中，打开需要持久化存储历史监控数据的应用项目，在微服务pom文件中添加如下依赖：

```
<dependency>
   <groupId>org.apache.servicecomb</groupId>
   <artifactId>metrics-core</artifactId>
</dependency>
<dependency>
   <groupId>org.apache.servicecomb</groupId>
   <artifactId>metrics-prometheus</artifactId>
</dependency>
```

步骤2 将添加依赖后的应用项目重新编译打包并上传。

- 将软件包上传至CodeArts软件发布库，请参考**上传软件包**。
- 将软件包上传至SWR软件仓库，请参考**上传软件包**。
- 将软件包上传至OBS对象存储中，请参考**上传对象**。

步骤3 部署应用组件。

- 新部署组件，请执行**步骤4**。
- 已部署组件，请执行**步骤5**。

步骤4 部署**步骤2**中打包并上传的组件，请参考**使用容器部署方式基于界面配置创建并部署组件**。

1. 在组件部署过程中，选择“高级配置 > 自定义指标监控”，填写下表参数：

参数名称	参数值
上报路径	/metrics
上报端口	9696

图 3-1 设置自定义指标监控

2. 组件部署成功后，执行**步骤6**。

步骤5 对接监控指标到AOM。

1. 登录ServiceStage控制台。
2. 选择“应用管理”。
3. 单击组件所在应用名称，进入“应用概览”页。
4. 在“组件列表”，单击组件名称，进入组件“概览”页。
5. 单击“部署”。
6. 选择“单批发布”，单击“下一步”。
7. 选择“高级配置 > 自定义指标监控”，填写下表参数：

参数名称	参数值
上报路径	/metrics
上报端口	9696

图 3-2 对接监控指标

8. 单击“部署”，等待组件重新部署成功。

步骤6 在AOM中查看监控指标并导出监控数据，请参考**指标浏览**。

----结束

4 使用 ServiceStage 零代码修改实现微服务注册引擎迁移

背景信息

本章节指导您将使用Java Chassis微服务框架开发并注册在ServiceComb引擎专业版上的微服务应用组件，零代码修改迁移注册到ServiceComb引擎专享版。

须知

微服务注册引擎迁移，会存在业务中断。请在迁移前谨慎评估并选择好时间窗口。

前提条件

已创建一个未开启安全认证的ServiceComb引擎专享版，请参考**创建微服务引擎**。

引擎所在虚拟私有云和子网同部署微服务应用组件时所选择的环境中的VPC一致。

操作步骤

步骤1 登录ServiceStage控制台。

步骤2 删除已部署的微服务应用组件实例。

1. 选择“应用管理”。
2. 单击微服务应用所在的应用名称，进入“应用概览”页。
3. 在组件列表，勾选待删除组件，单击“批量删除”。
4. 在弹出的对话框，单击“确定”。

步骤3 修改部署微服务应用组件的环境。

1. 单击左上角 < ，返回“应用管理”页面。
2. 选择“环境管理”。
3. 单击部署微服务应用的环境名称。
4. 在“资源”区域左侧列表，选择“中间件”资源类型下的“ServiceComb引擎”。

5. 勾选“Cloud Service Engine”，单击“移除资源”。
6. 单击“纳管资源”。
7. 勾选已创建的ServiceComb引擎专享版，单击“确定”。

步骤4 重新部署微服务应用组件，请参考**使用容器部署方式基于界面配置创建并部署组件**。

----结束

5 使用 ServiceStage 托管 Spring Boot 应用

5.1 使用 ServiceStage 托管 Spring Boot 应用前准备

Spring Boot是一个基于Spring框架的开源应用程序开发框架，可以帮助您快速构建可独立运行的、生产级别的应用程序。

本最佳实践使用Spring官方提供的样例代码，帮助您快速在ServiceStage上快速部署、访问和升级Spring应用。

准备资源

为了方便后续的操作，需要您提前准备好如下资源：

1. 创建一个虚拟私有云VPC，请参考**创建虚拟私有云和子网**。
2. 创建一个CCE集群（如果只是试用场景，“集群管理规模”选择“50节点”，“高可用”选择“否”即可），请参考**购买集群**。
   - CCE集群所在VPC为**1**所创建的VPC。
   - 集群中至少包含1个规格为8vCPUs、16GB内存或者2个规格为4vCPUs、8GB内存的ECS节点，并且绑定弹性公网IP。为CCE集群添加节点，请参考**创建节点**。
   - CCE集群不能被其他环境绑定。
3. 已在域名提供者处注册并获取公网域名，请参考**创建公网域名**。
4. 本例基于ServiceStage绑定GitHub源码仓库，实现源码构建、归档、应用创建。需要先到**GitHub**官网注册账号，请参考**在GitHub上创建帐户**。

复刻源码

步骤1 **登录GitHub**。

步骤2 导航到源码仓库。

- 基线版本源码仓库地址：**https://github.com/spring-guides/gs-spring-boot/tree/boot-2.7**
- 灰度版本源码仓库地址：**https://github.com/herocc19/gs-spring-boot-kubernetes**

步骤3 复刻源码仓库到个人账号下，请参考**复刻仓库**。

----结束

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ServiceStage控制台。

步骤2 选择“持续交付 > 仓库授权 > 新建授权”，进入创建仓库授权页面。

步骤3 “授权名称”，保持默认。

步骤4 设置“仓库授权”。

1. 选择“GitHub”仓库。
2. “授权方式”选择“OAuth”。
3. 单击“使用OAuth授权”。
4. 阅读了解服务声明后，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。”
5. 单击“确定”。
6. 输入您的GitHub账号及密码登录GitHub完成身份认证，等待授权完成。

步骤5 单击“确认”，在仓库授权列表可以查看到已经创建完成的授权。

----结束

创建组织

步骤1 选择“部署源管理 > 组织管理”。

步骤2 单击“创建组织”，在弹出的页面中填写“组织名称”，例如：org-test。

步骤3 单击“确定”。

图 5-1 创建组织

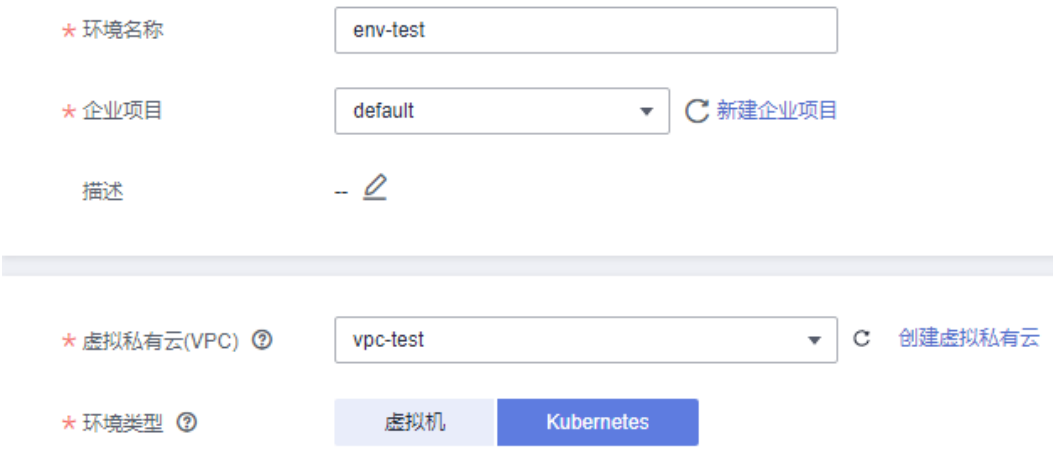
----结束

创建环境

步骤1 选择“环境管理 > 创建环境”，参照下表设置环境信息。

参数	参数说明
环境名称	输入环境名称（例如：env-test）。
企业项目	设置企业项目。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 **开通企业项目**后可以使用。
描述	保持默认。
虚拟私有云(VPC)	选择**准备资源**中已准备好的虚拟私有云VPC。 **说明** 环境创建完成后，不支持修改VPC。
环境类型	选择Kubernetes。

