ALIBABA CLOUD

# Alibaba Cloud

## API 网关

## 最佳实践

文档版本：20220513

阿里云

# 法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

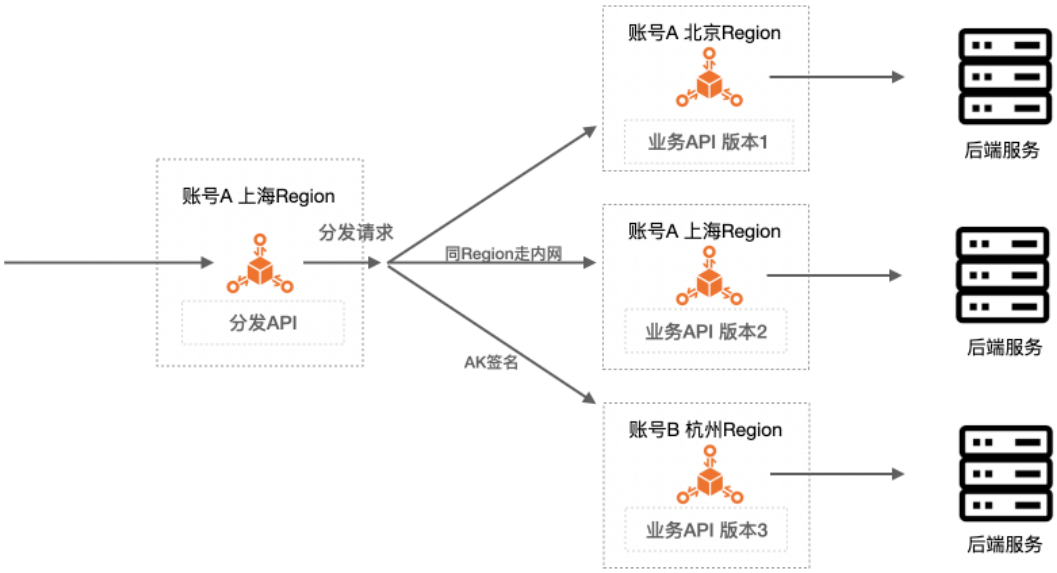
# 通用约定

| 格式 | 说明 | 样例 |
| --- | --- | --- |
| 危险 | 该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 危险<br>重置操作将丢失用户配置数据。 |
| 警告 | 该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。 | 警告<br>重启操作将导致业务中断，恢复业务时间约十分钟。 |
| 注意 | 用于警示信息、补充说明等，是用户必须了解的内容。 | 注意<br>权重设置为0，该服务器不会再接受新请求。 |
| 说明 | 用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。 | 说明<br>您也可以通过按Ctrl+A选中全部文件。 |
| > | 多级菜单递进。 | 单击**设置**> **网络**> **设置网络类型**。 |
| **粗体** | 表示按键、菜单、页面名称等UI元素。 | 在**结果确认**页面，单击**确定**。 |
| Courier字体 | 命令或代码。 | 执行 `cd /d C:/window` 命令，进入Windows系统文件夹。 |
| *斜体* | 表示参数、变量。 | `bae log list --instanceid`<br>*Instance_ID* |
| [] 或者 [a\|b] | 表示可选项，至多选择一个。 | `ipconfig [-all\|-t]` |
| {} 或者 {a\|b} | 表示必选项，至多选择一个。 | `switch {active\|stand}` |

# 目录

# 1.API网关自调用说明

用户在API网关创建的API除了能被客户端调用，还能被API网关本身调用。API网关自调用允许跨Region调用，如果同Region，可以走内网调用。API网关自调同时支持跨账号调用，使用授权过的AK绑定一个类型为APIGW_FRONTEND的后端签名插件，API网关在调用自身时会使用AK生成签名发送给目标API网关进行鉴权认证。一个典型的场景是，用户有个负责分发的API，这个API绑定一个后端路由插件和后端签名插件，后端路由插件根据请求的参数来进行后端路由，路由到其他业务API上。




## 1. 配置示例

### 1.1. 业务API的配置

一个API如果想被API网关走内网形式的自身调用，需要开通专享实例，把所属的分组迁移到专享实例，并且在控制台手动生成内网调用域名：

#### 1.1.1. 开通专享实例上内网调用能力


## 1.1.2. 开通分组上的自调用内网域名

生成两个分组，并且分别为两个分组生成各自的自调用域名


假设两个业务API分组的自调用域名分别为：

```
17ff4c9189004a1d87b557606b767334-cn-huhehaote-intranet.alicloudapi.com
c6e984b2dd784c0fb843f7c2a8878b15-cn-huhehaote-intranet.alicloudapi.com
```

### 1.1.3. 在两个分组上创建两个业务API

在每个分组下分别生成一个业务API，两个API均设置为APP鉴权模式，假设两个API相关属性为：

- API1：Method：Get Path：/business1 后端地址为：

```
http://backend1.alicloudapi.com:8080/business1
```

- API2 Method：Get Path：/business2 后端地址为：

```
http://backend2.alicloudapi.com:8080/business2
```

### 1.1.4. 对两个业务API进行授权

将这两个API分别授权给同一个APP，这个APP的AK假设为 KEY:TESTKEY SECRET:TESTSECRET

## 1.2. 分发API的配置

### 1.2.1. 创建分发API

我们创建一个分发API，这个API设置为匿名访问，Method设置为Get，Path设置为/distributeAPI，这个API所属的分组的访问域名为：`17ff4c9189004a1d87b557606b767334-cn-huhehaote.alicloudapi.com`

### 1.2.2. 创建、绑定后端路由插件

创建路由插件，将路由插件绑定到分发API上。

```
---
parameters:
  target: "Query:target"
routes:
- name: backend1
  condition: "$target = 'resource1'"
  backend:
    type: "HTTP"
    address: "17ff4c9189004a1d87b557606b767334-cn-huhehaote-intranet.alicloudapi.com"
    path: "/business1"
- name: backend2
  condition: "$target = 'resource12'"
  backend:
    type: "HTTP"
    address: "c6e984b2dd784c0fb843f7c2a8878b15-cn-huhehaote-intranet.alicloudapi.com"
    path: "/business2"
```

这个路由组件的意思是，绑定了这个插件的API，收到请求的时候，判断请求中query参数target，如果target的值为resource1，就给17ff4c9189004a1d87b557606b767334-cn-huhehaote-intranet.alicloudapi.com发送一个path为/business1的http请求，target值为resource2时情况类似。

### 1.2.3. 创建、绑定后端签名插件

创建后端签名插件，将后端签名插件绑定到分发API上

```
---
type: APIGW_FRONTEND
key: TESTKEY
secret: TESTSECRET
signatureMethod: HmacSHA256
```

这个插件的意思是，所有绑定了本插件的API，给后端发送请求时，会将请求中的内容按照API网关前端签名的算法算出请求的签名，并携带在请求中。

## 2. 调用分发API

在调用之前，需要确认所有API都发布到线上，之后再做测试：

```
curl 'http://17ff4c9189004a1d87b557606b767334-cn-huhehaote.alicloudapi.com/distributeAPI?target=resource1' -i
```

发送给后端的请求：

```
GET /business1 HTTP/1.1
User-Agent: curl/7.64.1
Via: 0045e52ee3a8400b8501b4c449b28779
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Forwarded-Proto: http
X-Forwarded-For: 192.168.XX.XX, 127.0.0.1
Host: backend1.alicloudapi.com:8080
X-Ca-Request-Id: 23853B41-C54D-45E9-8C43-EE4C1E8A7889
Via: bc48a42a3d17408b991b0bb4d18c23c0
```

```
curl 'http://17ff4c9189004a1d87b557606b767334-cn-huhehaote.alicloudapi.com/distributeAPI?target=resource2' -i
```

发送给后端的请求：

```
GET /business2 HTTP/1.1
User-Agent: curl/7.64.1
Via: 0045e52ee3a8400b8501b4c449b28779
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Forwarded-Proto: http
X-Forwarded-For: 192.168.XX.XX, 127.0.0.1
Host: backend2.alicloudapi.com:8080
X-Ca-Request-Id: AFD529D2-9B24-437E-8CEC-897E0BCD8B2F
Via: bc48a42a3d17408b991b0bb4d18c23c0
```