图 5-2 设置环境信息



步骤2 单击“立即创建”，进入环境详情页面。

步骤3 在“资源”下左侧列表，选择“计算”资源类型下的“云容器引擎 CCE”，单击“立即绑定”。

步骤4 在弹出的对话框中，选择**准备资源**中已创建的CCE集群资源，单击“确定”。

----结束

创建应用

步骤1 单击左上角<，返回“环境管理”页面。

步骤2 选择“应用管理 > 创建应用”，设置应用基本信息。

1. “应用名称”：输入应用名称（例如：springGuides）。
2. “企业项目”：企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。

   开通企业项目后可以使用。

步骤3 单击“确定”，完成应用创建。

图 5-3 创建应用

----结束

5.2 部署和访问 Spring Boot 应用

部署和访问Spring Boot应用包括以下几个操作过程：

1. **创建和部署Spring Boot应用组件**
2. **访问Spring Boot应用**

创建和部署 Spring Boot 应用组件

步骤1 登录ServiceStage控制台。

步骤2 单击“应用管理”，进入应用列表。

步骤3 单击**创建应用**时创建的应用名称（例如：springGuides）“操作”栏的“新增组件”。

步骤4 在“基本信息”区域，参考下表设置必填组件基本信息，其余参数保持默认。

参数	说明
组件名称	输入组件名称（例如：spring-boot）。
组件版本	输入：1.0.0。
所属环境	选择**创建环境**时创建的环境（例如：env-test）。
所属应用	选择**创建应用**时创建的应用（例如：springGuides）。

步骤5 在“组件包”区域，参考下表设置必填组件包参数，其余参数保持默认。

参数	说明
技术栈	组件技术栈类型选择Java。

参数	说明
源码/软件包	1. 选择“源码仓库”。 2. 选择“GitHub”。 3. “授权信息”选择**设置GitHub仓库授权**时创建的授权信息。 4. “用户名/组织”选择**复刻源码**时登录您的GitHub使用的用户名。 5. “仓库名称”选择已Fork到您的GitHub下的Spring Boot源码仓库的名称，例如：gs-spring-boot。 6. “分支”选择“boot-2.7”。

步骤6 在“构建”区域，参考下表设置必填构建参数，其余参数保持默认。

参数	说明
编译命令	1. 选择“使用自定义命令”。 2. 在命令输入框输入以下命令： `cd ./complete/;mvn clean package`
组织	选择**创建组织**时创建的组织名称。 组织用于管理组件构建生成的镜像。
构建环境	选择“使用当前环境构建”，使用组件所属的部署环境中的CCE集群进行镜像构建。 当前环境CCE集群的master节点和node节点的CPU架构必须保持一致，否则会导致组件构建失败。

图 5-4 设置构建参数

步骤7 单击“下一步”。

步骤8 在“访问方式”区域，单击  开启公网访问，参考下表设置组件公网访问参数。

参数	说明
公网访问	选择开启。
公网ELB	默认选择组件所属的部署环境中已纳管的ELB。
对外协议	保持默认。
域名	选择“绑定域名”，在输入框中输入**准备资源**时获取的公网域名。
监听端口	输入8080。

步骤9 单击“创建并部署”。

----结束

访问 Spring Boot 应用

步骤1 单击左上角<，返回“应用管理”页面。

步骤2 单击**创建应用**时创建的应用名称（例如：springGuides），进入“应用概览”页。

步骤3 在“组件列表”，单击**创建和部署Spring Boot应用组件**时设置的组件名称（例如：spring-boot）所在行“外部访问地址”列的访问地址，访问应用。

在页面显示如下信息，表示应用部署成功。

```
Greetings from Spring Boot!
```

----结束

5.3 使用 ELB 灰度发布升级组件版本

步骤1 返回ServiceStage控制台。

步骤2 单击“应用管理”，进入应用列表。

步骤3 单击**创建应用**时创建的应用名称（例如：springGuides），进入“应用概览”页面。

步骤4 在“组件列表”，单击**部署和访问Spring Boot应用**时创建的组件名称（例如：spring-boot），进入组件“概览”页面。

步骤5 在页面右上方，单击“升级”。

步骤6 “升级类型”选择“灰度发布”，单击“下一步”。

步骤7 参考下表设置灰度升级配置必填信息，其余参数保持默认。

参数	说明
软件包/镜像	固定为创建并部署组件时选择的GitHub源码仓库。 1. 单击“修改”。 2. “授权信息”选择**设置GitHub仓库授权**时创建的授权信息。 3. “用户名/组织”选择**复刻源码**时登录您的GitHub使用的用户名。 4. “仓库名称”选择已Fork到您的GitHub下的Spring Boot源码仓库的名称：gs-spring-boot-kubernetes。 5. “分支”选择“main”。
编译命令	1. 选择“使用自定义命令”。 2. 在输入框输入以下命令： `cd ./complete/;mvn clean package`
组件版本	输入：1.0.1。
部署架构	1. 单击“选择”。 2. 选择“类型三：对接ELB（服务A实现灰度）”。 3. 单击“确定”。
灰度策略	选择“基于流量比例”。
选择流量比例	• 灰度流量比例：设置为50%，即引入到新版本的流量比例为50%。 • 当前流量比例：自动调整为50%，即引入到当前版本的流量比例为50%。
灰度实例新增模式	选择“金丝雀（先增后减）”。
首批灰度实例数量	设置为1。
剩余实例部署批次	设置为1。

步骤8 单击“升级”。

等待组件状态由“升级/回滚中”转换为“灰度发布中”，表示已成功完成组件灰度发布。

步骤9 多次执行**访问Spring Boot应用**，在页面交替显示“Greetings from Spring Boot!”和“Hello”，说明组件版本ELB灰度发布成功。

----结束

6 使用 GitLab 对接 Jenkins 自动构建并升级部署到 ServiceStage 的组件

6.1 实践概述

代码开发完成后，每次上线前都需要先在Jenkins上打包成镜像包或Jar包，再将镜像包上传到SWR镜像仓库或者将Jar包上传到OBS对象存储，然后再使用ServiceStage升级组件版本配置。该流程较为繁琐，频繁发版本测试导致开发和运维效率低、用户体验差。

如果您的代码在GitLab上管理，使用ServiceStage进行应用托管并且已经部署了组件，则可以通过使用GitLab对接Jenkins自动构建打包，升级已经部署在ServiceStage上的组件版本配置。

本实践通过输出在Jenkins构建打包完成之后自动升级组件的shell脚本，实现了代码合入后自动构建打包并在ServiceStage上升级部署。

6.2 操作前准备

6.2.1 准备 Jenkins 环境

环境信息说明

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

- 虚拟机：CentOS 7.9
- Jenkins：2.319.3
- 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下载安装

  下载链接：**https://mirrors.jenkins.io/war-stable/**，参考**https://www.jenkins.io/zh/doc/book/installing/**进行安装。

- 安装git用于拉取代码进行构建命令

  ```
  yum install git –y
  ```

- JDK安装包下载

  https://www.oracle.com/cn/java/technologies/downloads/#java11

- Maven安装包下载

  https://maven.apache.org/download.cgi

- 安装Docker用于打包镜像包并上传到镜像仓库

  ```
  yum install docker
  ```

安装后检查

- 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: UTF-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
  ```

6.2.2 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

6.2.3 安装和初始化配置 obsutil 工具

obsutil工具用于上传软件包到OBS对象存储。

前提条件

1. 已获取访问密钥AK/SK，请参考**访问密钥**。
2. 已获取部署组件的ServiceStage所在区域的终端节点，请参考**地区和终端节点**。
3. 已在和部署组件的ServiceStage在同一区域的OBS中创建桶，用于存储软件包，请参考**创建桶**。

操作步骤

步骤1 登录安装了Jenkins的Linux虚拟机环境安装obsutil工具，请参考**下载和安装obsutil**。

说明

安装obsutil工具前需要在Jenkins所在Linux虚拟机中执行如下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obsutil工具软件包。
- 若执行如上命令的输出值是“aarch64”，请下载ARM 64位系统obsutil工具软件包。

步骤2 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ServiceStage所在区域的终端节点。

步骤3 检查使用obsutil上传文件到OBS是否正常。

1. 创建测试文件。
   ```
   touch test.txt
   ```
2. 使用obsutil工具上传。
   ```
   /root/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obs-mzc，将在当前目录新建的test.txt文件上传到obs-mzc桶中。提示“Upload successfully”表示上传成功。

```
[root@ecs-jenkins jar]# /root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil cp test1.txt obs://obs-mzc
Start at 2023-07-24 06:09:53.49127587 +0000 UTC
Parallel:      5                 Jobs:         5
Threshold:     50.00MB           PartSize:     auto
VerifyLength:  false             VerifyMd5:    false
CheckpointDir: /root/.obsutil_checkpoint
[-------------------------------------------------------------------------------] 100.00% 138B/s
58B/58B 622ms
Upload successfully, 58B, n/a, /root/jar/test1.txt --> obs://obs-mzc/test1.txt, cost [621], status [200], request id [000001898684BD614014A659111ABF74]
```

----结束

6.2.4 安装和初始化配置 KooCLI 工具

KooCLI工具用于调用ServiceStage服务提供的接口，对ServiceStage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Linux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
```

----结束

方式二：软件包安装

步骤1 登录Jenkins所在Linux虚拟机，执行如下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”，则为AMD 64位系统。
- 若执行如上命令的输出值是“aarch64”，则为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

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6-1**。

```
hcloud configure init
```

表 6-1 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ServiceStage服务部署区域。获取方法，请参考**地区和终端节点**。

步骤3 添加配置参数。

可能会出现找不到对应cli升级命令的问题，需要添加额外配置：

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

----结束

6.2.5 安装 Jenkins 插件并配置 Jenkins 工具

在使用GitLab对接Jenkins自动构建并部署组件到ServiceStage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全局参数配置：用于Jenkins流水线打包脚本对接git拉取代码并打包。

操作步骤

步骤1 在浏览器地址栏输入http://{安装Jenkins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6-2中的插件进行安装。

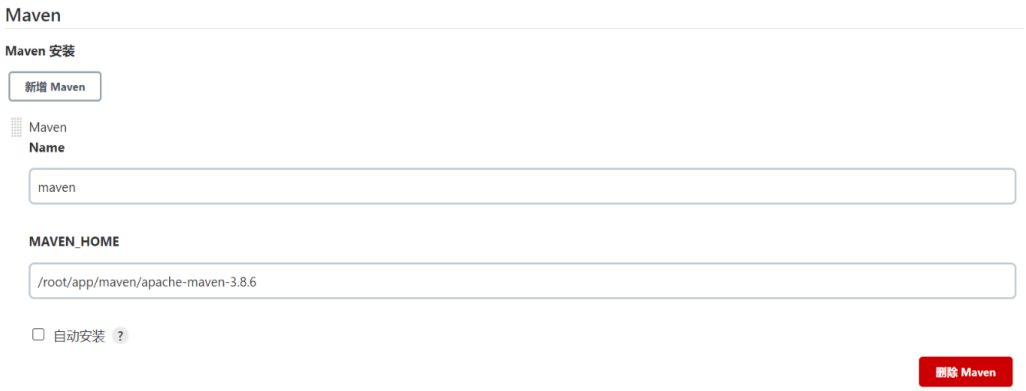
表 6-2 插件安装说明

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
Localization: Chinese (Simplified)	否	简体中文语言包

步骤4 选择“系统管理 > 全局工具配置”。

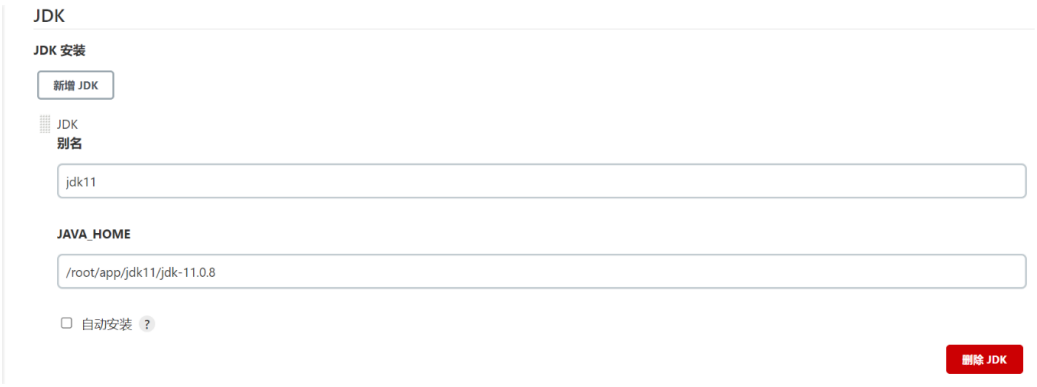
步骤5 设置Maven配置。

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。



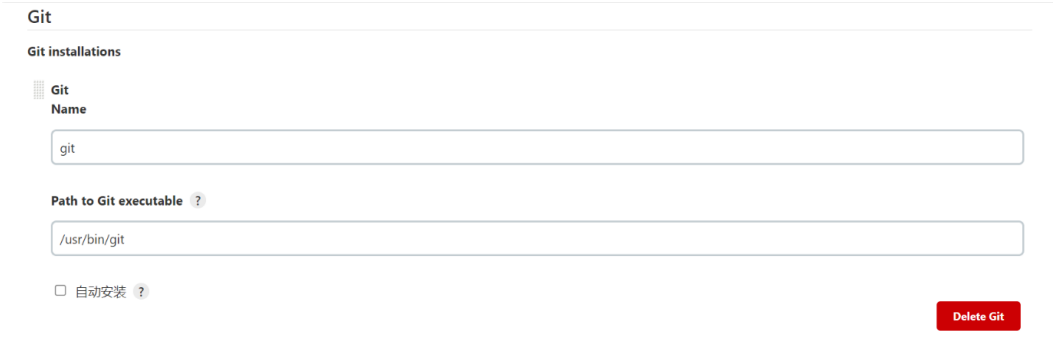
步骤6 配置JDK。

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。



步骤7 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。



----结束

6.3 操作步骤

6.3.1 对接测试

操作前需进行Jenkins对接GitLab测试，保证Jenkins通过API能够正常访问GitLab。

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”。

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。

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。

----结束

测试 Jenkins 对接 GitLab 配置

步骤1 在浏览器地址栏输入http://{安装Jenkins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

6.3.2 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ServiceStage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ServiceStage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Jenkins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Jenkins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，示例：test-upgrade，选择“流水线”，单击“确定”。

----结束

配置构建触发器

步骤1 配置Jenkins构建触发器。

1. 勾选“Build when a change is pushed to GitLab”，保存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”，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。
2. 取消勾选SSL verification的“Enable SSL verification”，单击“Add webhook”。

----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6-3**。

表 6-3 表 1 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，请参考**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，例如：/root/app/maven/apache-maven-3.8.6/bin/mvn。

参数	是否必须	参数类型	描述
upgrade_shell	是	String	upgrade.sh脚本在Jenkins所在虚拟机上存放的路径，例如：/root/jar/upgrade.sh。内容请参见**upgrade.sh脚本说明**。

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：

请使用您环境下的实际参数替换脚本中的参数变量。