# 2.基于Swagger完成API网关与用户CICD流程整合

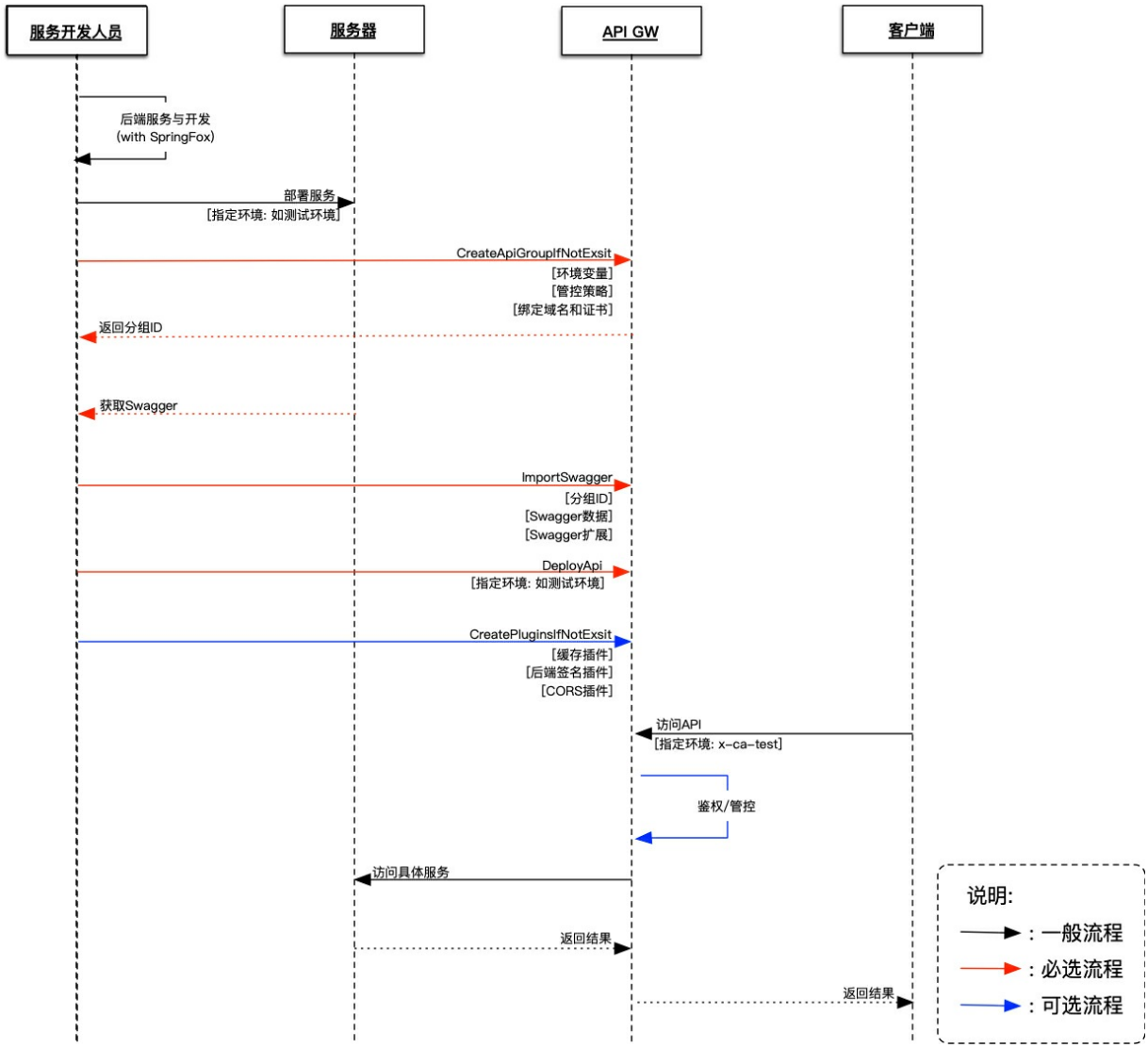
本文主要用于介绍用户CICD流程与API网关整合的通用方法实践。该方法的核心是以swagger为桥接，通过API网关的Import Swagger接口来完成您的API的自动创建和更新。

## 一、概述

API网关提供了完备的OpenAPI接口，用户可以通过这些接口完成API网关的管理需要。用户CICD集成API网关的核心，其实就是对API网关提供的管理接口进行封装整合。

如下图所示，一个整合了API网关的通用CICD流程，主要有如下步骤：

- 步骤1：自动从您的API业务代码中获取swaager定义
- 步骤2：创建API分组
- 步骤3：通过API网关的OpenAPI导入API定义，并发布在不同的环境中
- 步骤4：对API进行各项额外配置



## 自动获取Swagger定义

用户开发时，可以自行整合一个API DOC框架（本文示例选择的是SpringFox）到自己的后端服务中。这些框架一般对原有代码都是无侵入的，所以接入相对方便。同时，API DOC框架随后端服务启动时，提供Swagger的获取接口。API网关通过访问这些接口，自动获取Swagger定义

## 创建API分组

用户可以在API网关控制台手动创建分组，或者通过创建 API 分组创建分组。用户创建分组后，可以进行绑定独立域名以及绑定证书等操作。

## 通过OpenAPI导入API并发布

用户通过集成API网关提供的Import Swagger接口，并将Swagger获取接口获取的swagger数据作为参数，完成API的创建工作。在完成API在API网关的创建后，用户可以将其发布到所需的环境。完成API的发布后，用户即可访问对应的服务。

## 额外配置

为了更灵活的满足用户个性化管理需求，API网关提供了丰富的插件。包括：流量控制插件，IP访问控制插件，CORS插件以及JWT插件等。并根据需求选择配置相关的插件，来丰富该接口的鉴权及管控能力。

# 二、实现示例

本章节对提供的示例代码进行讲解说明，本文的示例代码仅提供了最基本的cicd场景，用户可根据实际情况进行调整和扩充。

## 2.1 示例代码

alibabacloud-cloudapi-java-cicd-demo

## 2.2 相关技术

本示例主要使用到的相关技术如下：

### 2.2.1 API网关相关链接

1. 通过导入Swagger创建API 介绍了如何配置可导入API网关的Swagger文件，API网关通过设计swagger支持的扩展内容，来完成标准swagger的直接导入和带有用户自定义配置的swagger导入。
2. 导入Swagger创建API 介绍了Import Swagger接口. 接口中主要参数为：

- groupId: 用于存储用户swagger导入的api的分组ID。
- data: 用户的swagger文件 。
- globalCondition: 用于用户添加自定义的配置内容, 比如通过x-apigateway-backend-info标签，加入用户自定义后端的内容。

### 2.2.2 SpringFox

为了更直观的展示如何运用swagger便捷的将后端服务同步到API网关中，本文提供了一个使用SpringFox库的后端服务。 SpringFox是一个开源的API DOC的框架，它可以通过为服务Controller接口添加注释的方式，便捷的形成swagger，并在服务运行时，提供获取swagger的接口。

## 2.3 服务端自动生成swagger定义

服务端创建了一个基于springboot的极简web服务，用于展示一般用户基于现有服务提供swagger的方法。

### 2.3.1 依赖库

```
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
```

### 2.3.2 SpringFox的配置

SpringFox是对用户原始服务相对用好的框架。用户无需对现有的接口和模型类进行任何修改，仅需要增加一个SpringFox的配置类，即可完成Springfox框架的接入工作。在服务启动后，该框架会自动解析springboot web框架中的注释，来自动生成swagger中所需的服务信息。同时，对于服务中引用的类进行解析，来生成swagger所需的模型信息。

### SpringFox框架的配置项

本示例中，配置项进行了如下核心设置：

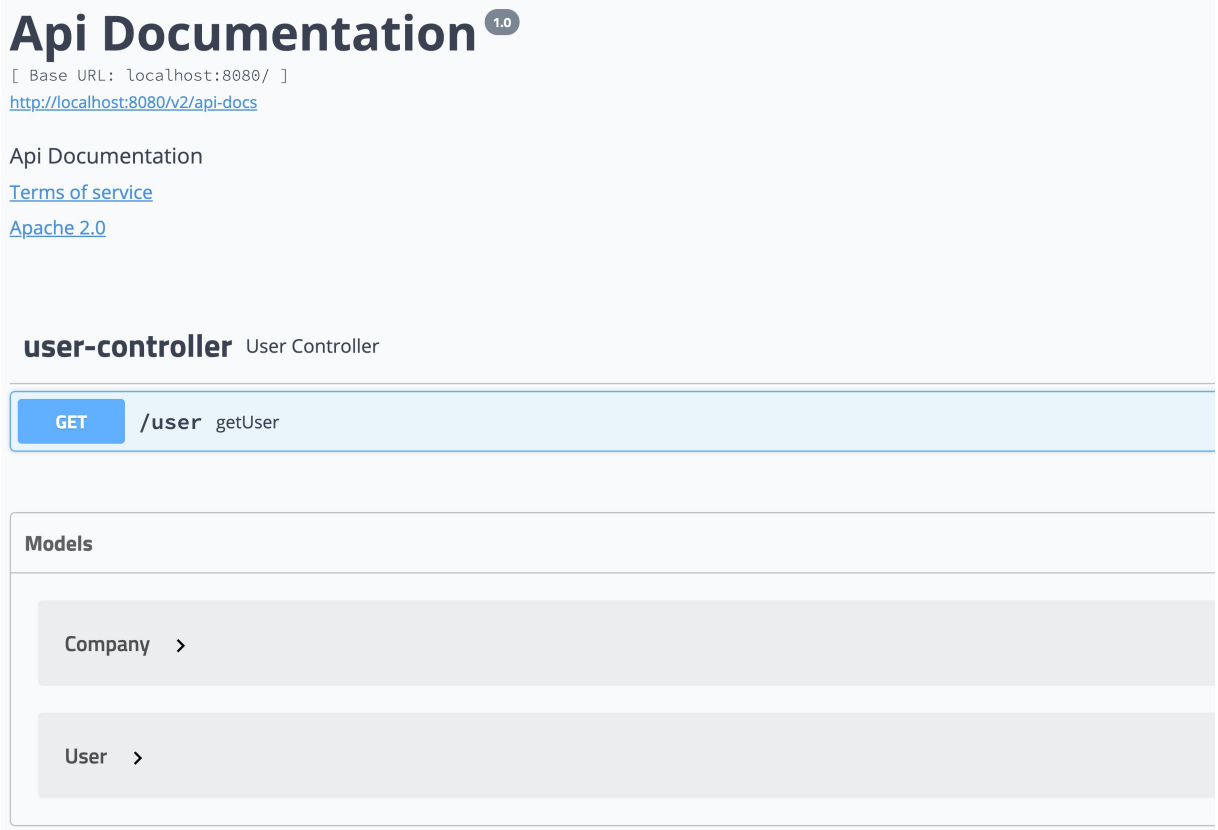
- 指出文档格式(Swagger2)
- 指出用于生成API DOC的包名

```
@Configuration
@EnableSwagger2
public class SpringFoxConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.example.swagger.controller")
)
                .paths(PathSelectors.any())
                .build();
    }
}
```

### 如何查看Swagger

启动服务后，可以通过访问 `http://localhost:8080/swagger-ui.html` 来打开swagger UI来进行swagger提供的功能，具体查看官网 `swagger.io` 说明。

从截图中我们可以看到，示例代码中的服务接口以及模型类均已被自动解析




## 如何获取Swagger

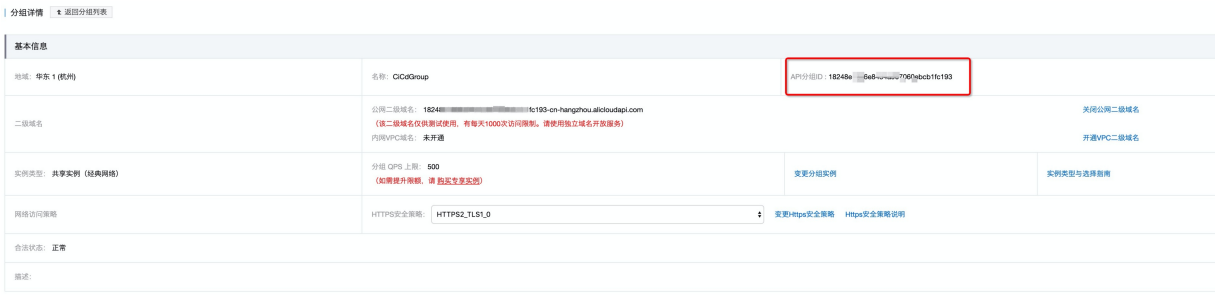
为了可以接入用户CICD的流程，获取swagger内容的接口是必不可少的。用户启动服务后，用户可以通过/v2/api-docs接口获取swagger内容。 http://localhost:8080/v2/api-docs](http://localhost:8080/v2/api-docs)

## 2.4 创建API分组

用户可以通过一下方式创建分组

## 2.4.1 通过控制台创建

用户通过在API网关控制台-分组管理界面创建分组。完成分组创建后，可以进入分组详情页，查看对应的分组ID及其他相关信息。




## 2.4.2 通过OpenAPI创建API分组

通过API网关的openapi提供的创建 API 分组接口完成APIGroup的创建。同时，用户可以配合其他API完成用户独立域名以及绑定证书等其他操作。

## 2.5 ImportSwagger接口的使用

用户能否实现基于API网关的CICD流程，其核心关键是：

- 是否有接口可以获取swagger
- 标准swagger如何导入API网关

如2.2.2章节所介绍的内容. 我们已经可以获取描述后端服务所有接口和模型的swagger. 如何将标注的Swagger导入到API网关是我们本章节的重点.

### 2.5.1 基于API网关的Swagger扩展

API网关为了便于用户体验API网关的能力，支持将原生Swagger导入到API网关中。通过这种方式，API网关会自动创建基于Swagger的所有API，并且其默认后端为MOCK形式。

但是，用户真实的CICD场景往往需要的是API网关创建的API最终真正的访问到用户的后端服务。这样，就需要用户进行后端服务信息的配置。但是，这些信息原生的swagger是不具备的，所以，API网关提供了基于Swagger的扩展。

基于Swagger的扩展主要包括：后端服务配置，后端参数配置，认证方式等API网关所必须的或者特有的内容的配置。这些拓展支持全局配置。同时，也支持每个API中单独配置。 最终达到，用户原生swagger配合API网关swagger扩展，真正实现满足用户生产环境所需的API配置。

本文示例中，仅使用了x-apigateway-backend-info扩展来进行说明。示例中，用户的后端服务HTTP服务，我们假设其线上域名为www.aliyun.com。当然，后段地址也可以通过*#环境变量#*的形式进行配置。

我们在通过导入Swagger创建API一文中查看后端为HTTP类型的后端如何配置，其中type和address为必填项，后端path和method为空时，当前api的请求地址和http方法，timeout默认为10000ms。

```
x-aliyun-apigateway-backend:
  type: HTTP
  address: 'http://www.aliyun.com'
  path: '/builtin/echo'
  method: get
  timeout: 10000
```

### 2.5.2 通过ImportSwagger导入用户原生Swagger

ImportSwagger是用户原生Swagger导入到API网关的核心入口. 这个接口有三个核心参数:

- GroupId：用于指定存储swagger导入API的分组。
- data: 用于指定用户原生的Swagger。
- globalCondition: 用于传入用户的自定义配置，如通用后端服务地址。

## Maven依赖

```
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-core</artifactId>
            <version>4.5.0</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-cloudapi</artifactId>
            <version>4.9.3</version>
        </dependency>
```

## 获取用户原生swagger

上文已经提及，用户在完成SpringFox框架引入和配置后，在服务启动时，就会自动生成/v2/api-docs接口用于获取描述后端服务的swagger。

```
    private String getSwaggerData(){
        HttpURLConnection connection = null;
        ...
        ...
        String result = null;
        try {
            URL url = new URL(swaggerApiDocUrl);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.connect();
                                                ...
            ...
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
                                                ...
            ...
            connection.disconnect();
        }
        return result;
    }
```

## 配置基于API网关的Swagger扩展

正确的配置Swagger扩展是用户通过导入Swagger创建满足其需求的API的核心环节. 这些基于API网关的swagger扩展可以通过ImportSwagger接口中的globalCondition参数传入. globalCondition的格式为Json String。

- key: 基于swagger的API网关扩展的名称, 如：x-apigateway-backend-in
- value: 对应swagger扩展的值。

示例代码中，通过配置API的通用后端地址，实现导入API网关的api可以访问到其真正的后端服务。

为了示例代码的整洁，我们根据2.2.3.1章节中http类型的后端服务说明，创建SwaggerBackendInfoBase类，用于配置后端服务信息。

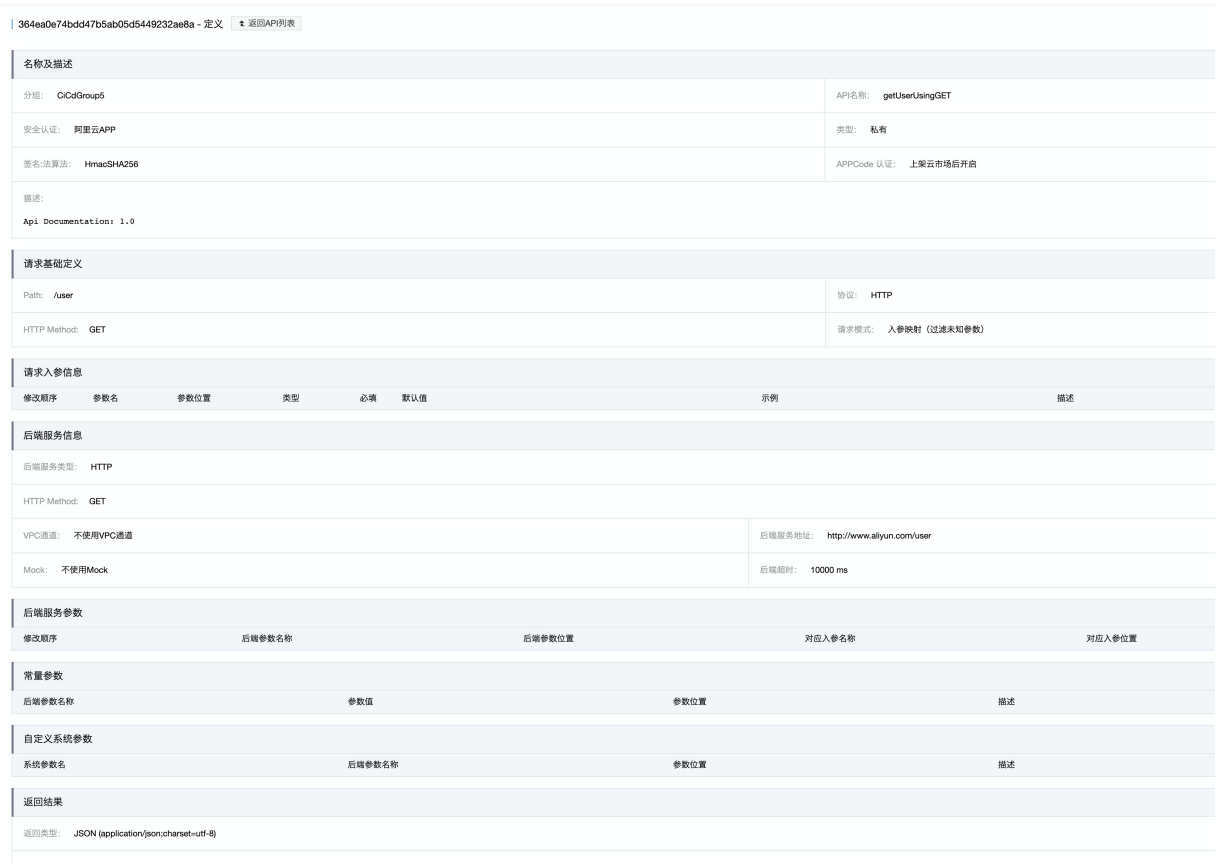
```
public class SwaggerBackendInfoBase {
    private String type;
    private String address;
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}
```