```
node {
    // 定义代码仓库地址，例如：http://10.95.156.58:8090/zmg/test.git
    def git_url = '{代码仓库地址}'
    // GitLab凭据id
    def credentials_id = '{GitLab凭据id}'
    // git代码仓库分支名称，例如：main
    def branch_name = '{git代码仓库分支名称}'
    // maven安装的可执行文件路径，例如 :/root/app/maven/apache-maven-3.8.6/bin/mvn
    def maven = '{maven安装的可执行文件路径}'
    // upgrade.sh脚本存放路径，例如：/root/jar/upgrade.sh
    def upgrade_shell = '{upgrade.sh脚本存放路径}'

    stage('Clone sources') {
        git branch: branch_name, credentialsId: credentials_id, url: git_url
    }
    stage('Build') {
        // 构建jar包
        sh "'$maven' clean package -Dmaven.test.failure.ignore=true -Dmaven.wagon.http.ssl.insecure=true -
Dmaven.wagon.http.ssl.allowall=true"
    }
    stage('upgrade') {
        // 执行脚本，使用构建上传到obs的jar包升级ServiceStage组件，超时时间5分钟
        sh "timeout 300s '$upgrade_shell'"
    }
}
```

说明

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

----结束

6.3.3 upgrade.sh 脚本说明

脚本内容

请使用您环境下的实际参数替换脚本中的参数变量。

```
#!/bin/bash
# 项目id
project_id='{项目id}'
# 应用id
application_id='{应用id}'
# 组件id
component_id='{组件id}'
# 分批信息
rolling_release_batches=1
```

```
# 部署类型
deploy_type="package"


### 方法简要说明:
### 1. 查找一个字符串，如下述代码里面的key所示。如果没找到，则直接返回defaultValue。
### 2. 查找最近的冒号（:），找到后则冒号后的内容即为值的内容。
### 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);
        }
    }'
}

#查询组件信息
function getComponentInfo() {
    # 查询组件信息
    component_detials=`hcloud ServiceStage ShowComponentInfo/v3 --project_id="$project_id" --
application_id="$application_id" --component_id="$component_id"`

    # 打印组件信息
    echo "$component_detials"

    # 获取组件当前的名称
    test_name=`getJsonValuesByAwk  "$component_detials" "name" "defaultValue"`
    lenj=${#test_name}
    component_name=${test_name:1:lenj-2}
    echo "name : $component_name"


    data_time=$(date +%Y.%m%d.%H%M)
    seconds=$(date +%S)
```

```
    component_version="${data_time}${seconds:1:1}"
    echo "version: $component_version"
}

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

    # 项目打包后生成的镜像：镜像名称:版本名称
    machine_image_name='java-test:v1'
    # 上传到swr的镜像仓库路径
    swr_image_url='{镜像仓库地址}/{组织名称}/{镜像名称}:{版本}'
    # AK 用于登录swr镜像仓库
    AK='BMCKUPO9HZMI6BRDJGBD'
    #SWR登录密钥，用于登录SWR镜像仓库
    SK='{SWR登录密钥}'
    # SWR镜像仓库地址
    swr_url='{SWR镜像仓库地址}'
    #region名称
    region="{region名称}"

    echo "upload image to swr"
    docker tag "$machine_image_name" "$swr_image_url"

    login_secret=`printf "$AK" | openssl dgst -binary -sha256 -hmac "$SK" | od -An -vtx1 | sed 's/[ \n]//g' |
sed 'N;s/\n//'`

    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"`
    # 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"

   action_result=`hcloud ServiceStage ModifyComponent/v3 --project_id="$project_id" --
application_id="$application_id" --component_id="$component_id" --version="$component_version" --
runtime_stack.name="Docker" --runtime_stack.type="Docker" --source.kind="image" --
source.storage="swr" --source.url="$swr_image_url" --name="$component_name" --
deploy_strategy.rolling_release.batches=$rolling_release_batches --deploy_strategy.type="RollingRelease" `

}

# jar包部署使用场景
function obs_jar_upgrade() {

   # obsutil安装的可执行文件绝对路径
   obsutil='/root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil'
   # obs桶名
   bucket='obs://{obs桶名}'
   echo "upload jar to obs"
   # 将项目下构建生成的jar包上传到obs
   obs_result=`"$obsutil" cp ./target/*.jar "$bucket"`
   # 打印上传结果
   echo "$obs_result"
   # 上传到obs的jar包链接
   obs_jar_url='obs://{obs桶名}/{Jar包名称}'

   echo "upgrade component"

   action_result=`hcloud ServiceStage ModifyComponent/v3 --project_id="$project_id" --
application_id="$application_id" --component_id="$component_id" --version="$component_version" --
runtime_stack.name="OpenJDK8" --runtime_stack.type="Java" --source.kind="package" --
source.storage="obs" --source.url="$obs_jar_url" --name="$component_name" --
```

```
deploy_strategy.rolling_release.batches=$rolling_release_batches --deploy_strategy.type="RollingRelease" `

}

# 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" != "SUCCEEDED" ]]; do
        job_status_result=`hcloud ServiceStage ShowJobDetail/v2 --project_id="$project_id" --job_id="$id"`
        job_status=`getJsonValuesByAwk  "$job_status_result" "EXECUTION_STATUS" "defaultValue"`
        lenj=${#job_status}
        job_status=${job_status:1:lenj-2}
        echo "$job_status"
        if [[ "$job_status" != "RUNNING" && "$job_status" != "SUCCEEDED" ]]; then
            echo '部署失败'
            echo "$job_status_result"
            return
        fi
        sleep 15s
    done
    echo '部署成功'
}