我们根据用户的后端服务信息，完成globalCondition参数的赋值。

```
            // 配置API网关所需的服务后端信息
            SwaggerBackendInfoBase info = new SwaggerBackendInfoBase();
            info.setType("HTTP");
            info.setAddress("http://www.aliyun.com");
            // 将API的后端服务后端信息
            Map<String, String> globalCondition  = new HashMap<>();
            globalCondition.put("x-aliyun-apigateway-backend", JSON.toJSONString(info));
```

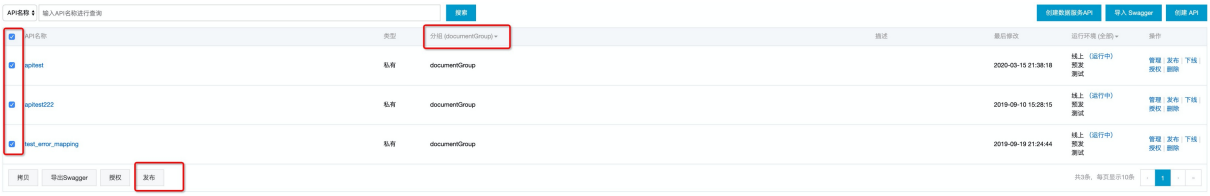
## ImportSwagger

上文中,我们已经完成了Group的创建，原生Swagger的获取以及GlobalCondition参数的创建。我们调用API网关openapi提供的ImportSwagger接口完成API的导入，并可以API网关控制台查看导入结果。




## 2.6 部署API

用户可以在API管理页面选择对应的分组，通过选择需要发布的API进行发布.用户通过发布使导入的API生效。




## 三、总结

自此，如何将用户后端服务的API接口快速的同步到API网关已经完成。API网关致力于让用户专注于后端服务的开发，将认证、流控等通用切面功能交给API网关处理，结合ImportSwagger接口的使用，完成后端服务与API网关的联动，大大减少了API网关的配置过程。同时，让API网关的配置和管理集成到用户的CICD流程中成为可能。

# 3.API网关灰度发布最佳实践

本文主要用于介绍在API后端服务版本迭代过程中，新版本服务正式发布前通过API网关进行灰度发布，A/B Test的通用方法实践。该方法的核心是通过配置后端路由插件来确保可以控制服务升级对用户造成的影响。

## 一、概述

灰度发布是指在API的新、旧版本间平滑过渡的一种发布方式。API网关客户在应用迭代过程中会不断有新版本API发布，在新版本正式发布前，可以使用灰度流量控制先进行小规模验证，将升级带来的影响限定在指定的用户范围内可以最大程度上保障线上业务的稳定运行，通过收集使用体验的数据，对应用新版本的功能、性能、稳定性等指标进行评判，然后再全量升级。

常见的灰度发布和A/B Test场景中，需要根据最终用户的信息进行区分，借助API网关的路由插件功能，能够从API HTTP Request请求中识别出和最终用户相关的参数，如userId，并进行逻辑条件判断，从而将请求分流到不同的后端服务中。

## 二、典型场景

API网关后端路由插件支持的灰度分流条件

- 使用API网关的APP进行灰度：获得授权的APP事实上代表身份不同的API调用者，这种情况下当有两个以上授权APP的时候，可以根据调用者APP将请求转发到不同后端。假设，某个API授权给了多个应用，在API后端服务有升级的情况下，为了控制升级对线上可能存在的未知影响，我们可以参照本文3.1 给出的后端路由插件配置，将其中两个应用的请求路由到新版后端服务。
- 基于JWT的灰度： 当API绑定JWT插件时，API网关可以从JWT中解析没有加密过的用户身份相关的claim，比如用户userId， 支持按照从 Token 中解析得到的参数路由到不同后端，从而实现灰度发布。比如用户可以参考本文 3.2 节给出的插件配置，通过限制userId字段的尾数，实现20%的用户请求到新版本服务。
- 基于HTTP Reuqest 内的用户参数进行灰度，当Query | Form | Header 中存在和用户身份直接相关的参数时，我们就可以直接利用该请求参数转发请求到不同后端。比如针对一部分使用老版本客户端的用户返回特定的提示信息，提示用户升级客户端，可以参考3.3节的说明进行配置。
- 针对特定访问IP段进行灰度：可以利用不同的请求源IP地址段进行后端路由，从而实现灰度发布。比如有的时候我们为了更高效的测试，需要在内测环境和生产环境使用完全相同的前端请求代码，可以参考3.4节的说明实现不同环境访问不同后端。

## 三、配置过程

本节是针对上述四种场景给出对应的后端路由插件配置，需要应用灰度发布的API绑定相应插件后，当请求符合条件时，请求会被转发到插件配置的后端， 否则将被正常访问API后端服务。当有多个灰度场景时，可以参考文档后端路由在一个后端路由插件中配置多条路由。在插件条件表达式中使用的参数要按照文档参数与条件表达式的使用的描述在插件中定义。

3.1 根据调用者APP进行灰度发布。如果用户需要指定授权给某些应用的API调用请求进入灰度环境，可以利用系统参数 CaAppId 来配置路由条件，配置如下

```
---
routes:
-  name: appGrey
   condition: "$CaAppId = 4534463 or $CaAppId = 86734572"
   backend:
      type: HTTP
      address: "http://example-test.alicloudapi.com:8080"
      path: "/web/xxx"
      method: GET
      timeout: 7000
```

3.2 根据JWT中的自定义claim进行灰度发布。假设JWT中有名为userId的自定义claim，并且userId的尾数均匀分布。则按照如下的3个步骤进行配置，可以对20%的用户进行灰度。

- 用户需要先在API上绑定配置如下的JWT插件

```
---
parameter: X-Token          # 从指定的参数中获取JWT，对应API的参数
parameterLocation: header   # API为映射模式时可选，API为透传模式下必填，用于指定JWT的读取位置，仅支持`query`,`header`
claimParameters:            # claims参数转换，网关会将jwt claims映射为后端参数
 - claimName: userId         # claim名称,支持公共和私有
   parameterName: userId     # 映射后参数名称
   location: query           # 映射后的参数位置，支持`query,header,path,formData`
#
# `Json Web Key`的`Public Key`
jwk:
   kty: RSA
   e: AQAB
   kid: showJwt
   alg: RS256
   n: gNHI8tm3lnsdCi09SrBPs9-Oau7Z1SFhIEOT2h5AJ49FSJA0XEyU4OadtV70BLIEy94dzcUK8f0e477AVoUO0RZdcXjztFtpJnA1Ktrzn9zAmKcXb2IuKXrBKkQStcKqoSbBlR84mDElp_gxfNqpmoLy0q08rkmjh1utd8E_S4QMDDaFtQ68ggJcDY-oX5FSiVidKNrKagEzQKpk5SgJFE8wpJOkW-YKouqLsL5lFyqnkgn7J3MvDqEBKqgiCY-zXYaxnkLNfkrAt7jTe4b4a2PiKD0-bHIZwzd2NVhuLGwx4pB1tFL51E-KeewZhTsoUbQ3v_ZerZ2_630WOH7IWQ
```

- 把灰度条件要用到的字段 userId 作为自定义 claim 加到JWT的payload部分，参考文档文API 网关 OpenID Connect 使用指南, 发起API请求的时候在请求header部分增加一个 X-Token 的头，值为使用私钥JWK对JWT进行签名生成的ID Token。
- 再绑定一个如下配置的后端路由插件，可以把userId尾数为“0”和 “1”的用户请求转发到指定的后端地址，从而实现对20%的用户进行灰度发布

```
---
parameters:
    userId: "Token:userId"
routes:
 -  name: userIdGrey
    condition: "$userId like '%0' or $userId like '%1'"
    backend:
       type: HTTP
       address: "http://example-test.alicloudapi.com:8080"
       path: "/web/xxx"
       method: GET
       timeout: 7000
```

3.3 根据客户端版本进行灰度发布。例如，需要灰度的API请求示例如下：

```
http://example-cn-hangzhou.alicloudapi.com/testJwtShow?clientVersion=1.9
```

需要对clientVersion小于2.0.5的版本进行灰度，配置如下：

```
---
parameters:
   clientVersion: "Query:clientVersion"
routes:
 -  name: oldVersionClient
    condition: "$clientVersion < '2.0.5'"
    backend:
       type: "MOCK"
       statusCode: 400
```

3.4 根据请求源IP进行灰度发布。当一些服务内测时，生产环境和内测环境的有访问不同后端的需求，可以根据请求源IP分发到不同版本后端。如下配置，可以利用API网关系统参数CaClientIp 将源ip地址段为106.11.31.0/24 的请求转发到指定的后端。

```
routes:
 -  name: InternalTesting
    condition: "$CaClientIp in_cidr  '106.11.31.0/24'"
    backend:
       type: HTTP
       address: "http://example-test.alicloudapi.com:8080"
       path: "/web/xxx"
       method: GET
       timeout: 7000
```

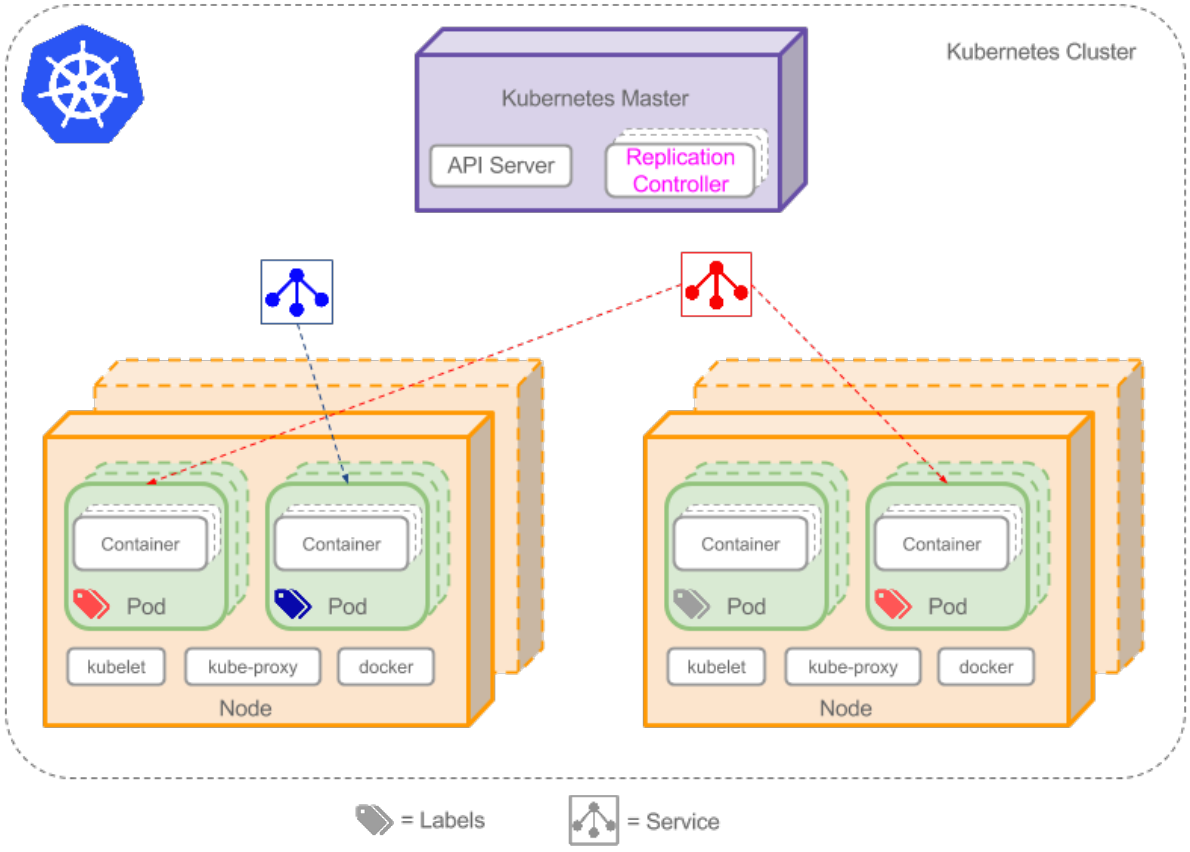
# 4.API网关为K8s容器应用集群提供强大的接入能力

## 1. Kubernetes 集群介绍

Kubernetes（k8s）作为自动化容器操作的开源平台已经名声大噪，目前已经成为容器玩家主流选择。Kubernetes在容器技术的基础上，增加调度和节点集群间扩展能力，可以非常轻松地让你快速建立一个企业级容器应用集群，这个集群主要拥有以下能力：

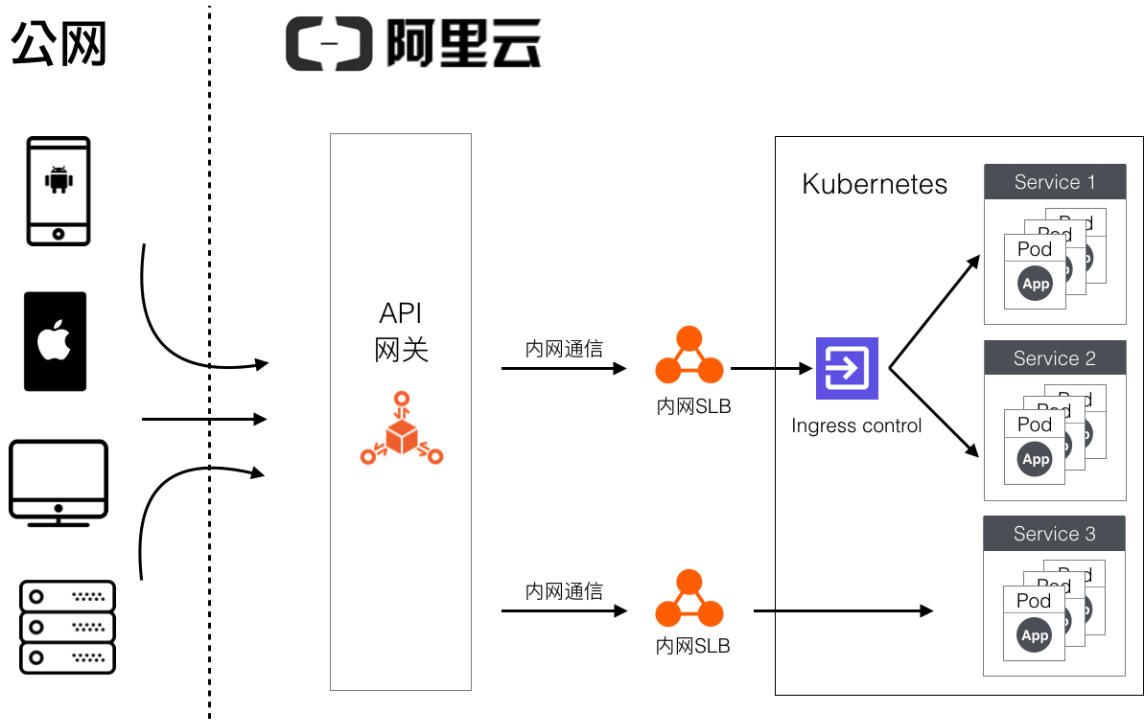
- 自动化容器的部署和复制
- 随时扩展或收缩容器规模
- 将容器组织成组，并且提供容器间的负载均衡
- 很容易地升级应用程序容器的新版本
- 提供容器弹性，如果容器失效就替换它

下面是一个典型的Kubernetes架构图：




## 2. API网关作为Kubernetes集群的接入层架构

我们可以看到Kubernetes集群是有足够理由作为应用服务的首选，但是Kubernetes集群没有足够的接入能力，特别在大型应用中，它是不能够直接对用户提供服务的，否则会有非常大的安全风险。而API网关作为成熟的云产品，已经集成了非常丰富的接入能力，把API网关放在Kubernetes集群前面作为应用集群的接入服务使用，将大大提高Kubernetes集群的服务能力，可以作为标准的大型互联网应用的标准架构。下面是使用阿里云架构图：




**是否启用Ingress Control?**

从架构图中我们可以看到，API网关作为Kubernetes集群的桥头堡，负责处理所有客户端的接入及安全工作，API网关和Kubernetes集群中Ingress Control的内网SLB或者服务的内网SLB进行通信。具体什么时候用Ingress Control呢？如果Kubernetes集群内只有一个服务，网关直接和此服务关联的内网SLB进行通信最高效。如果Kubernetes集群内有多个服务，如果使用服务的内网SLB对网关提供服务，将会生成很多SLB，资源管理起来会比较麻烦，此时我们可以使用Ingress Control做Kubernetes中服务发现与七层代理工作，API网关的所有请求发送到Ingress Control关联的内网SLB上，由Ingress Control将请求分发到Kubernetes集群内的容器内，这样我们也只需要一个内网SLB就能将Kubernetes集群内的所有服务暴露出去了。

## 3. API网关接入能力

读者要问了，接入了API网关具体能为整个架构带来哪些好处呢？下面我们列一下这种架构中，API网关具体能给整个应用带来什么价值。

1.API网关允许客户端和API网关使用多种协议进行通信，其中包括：

```
* HTTP
* HTTP2
* WebScoket
```

2.API网关使用多种方法保证和客户端之间的通信安全：

```
* 允许定义API和APP之间的授权关系，只有授权的APP允许调用；
* 全链路通信都使用签名验证机制，包括客户端和API网关之间的通信和API网关和后端服务之间的通信，保证请求在整个链路上不会被篡改；
* 支持用户使用自己的SSL证书进行HTTPS通信；
* 支持OPENID CONNECT；
```