function upgradeTask() {

    if [[ "$deploy_type" == "package" ]]; then
        obs_jar_upgrade
    elif [[ "$deploy_type" == "image" ]]; then
        swr_image_upgrade
    else
        return
    fi
    # 打印升级组件的结果
    echo "$action_result"
    # 获取结果中的job_id
    job_id=`getJsonValuesByAwk  "$action_result" "job_id" "defaultValue"`
    echo "$job_id"
    # 等待升级完成
    waitDeployFinish "$job_id"
}
function main() {
    getComponentInfo
    upgradeTask
}
main
```

脚本参数说明

参数	是否必须	参数类型	描述
region	是	String	Region名称。获取方法，请参考**参数值获取**。
project_id	是	String	项目ID。获取方法，请参考**参数值获取**。

参数	是否必须	参数类型	描述
application_id	是	String	应用ID。获取方法，请参考**参数值获取**。
component_id	是	String	组件ID。获取方法，请参考**参数值获取**。
rolling_release_batches	是	int	分批部署批次。
deploy_type	是	String	部署类型。 • package表示软件包部署。 • image表示镜像部署。
obsutil	否	String	当使用软件包部署如jar包部署时为必选参数，上传jar包到obs的工具安装的绝对路径。例如：/root/tools/obsutil/obsutil_linux_amd64_5.4.6/obsutil。
bucket	否	String	当使用软件包部署时为必选参数，上传到obs的桶路径，格式为obs://{桶名称}，例如：obs://obs-mzc。
obs_jar_url	否	String	当使用软件包部署时为必选参数。软件包上传obs后的链接，格式为obs://{桶名}/{软件包名}。例如，obs://obs-mzc/spring-demo-0.0.1-SNAPSHOT.jar。
machine_image_name	否	String	当使用镜像部署时为必选参数，Jenkins打包构建后生成的镜像，格式为：{镜像名称}:{版本}，例如：java-test:v1。
swr_image_url	否	String	当使用镜像部署时为必选参数，上传到SWR镜像仓库的镜像包路径，格式为：{镜像仓库地址}/{组织名称}/{镜像包名称}:{版本}，其中SWR镜像仓库地址格式为：swr.{区域所属项目名称}.myhuaweicloud.com。
AK	否	String	当使用镜像部署时为必选参数。访问密钥ID，即AK，用于登录SWR镜像仓库。获取方法，请参考**访问密钥**。
SK	否	String	当使用镜像部署时为必选参数。与访问密钥ID（AK）结合使用的密钥，即SK，用于登录SWR镜像仓库。获取方法，请参考**访问密钥**。
login_secret	否	String	当使用镜像部署时为必选参数。SWR镜像仓库的登录密钥，用于登录SWR镜像仓库。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//' {AK}、{SK}请替换为已获取到的AK、SK的值。
swr_url	否	String	当使用镜像部署时为必选参数。SWR镜像仓库地址，格式为：swr.{区域所属项目名称}.myhuaweicloud.com

参数值获取

- 获取region、project_id
  a. 登录ServiceStage控制台。
  b. 鼠标移动到右上角登录用户名上，在下拉菜单选择“我的凭证”。
  c. 查看所属区域的项目和项目ID，即为对应的region和project_id值。
- 获取application_id、component_id
  a. 登录ServiceStage控制台。
  b. 单击“组件管理”。
  c. 单击对应的组件名称。
  d. 在“概览”界面的“配置详情”区域，单击“组件配置”。
     查看CAS_APP_ID、CAS_APPLICATION_ID的值，即为application_id、component_id。

6.4 构建验证

6.4.1 手动构建验证

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

步骤2 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。

步骤4 单击“立即构建”，生成构建任务。

在“构建历史”以及“阶段视图”中会有对应的构建任务信息。鼠标悬浮在对应步骤上，会展示任务状态以及日志按钮。单击“log”查看日志。

步骤5 登录ServiceStage控制台。

步骤6 单击“组件管理”。

步骤7 在组件列表中单击升级的组件名称，进入组件“概览”界面。

在“概览”界面，查看“组件版本”以及组件包“代码源”是否已经更新。

步骤8 单击“部署记录”，查看对应的部署记录。

----结束

6.4.2 GitLab 自动触发 Jenkins 构建

GitLab触发Jenkins构建，有以下两种方式：

- 方式一：通过配置好的Webhook来Push events，触发Jenkins构建任务。
- 方式二：修改构建配置指定分支的文件来Push events，触发Jenkins构建任务。

本章节通过方式一为例，来触发Jenkins构建。

操作步骤

步骤1 登录GitLab，进入代码仓库。

步骤2 单击“Settings”，选择“Webhooks”，在右下角的“Test”下拉框，选择“Push events”。

步骤3 在浏览器地址栏输入http://{安装Jenkins的Linux虚拟机IP}:8080，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。

步骤4 单击构建任务编号，选择“Console Output”，查看构建输出日志。

步骤5 登录ServiceStage控制台。

步骤6 单击“组件管理”。

步骤7 在组件列表中单击升级的组件名称，进入组件“概览”页面。

在“概览”界面，查看“组件版本”以及组件包“代码源”是否已经更新。

步骤8 单击“部署记录”，查看对应的部署记录。

----结束

7 使用 ServiceStage 全链路流量控制实现 Spring Cloud 应用全链路灰度

7.1 全链路流量控制概述

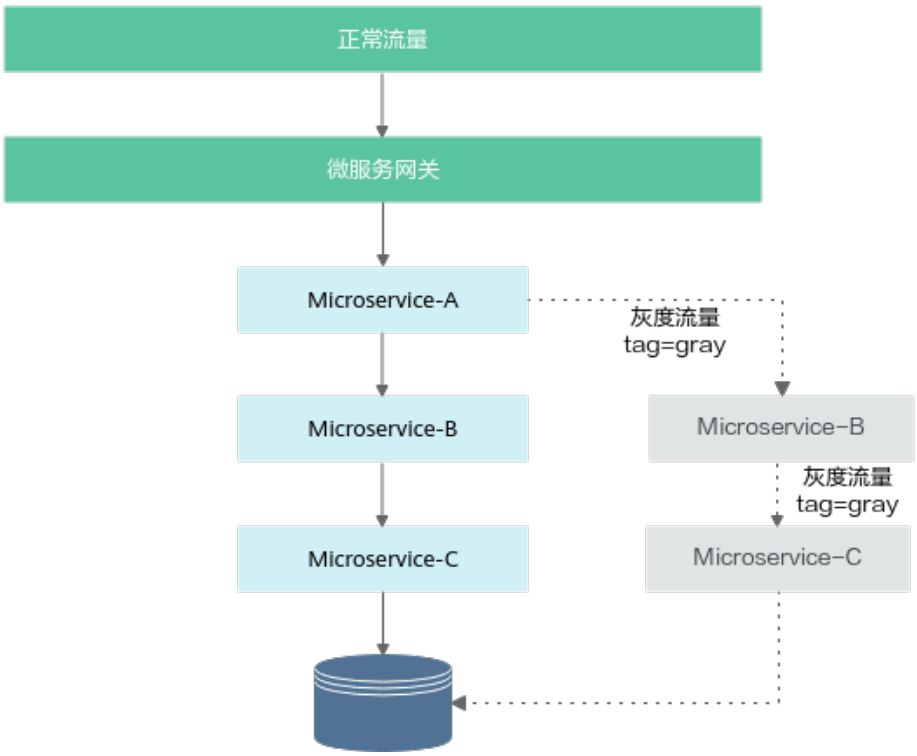
说明

仅“华东-上海1”、“亚太-新加坡”区域支持全链路流量控制功能。

背景信息

在一个存在多个微服务组件的应用体系中，对其中一个微服务组件进行版本升级发布时，会涉及到其他微服务组件版本的变动升级。所以要求在灰度验证时能够使得灰度流量同时经过涉及到的所有微服务组件灰度版本。如图7-1所示，由于Microservice-B组件发布新版本涉及到Microservice-C组件的变动，所以在做灰度验证时要求流量能够同时经过Microservice-B、Microservice-C组件的灰度版本。因此，需要额外配置涉及到的Microservice-C组件微服务治理规则，确保流入灰度环境下Microservice-B组件流量能够转发到Microservice-C组件的灰度版本。

图 7-1 全链路流量控制示意图

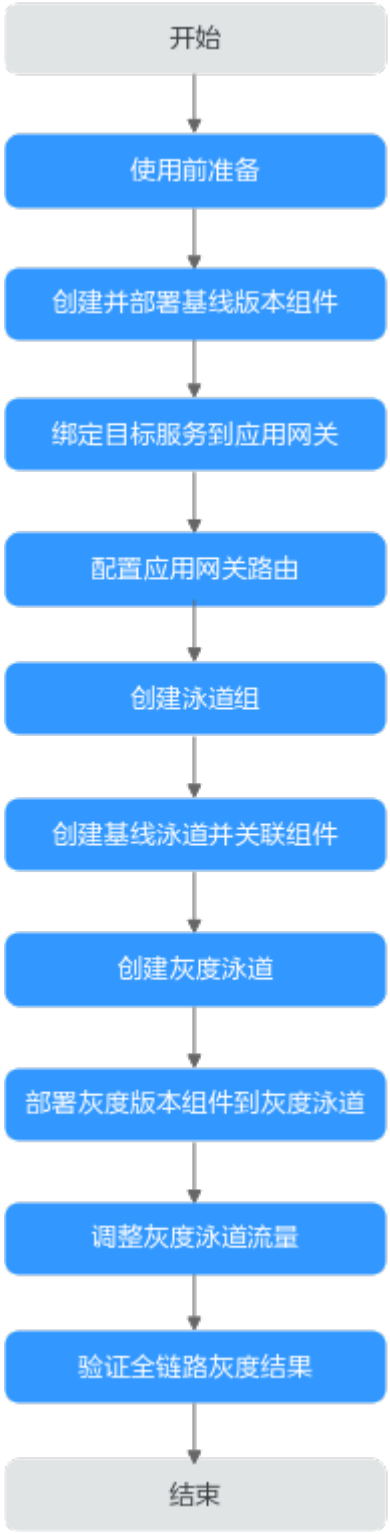


但是在真实业务场景中，由于业务的微服务组件规模和数量很大，一条请求链路可能会经过很多个微服务组件。新功能发布时也可能会涉及到多个微服务组件同时变更，并且业务的微服务组件之间依赖错综复杂。频繁的组件版本发布、以及组件多版本并行开发导致流量治理规则日益膨胀，给整个系统的可维护性和稳定性带来了严重挑战。

对于以上问题和挑战，业界结合实际业务场景和产业实践经验，提出了一种端到端的灰度发布方案，来进行全链路流量控制，即全链路灰度。全链路流量控制功能将应用组件的相关版本隔离成一个独立的运行环境（即泳道），通过设置流控路由规则，将满足规则的请求流量路由到目标版本应用。

使用流程

图 7-2 全链路流量控制使用流程



全链路流量控制的使用流程如图7-2所示。

1. 使用前准备

   使用ServiceStage全链路流量控制实现Spring Cloud应用全链路灰度前，需要先准备VPC、CCE、ServiceComb引擎专享版、微服务应用网关等资源，并完成Fork全链路流量控制示例源码到您的GitHub代码库、创建组件部署环境等准备工作，请参考**使用ServiceStage全链路流量控制实现Spring Cloud应用全链路灰度前准备**。

2. 创建并部署基线版本组件

   创建并部署基线版本组件，用于关联到后续操作创建的基线泳道，请参考**创建并部署基线版本组件**。

3. 绑定目标服务到应用网关

   为应用网关绑定目标服务，以便网关获取组件微服务地址，请参考**绑定目标服务到应用网关**。

4. 配置应用网关路由

   为绑定了目标服务的应用网关配置路由规则，当应用网关收到访问流量时，会根据已配置的路由规则判断是否匹配并做相应的处理，请参考**配置应用网关路由**。