3.API网关具备iOS/Android/Java三种SDK的自动生成能力，并且具备API调用文档自动生成能力；

4.API网关支持入参混排能力，请求中的参数可以映射到后端请求中的任何位置；

5.API网关提供参数清洗能力，用户定义API的时候可以指定参数的类型，正则等规则，API网关会帮用户确认传输给后端服务的请求是符合规则的数据；

6.API网关支持流量控制能力，支持的维度为用户/APP/API;

7.API网关提供双向通信的能力；

8.API网关提供基于请求数/错误数/应答超时时间/流量监控报警能力，所有的报警信息会使用短信或者邮件在一分钟内发出；

9.API网关已经和阿里云的SLS产品打通，用户可以将所有请求日志自动上传到用户自己的SLS中，后继好对访问日志进行统计分析；

10.API网关支持Mock模式，在联调中这个能力非常方便；

11.API网关支持用户配置调用方的IP白名单和黑名单； 12.API网关结合阿里云的云市场，为Provider提供向API使用者收费的能力。

阿里云的API网关是一个上线数年的成熟云产品，在稳定性和性能方面，经过了时间和阿里云的工程师的不断打磨，有高性能需求的用户尽管放马过来。

## 4. 在阿里云快速配置Kubernetes集群和API网关

阿里云支持快速创建Kubernetes集群，同一个Region内的Kubernetes集群和API网关的集成也非常简单，下面我们来一步一步地在阿里云中配置出本文第二节中架构设计。第二节中的架构设计中，API网关和Kubernetes有两种结合的模式，一种是API网关将请求发送到Ingress Control前的SLB，由Ingress Control将请求路由到Kubernetes对应的节点中，第二种是API网关直接将请求发送到Kubernetes中服务前的SLB，由SLB直接将请求转发到Kubernetes中服务对应的节点中。第一种模式只需要Ingress Control前有一个SLB就可以了，由Ingress Contro做服务发现与路由，第二种模式每个服务前都需要申请一个SLB，适合并发量大的场景。

## 4.1 Ingress Control模式的配置方式

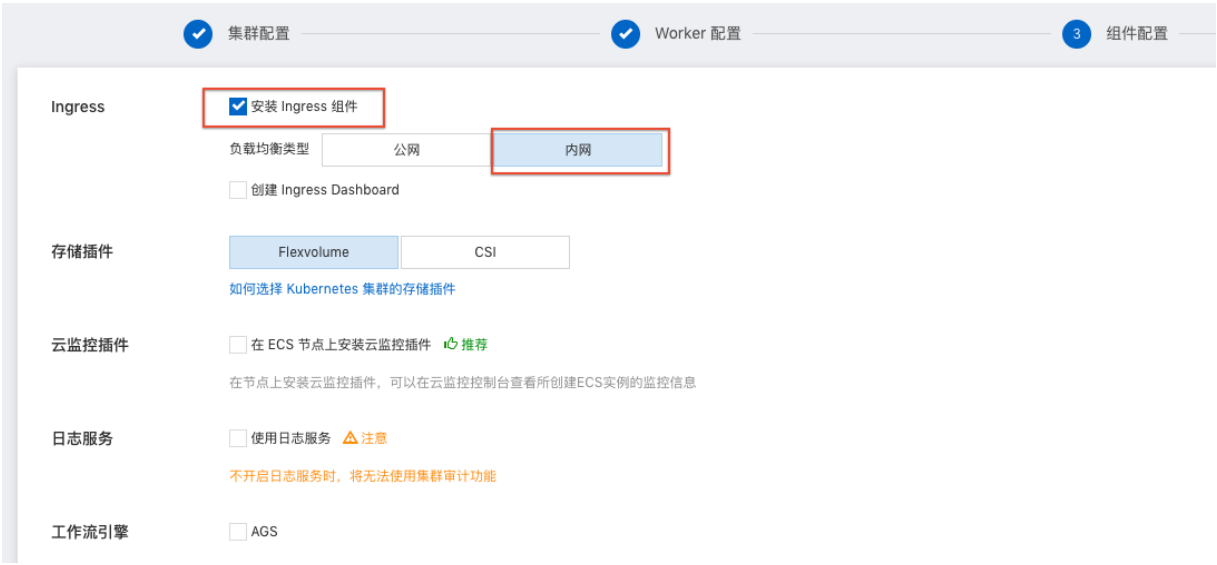
### 4.1.1 创建带有Ingress Control组件的Kubernetes集群

首先我们来通过控制台创建一个具备Ingress Control组件的Kubernetes集群。

1.进入Kubernetes集群管理控制台界面：https://cs.console.aliyun.com/#/k8s/cluster/list

2.点击左上角“创建Kubernetes集群”按钮，进入


3.在创建Kubernetes集群页面选择不同规格的，具体创建选项和常规创建参数一致，在组件配置子页面，需要注意勾选创建Ingress组件：



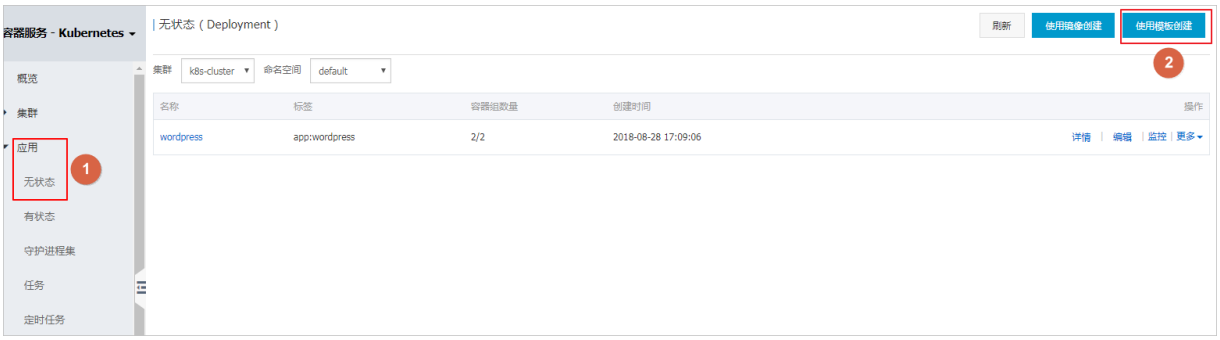

4.创建成功后可以在Kubernetes列表页面看到刚才创建的集群。

## 4.1.2 在Kubernetes集群内创建一个多容器的服务

现在集群有了，我们需要在集群内创建一个服务，这个服务由2个容器组成，每个容器都由最新的Tomcat镜像生成。容器的端口是8080，Ingress Control提供服务的端口是80。

1.进入Kubernetes集群管理控制台界面：https://cs.console.aliyun.com/#/k8s/cluster/list

2.进入Kubernetes集群的控制台页面后，点击左边菜单栏的“应用”菜单下的“无状态”按钮，进入应用列表页面后，点击右上角的“使用模板创建”按钮进入创建页面：




3.进入创建页面后，点击使用文本创建按钮，输入下面的资源编排文本点击上传按钮进行创建：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat-demo
  labels:
    app: tomcat
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:latest
        ports:
        - containerPort: 8080