5. 创建泳道组

   泳道组是一组泳道的集合，用于区分不同的组织或场景，请参考**创建泳道组**。

6. 创建基线泳道并关联组件

   泳道用于为相同版本组件定义一套隔离环境。只有满足了流控路由规则的请求流量才会路由到对应泳道里的打标签组件。基线泳道包括应用中所有组件的基线版本。当微服务调用链中不存在某个组件的时候，会默认访问基线泳道中的组件，请参考**创建基线泳道并关联组件**。

7. 创建灰度泳道

   灰度泳道用于关联应用的灰度版本组件，请参考**创建灰度泳道**。

8. 部署灰度版本组件到灰度泳道

   灰度泳道创建后，根据实际业务需要创建灰度版本组件，用于调整流量至灰度泳道验证业务，请参考**部署灰度版本组件到灰度泳道**。

9. 调整灰度泳道流量

   根据实际业务需要修改路由配置，调整流量至灰度泳道，请参考**调整灰度泳道流量**。

10. 验证全链路灰度结果

    访问组件实例，验证全链路灰度结果，请参考**验证全链路灰度结果**。

使用限制

- 全链路流量控制依赖**应用网关**提供微服务访问流量入口，用于通过标签控制流量转发到对应泳道的目标服务组件。
- 使用Kubernetes类型环境部署且绑定了ServiceComb引擎专享版的微服务组件，支持使用全链路流量控制。关于创建并部署组件，请参考**创建并部署组件**。
- 微服务组件依赖的开发框架为spring-cloud-huawei 1.11.5-2021.0.x及以上版本。
- 加入全链路流量控制的组件，将不再支持通过灰度发布方式升级组件。关于灰度发布升级组件，请参考**灰度发布（金丝雀）方式升级**。

7.2 使用 ServiceStage 全链路流量控制实现 Spring Cloud 应用全链路灰度前准备

准备资源

1. 创建一个虚拟私有网络VPC，请参考**创建虚拟私有云和子网**。
2. 创建一个1.15以上版本的CCE集群（如果只是试用场景，“集群管理规模”选择“50节点”，“高可用”选择“否”即可），请参考**购买集群**。
   - 集群中至少包含1个规格为8vCPUs、16GB内存或者2个规格为4vCPUs、8GB内存的ECS节点，并且绑定弹性公网IP。
   - 集群所在VPC为**1**创建的VPC。
   - CCE集群不能被其他环境绑定。
3. 创建不开启安全认证的2.4.0及以上版本的ServiceComb引擎专享版，请参考**创建微服务引擎**。

   如果ServiceComb引擎所在VPC与**1**创建的VPC不一致，需正确配置VPC连通。
4. 创建微服务应用网关，请参考**创建应用网关**。

   如果微服务应用网关与环境所在VPC不一致，需正确配置VPC连通。
5. 本例基于ServiceStage绑定GitHub源码仓库，实现源码构建、归档、应用创建，需要先到**GitHub**官网注册账号。

Fork 全链路流量控制示例源码

使用您的账号登录GitHub，并Fork全链路流量控制示例源码。源码地址：**https://github.com/servicestage-demo/full-link-router-sch-demo**。

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ServiceStage控制台。

步骤2 选择“持续交付 > 仓库授权 > 新建授权”，参考下表配置授权信息。参数前面带*号的是必须设置的参数。

参数	说明
*授权名称	授权名称保持默认，创建之后不可更改。
*仓库授权	1. 选择GitHub代码仓库。 2. “授权方式”选择“OAuth”。 3. 单击“使用OAuth授权”，根据页面提示完成访问GitHub源码仓库授权。

----结束

创建组织

步骤1 选择“部署源管理 > 组织管理”。

步骤2 单击“创建组织”，在弹出的页面中填写“组织名称”，例如：org-test。

步骤3 单击“确定”。

图 7-3 创建组织

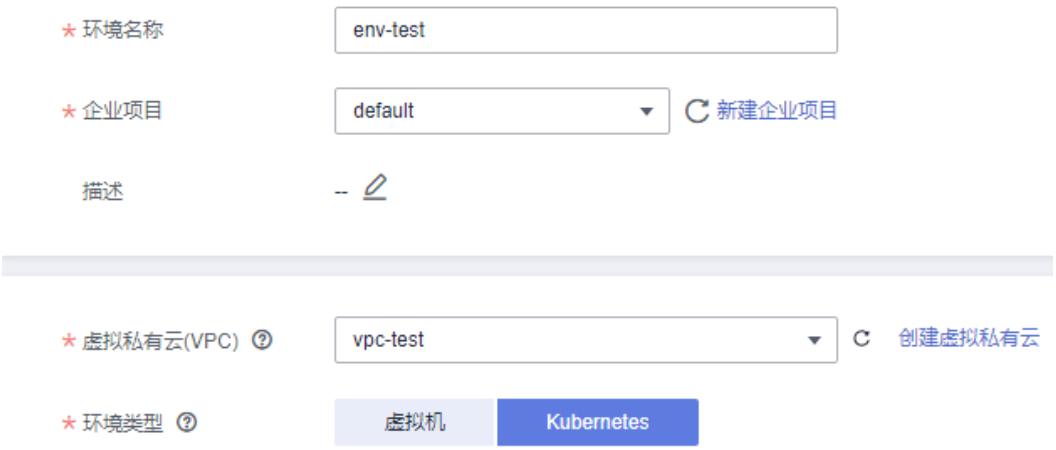
----结束

创建环境

步骤1 选择“环境管理 > 创建环境”，参照下表设置环境信息。

参数	参数说明
环境名称	输入环境名称（例如：env-test）。
企业项目	设置企业项目。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 已**开通企业项目**后可以使用。
描述	保持默认。
虚拟私有云(VPC)	选择**准备资源**中已准备好的虚拟私有云VPC。 **说明** 环境创建完成后，不支持修改VPC。
环境类型	选择Kubernetes。

图 7-4 设置环境信息



步骤2 单击“立即创建”，进入环境详情页面。

步骤3 在“资源”下左侧列表，选择“计算”资源类型下的“云容器引擎 CCE”，单击“立即绑定”。

步骤4 在弹出的对话框中，选择**准备资源**中已创建的CCE集群资源，单击“确定”。

步骤5 在“资源”下左侧列表，选择“中间件”资源类型下的“ServiceComb引擎”，单击“纳管资源”。

步骤6 在弹出的对话框中，勾选**准备资源**中已创建的ServiceComb引擎资源，单击“确定”。

----结束

创建应用

步骤1 单击左上角 < ，返回“环境管理”页面。

步骤2 选择“应用管理 > 创建应用”，参考下表设置应用信息。

参数名称	参数说明
应用名称	输入“应用名称”，例如：spring-lane。
企业项目	设置企业项目。 企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。 已**开通企业项目**后可以使用。

步骤3 单击“确定”，完成应用创建。

----结束

7.3 创建并部署基线版本组件

创建并部署基线版本组件，用于关联到后续操作创建的基线泳道。此处需要分别创建并部署3个基线版本组件：unit-controller、unit-consumer和unit-provider。

部署组件

步骤1 登录ServiceStage控制台。

步骤2 单击“应用管理”，进入应用列表。

步骤3 单击**创建应用**时创建的应用名称（例如：spring-lane）“操作”栏的“新增组件”。

步骤4 在“基本信息”区域，参考下表设置必填组件基本信息，其余参数保持默认。

参数名称	参数说明
组件名称	输入组件的名称，例如：unit-controller。
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。
所属环境	选择**创建环境**时创建的环境（例如env-test）。
所属应用	选择**创建应用**时创建的应用（例如：spring-lane）。

步骤5 在“组件包”区域，参考下表设置必填组件包参数，其余参数保持默认。

参数	说明
技术栈	组件技术栈类型选择Java。
源码/软件包	1. 选择“源码仓库”。 2. 选择“GitHub”。 3. “授权信息”选择**设置GitHub仓库授权**时创建的授权信息。 4. “用户名/组织”选择**Fork全链路流量控制示例源码**时登录您的GitHub使用的用户名。 5. “仓库名称”选择已Fork到您的GitHub下的全链路流量控制示例源码仓库的名称，例如：full-link-router-sch-demo。 6. “分支”选择“main”。

步骤6 在“构建”区域，设置必填构建参数。

1. “编译命令”：选择“使用自定义命令”，参考下表分别为unit-controller、unit-consumer和unit-provider组件设置自定义编译命令。

组件名称	自定义编译命令
unit-controller	cd ./unit-controller/;mvn clean package

组件名称	自定义编译命令
unit-consumer	cd ./unit-consumer/;mvn clean package
unit-provider	cd ./unit-provider/;mvn clean package

2. “组织”：选择**创建组织**时创建的组织名称。
3. “构建环境”：选择“使用当前环境构建”。
4. 其余参数，保持默认。

步骤7 单击“下一步”。

步骤8 在“资源”区域，参考下表设置各组件“实例数”，其余参数设置保持默认。

组件名称	实例数
unit-controller	1
unit-consumer	1
unit-provider	1

步骤9 绑定ServiceComb引擎。

说明

- 组件部署以后，微服务会注册到设置的ServiceComb引擎。
- 所有组件需要注册到同一个ServiceComb引擎，才能互相发现。

1. 选择“云服务配置 > 微服务引擎”。
2. 单击“绑定微服务引擎”。
3. 选择当前环境下已纳管的ServiceComb引擎。
4. 单击“确定”。

步骤10 单击“创建并部署”。

等待组件部署完成。

----结束

确认部署结果

步骤1 单击左上角‹，返回“应用管理”页面。

步骤2 选择“微服务引擎 > 微服务目录”。

步骤3 在微服务引擎下拉列表选择部署了微服务应用的ServiceComb引擎。

步骤4 在“微服务列表”页签，如果各微服务实例数如下表所示，则部署成功。

组件名称	实例数
unit-controller	1
unit-consumer	1

组件名称	实例数
unit-provider	1

----结束

7.4 绑定目标服务到应用网关

为应用网关绑定目标服务，以便网关获取组件微服务地址。

操作步骤