---
apiVersion: v1
kind: Service
metadata:
  name: tomcat-service
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: tomcat
  sessionAffinity: None
  type: NodePort
```

对这段编排模板创建文本做下解释：使用最新的Tomcat镜像创建两个容器的意思，容器的服务端口是8080，并且创建一个命名为tomcat-service的服务，对外服务的端口为80，映射到容器8080的端口；

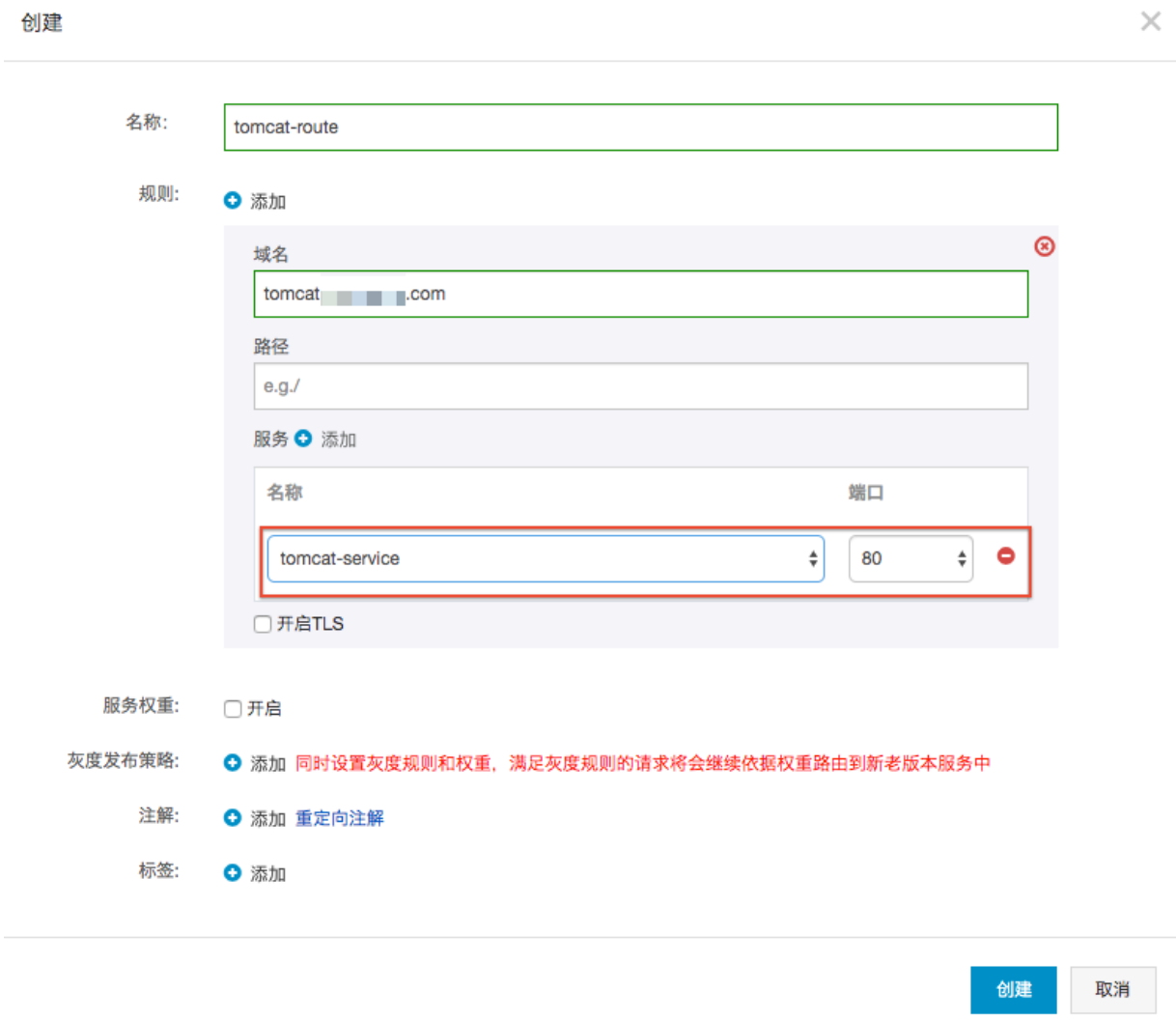
好了，目前为止，我们已经创建了一个Kubernetes集群，并且在这个集群下面创建了两个容器，每个容器上面跑着一个最新的Tomcat。这两个容器组成一个无状态应用，并且组成了一个命名为tomcat-service的服务。我们可以进入无状态应用详情页面看到整个应用的运行情况。目前我们的API网关没有办法访问到这个服务，需要我们在Ingress Control上建立一条到这个服务的路由才能把全链路打通

## 4.1.3 在Ingress Control上给服务加上路由

现在我们有应用了，还需要在Ingress Control上为这个应用建立一个服务，然后在服务上建立一条路由，这样API网关将请求发送给Ingress Control时，Ingress Control会根据配置的路由信息将请求代理到对应的应用节点上。

1.在容器服务控制台上点击“路由与负载均衡”菜单下的“服务”按钮，点击右上角的“创建”按钮；

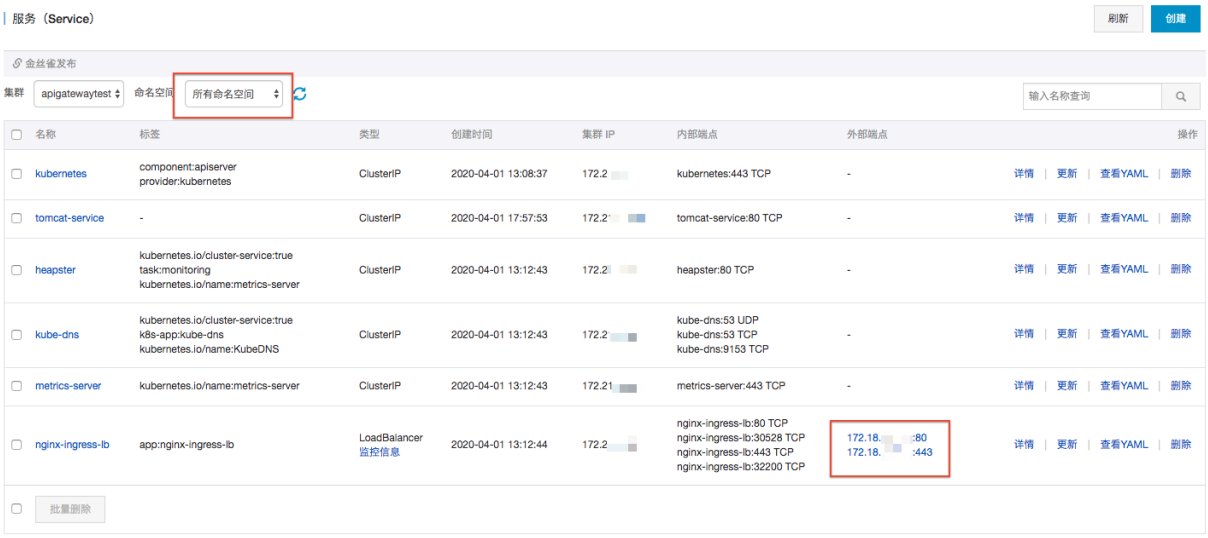
2.在创建路由页面填写服务对应的域名，所监听的端口等：




## 4.1.4 在API网关对Ingress Control的内网SLB授权

API网关如果要访问Ingress Control的内网SLB，需要增加VPC网络的授权，首先我们需要准备VPC的标识和内网SLB的实例ID，我们可以在SLB控制台获取。

1.在Kubernetes管理控制台的“服务于负载均衡”菜单下点击“服务”菜单，进入服务管理页面后，选择命名空间为"所有命名空间"后可以在列表中看到Ingress Control的内网SLB地址：




2.登录SLB控制台（https://slb.console.aliyun.com/），找到刚才创建Kubernetes服务时自动创建的VPC，点击进去查看详情，我们可以在这个页面看到VPC的标识：


好了，目前我们已经获取到了VPC的ID和SLB的实例ID,下面我们来创建API网关的授权：

3.进入API网关授权页面：https://apigateway.console.aliyun.com/#/cn-beijing/vpcAccess/list ，点击右上角的创建授权按钮，弹出创建VPC授权的小页面，将刚才查询到的VPC标识和SLB实例ID填入到对应的内容中：

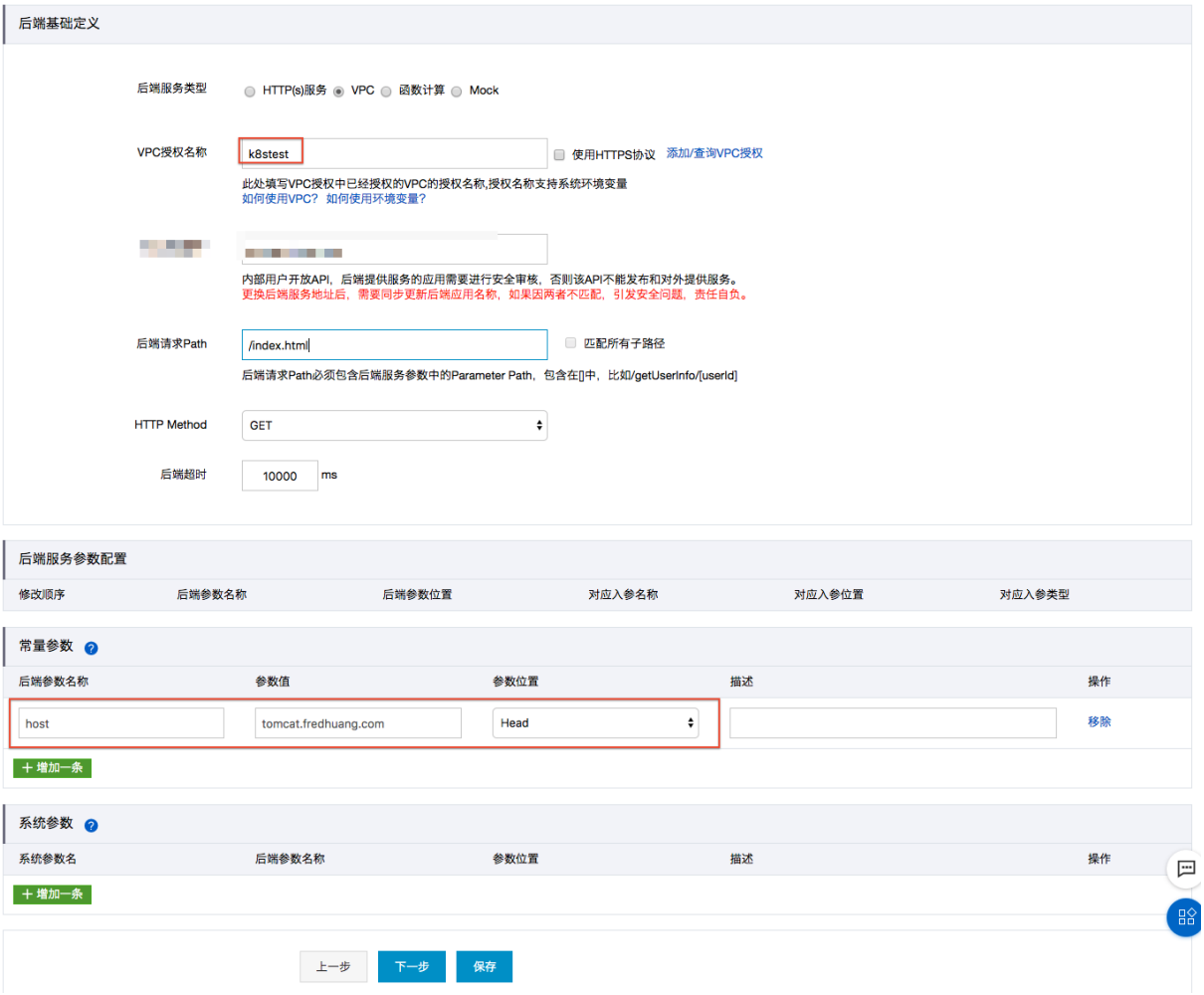

点击确认按钮后，授权关系就创建好了，需要记住刚才填写的授权名称。

### 4.1.5 创建API

在API网关控制台创建API的时候，后端服务这块，有两点需要注意的：