步骤1 为**准备资源**时创建的应用网关创建服务来源，请参考**创建服务来源**。

服务来源参数请参考下表进行设置。

参数名称	参数说明
来源类型	目标服务的来源，选择“CSE ServiceComb引擎”。
来源名称	输入目标服务的名称，例如：servicecomb。
引擎实例	选择**准备资源**时已经创建的ServiceComb引擎专享版。

步骤2 为**准备资源**时创建的应用网关绑定目标服务，请参考**创建服务**。

参考下表填写相关参数。

参数名称	参数说明
来源类型	目标服务的来源，选择“CSE ServiceComb引擎”。
服务来源	选择**步骤1**输入的目标服务的名称，例如：servicecomb。
环境选择	保持默认。
服务列表	选择接入了**步骤1**所选择的ServiceComb引擎的目标服务实例“unit-controller”。

----结束

7.5 配置应用网关路由

为应用网关配置路由规则，当应用网关收到访问流量时，会根据已配置的路由规则判断是否匹配并做相应的处理。

操作步骤

步骤1 **登录微服务引擎控制台**。

步骤2 为**准备资源**时创建的应用网关配置路由，请参考**创建路由**。

参考下表填写相关参数，其余参数保持默认。

参数名称	参数说明
路由名称	输入路由名称，例如：controller。
域名	勾选下拉列表中的“*”。
路由规则	1. 服务地址：选择“前缀匹配”，输入“/”。 2. 请求方法：选择“GET”。 3. 服务名称：选择“unit-controller”。 4. 分组：设置为“不限制”。 5. 权重：设置为100。

须知

使用全链路流量控制实现灰度发布时，一条路由下的所有路由规则必须绑定同一个目标服务。如需绑定多个目标服务，请创建多条路由。

----结束

7.6 创建泳道组

泳道组是一组泳道的集合，用于区分不同的组织或场景。

操作步骤

步骤1 登录ServiceStage控制台。

步骤2 选择“全链路流量控制 > 创建泳道组”，参考下表填写泳道组必填信息，其余参数保持默认。

参数名称	参数说明
泳道组名称	输入泳道组的名称，例如：lane-test。
流量入口网关	选择**绑定目标服务到应用网关**时选择的应用网关，用于转发打标签服务流量。
*目标服务	选择**绑定目标服务到应用网关**时为应用网关绑定的目标服务名称“unit-controller”，通过该网关转发服务流量。

步骤3 单击“确定”，完成泳道组创建。

----结束

7.7 创建基线泳道并关联组件

泳道用于为相同版本组件定义一套隔离环境。只有满足了流控路由规则的请求流量才会路由到对应泳道里的打标签组件。基线泳道包括应用中所有组件的基线版本。当微服务调用链中不存在某个组件的时候，会默认访问基线泳道中的组件。

操作步骤

步骤1 登录ServiceStage控制台。

步骤2 单击“全链路流量控制”。

步骤3 单击待创建基线泳道所在泳道组名称（例如：lane-test），进入“全链路流量控制”页面。

步骤4 单击“创建基线泳道”，参考下表填写泳道信息。

参数名称	参数说明
泳道名称	输入泳道的名称，例如：base。
标签	用于在Kubernetes类型的环境下创建并部署组件时，将绑定ServiceComb引擎专享版的组件打上相应的标签以标记流量。当有请求访问时，应用网关会根据路由规则将流量转发到对应流量标签的微服务上。当无法找到对应标签的微服务时，将转发至基线泳道对应的微服务。 基线泳道的标签默认为base，不可修改。

步骤5 单击“确定”，完成基线泳道创建。

步骤6 单击已创建好的基线泳道卡片上的“关联组件”或右上角的。

步骤7 在弹出的“关联组件”对话框勾选**创建并部署基线版本组件**时已部署好的全部待关联组件：unit-controller、unit-consumer和unit-provider。

步骤8 单击“确定”，完成基线泳道组件关联。

单击基线泳道卡片上当前已关联的组件数，可以查看基线泳道已关联的组件列表。

----结束

7.8 创建灰度泳道

基线泳道创建完成后，需要再创建灰度泳道用于部署组件灰度版本，用于调整流量至灰度泳道验证业务。

操作步骤

步骤1 登录ServiceStage控制台。

步骤2 单击“全链路流量控制”。

步骤3 单击待创建灰度泳道所在泳道组名称（例如：lane-test），进入“全链路流量控制”页面。

步骤4 单击“创建泳道”，参考下表填写灰度泳道信息。

参数名称	参数说明
*泳道名称	输入灰度泳道的名称，例如：gray。
*标签	输入gray。 用于在Kubernetes类型的环境下创建并部署组件时，将绑定ServiceComb引擎专享版的组件打上相应的标签以标记流量。当有请求访问时，应用网关会根据路由规则将流量转发到对应流量标签的微服务上。

步骤5 单击“确定”，完成灰度泳道创建。

----结束

7.9 部署灰度版本组件到灰度泳道

灰度泳道创建后，根据实际业务需要创建灰度版本组件，用于调整流量至灰度泳道验证业务。

此处需要分别创建并部署2个灰度版本组件：unit-controller-gray和unit-provider-gray。

操作步骤

步骤1 登录ServiceStage控制台。

步骤2 单击“全链路流量控制”。

步骤3 单击待操作灰度泳道所在泳道组名称（例如：lane-test），进入“全链路流量控制”页面。

步骤4 单击待操作非基线泳道（例如：gray）上的  。

步骤5 在“基本信息”区域，参考下表设置必填组件基本信息，其余参数保持默认。

参数名称	参数说明
组件名称	输入组件的名称，例如：unit-controller-gray。
组件版本	单击“自动生成”，默认以您单击“自动生成”时的时间来生成版本号。格式为yyyy.mmdd.hhmms，s取时间戳中秒数的个位值。例如：时间戳为2022.0803.104321，则版本号为2022.0803.10431。
所属环境	选择**创建环境**时创建的环境（例如env-test）。
所属应用	选择**创建应用**时创建的应用（例如：spring-lane）。

步骤6 在“组件包”区域，参考下表设置必填组件包参数，其余参数保持默认。

参数	说明
技术栈	组件技术栈类型选择Java。
源码/软件包	1. 选择“源码仓库”。 2. 选择“GitHub”。 3. “授权信息”选择**设置GitHub仓库授权**时创建的授权信息。 4. “用户名/组织”选择**Fork全链路流量控制示例源码**时登录您的GitHub使用的用户名。 5. “仓库名称”选择已Fork到您的GitHub下的全链路流量控制示例源码仓库的名称，例如：full-link-router-sch-demo。 6. “分支”选择“main”。

步骤7 在“构建”区域，设置必填构建参数。

1. “编译命令”：选择“使用自定义命令”，参考下表分别为unit-controller-gray和unit-provider-gray组件设置自定义编译命令。

组件名称	自定义编译命令
unit-controller-gray	cd ./unit-controller/;mvn clean package
unit-provider-gray	cd ./unit-provider/;mvn clean package

2. “组织”：选择**创建组织**时创建的组织名称。
3. “构建环境”：选择“使用当前环境构建”。
4. 其余参数，保持默认。

步骤8 单击“下一步”。

步骤9 在“资源”区域，参考下表设置各组件“实例数”，其余参数设置保持默认。

组件名称	实例数
unit-controller-gray	1
unit-provider-gray	1

步骤10 绑定ServiceComb引擎。

说明

- 组件部署以后，微服务会注册到设置的ServiceComb引擎。
- 所有组件需要注册到同一个ServiceComb引擎，才能互相发现。

1. 选择“云服务配置 > 微服务引擎”。
2. 单击“绑定微服务引擎”。
3. 选择当前环境下已纳管的ServiceComb引擎。
4. 单击“确定”。

步骤11 单击“创建并部署”。

等待unit-controller-gray、unit-provider-gray组件全部部署完成。

----结束

7.10 调整灰度泳道流量

根据实际业务需要修改路由配置，调整流量至灰度泳道。

操作步骤

步骤1 登录ServiceStage控制台。

步骤2 单击“全链路流量控制”。

步骤3 单击待操作灰度泳道所在泳道组名称（例如：lane-test），进入“全链路流量控制”页面。

步骤4 单击泳道组当前关联的流量入口网关卡片上的“网关路由配置”。

步骤5 “配置方式”选择“基于内容配置”。

步骤6 单击以对应灰度泳道名称命名的页签（例如：gray），进入为指定灰度泳道配置网关路由页面。

步骤7 单击“新增匹配规则”，参考下表设置路由匹配规则。

参数名称	参数说明
匹配类型	支持的路由规则匹配类型。 当前仅支持基于“请求头”类型的匹配。
参数名称	“匹配类型”对应的key值，设置为：type。
条件类型	“条件值”满足的匹配规则，选择“前缀匹配”。
条件值	“匹配类型”对应的value值，设置为：gray。

步骤8 单击“确定”，完成基于内容的网关路由配置。

步骤9 开启泳道流量，使配置生效。

1. 单击  。
2. 单击“确定”。

配置会对该灰度泳道下的所有组件生效。如果业务请求无法匹配到泳道所配置的路由规则，则网关将业务请求转发至基线泳道的组件处理。

----结束

7.11 验证全链路灰度结果

访问灰度泳道实例

打开cmd命令，执行以下命令访问灰度泳道unit-controller组件提供的服务：


```
curl -H "type:gray" http://${网关访问地址}/unit-controller/hello
```

网关访问地址取值，请参考**查看应用网关信息**。

执行结果示例如下：

```
{
    "unit-consumer": {
        "SERVICECOMB_INSTANCE_PROPS": "cas_lane_tag:base,affinity-tag:base","ip":"x.x.x.x"
    },
    "unit-provider": {
        "SERVICECOMB_INSTANCE_PROPS": "cas_lane_tag:gray,affinity-tag:gray","ip":"x.x.x.x"
    },
    "unit-controller": {
        "SERVICECOMB_INSTANCE_PROPS": "cas_lane_tag:gray,affinity-tag:gray","ip":"x.x.x.x"
    }
}
```

全部流量路由方向为从灰度泳道（gray）的unit-controller实例流向基线泳道（base）的unit-consumer实例，再流向灰度泳道（gray）的unit-provider实例。

访问基线泳道实例

打开cmd命令，执行以下命令访问基线泳道unit-controller组件提供的服务：

```
curl http://${网关访问地址}/unit-controller/hello
```

网关访问地址取值，请参考**查看应用网关信息**。

执行结果示例如下：

```
{
    "unit-consumer": {
        "SERVICECOMB_INSTANCE_PROPS": "cas_lane_tag:base,affinity-tag:base","ip":"x.x.x.x"
    },
    "unit-provider": {
        "SERVICECOMB_INSTANCE_PROPS": "cas_lane_tag:base,affinity-tag:base","ip":"x.x.x.x"
    },
    "unit-controller": {
        "SERVICECOMB_INSTANCE_PROPS": "cas_lane_tag:base,affinity-tag:base","ip":"x.x.x.x"
    }
}
```

全部流量路由方向为从基线泳道（base）的unit-controller实例流向unit-consumer实例，再流向unit-provider实例。