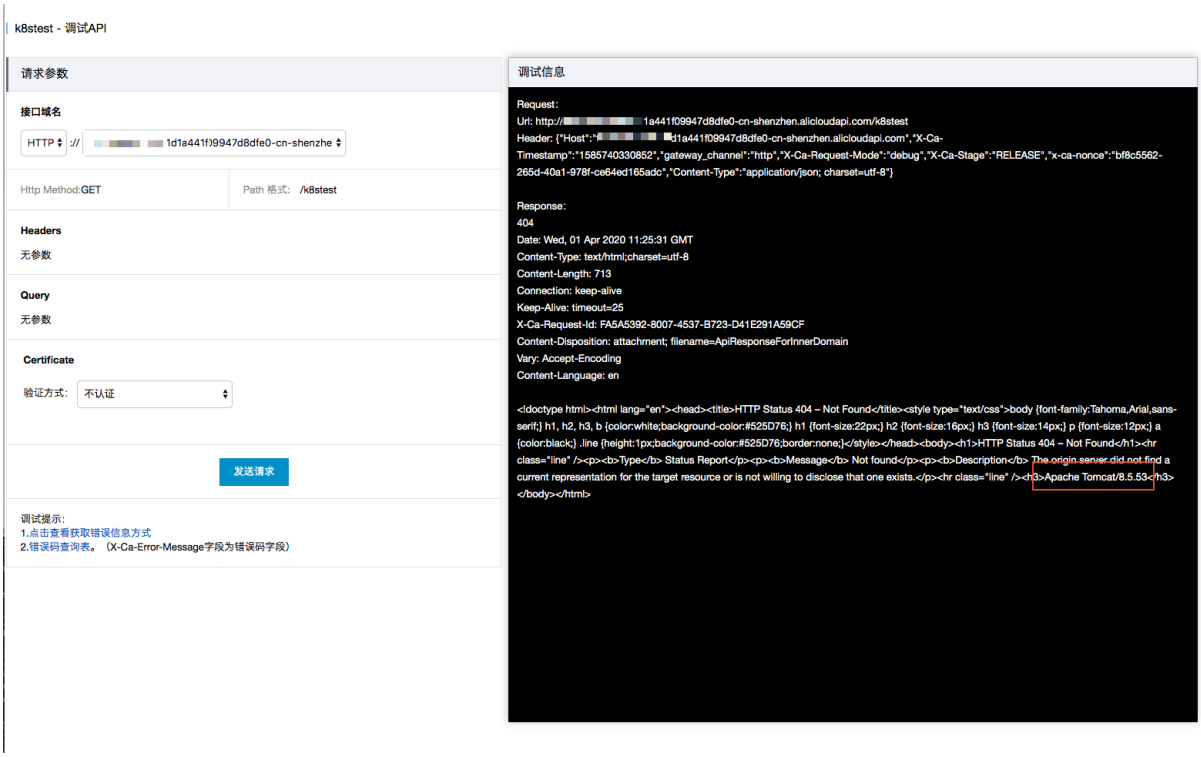
1.我们需要填写刚才创建的VPC授权名称；

2.在创建API页面的常量参数添加一个名字为host，位置header的参数，参数值设置为4.1.3节中设置的路由中填写的域名。




## 4.1.6 调用测试

API建立好了以后，我们把API发布到线上就可以使用API网关的测试工具进行测试，看看能否将请求发送到刚才创建的Kubernetes集群中去。 我们进入刚才创建好的API的详情页面，点击左侧菜单中的调试API，进入调试页面。在调试页面填写好相应的参数，点击“发起请求”按钮。




我们可以看到，请求发送到了Kubernetes的集群中的容器中，并且收到了容器中tomcate的404的应答（因为没有配置对应的页面）。

## 4.2 Kubernetes服务内网SLB结合模式的配置方式

通过Kubernetes服务的SLB结合API网关与Kubernetes的模式的配置方式与前一节中通过Ingress Control结合模式的配置方式有两点不同：1.申请Kubernetes中的容器服务时，需要指定生成内网SLB，2.找到这个SLB的VPC ID与SLB ID，使用这两个ID到API网关去进行授权即可。上一节中描述的创建Kubernetes集群的步骤，本节不再冗余描述。本节主要描述创建携带内网SLB的Kubernetes服务，并且找到这个内网SLB的IP地址。在找到SLB的IP地址后，具体授权方法和4.1.4中描述的一致，本节也不再重复描述。

### 4.2.1 生成携带内网SLB的Kubernetes服务

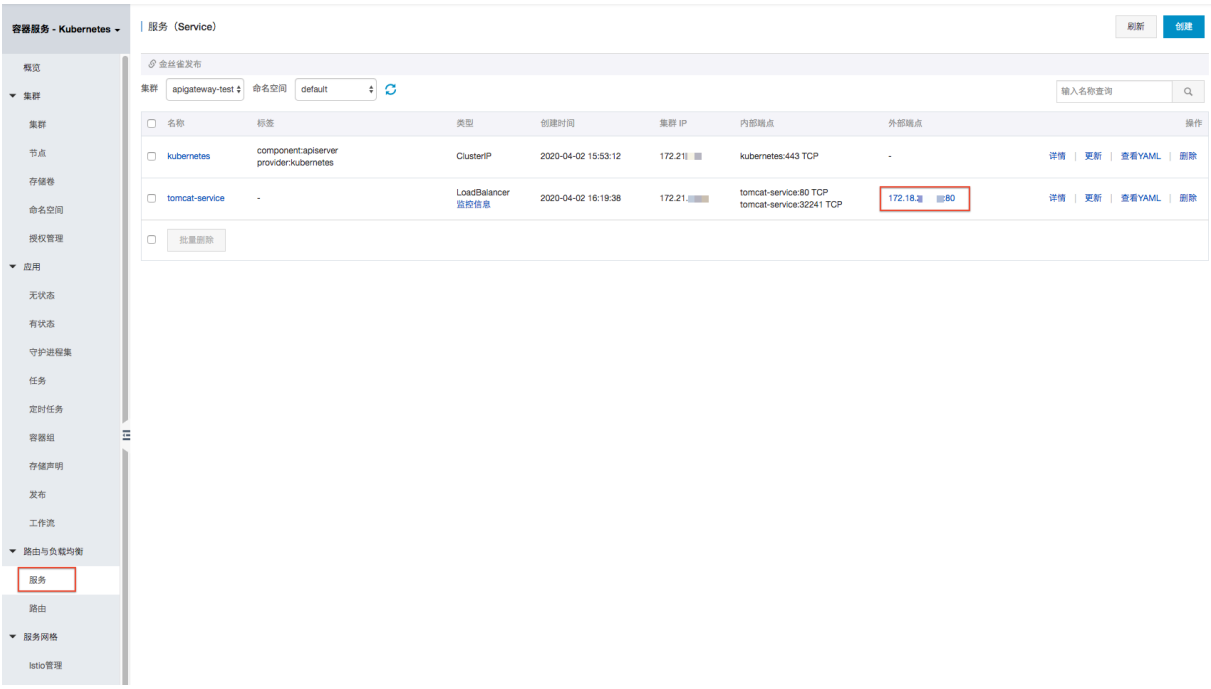
在4.1.1操作之后，我们有了一个Kubernetes集群，现在我们通过资源编排文本在这个集群中创建带有内网SLB的服务。我们注意下，本段资源编排代码和4.1.2中的资源编排代码不同的是，我们把最后一行的Type变成了LoadBalancer，并且指定了这个SLBLoadBalancer为内网：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat-demo
  labels:
    app: tomcat
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat:latest
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: tomcat-service
  annotations:
    service.beta.kubernetes.io/alicloud-loadbalancer-address-type: intranet
spec:
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
  selector:
    app: tomcat
  type: LoadBalancer
```

对这段编排模板创建文本做下解释：使用最新的Tomcat镜像创建两个容器的意思，容器的服务端口是8080，并且创建一个命名为tomcat-service的服务，这个服务前有一个内网SLB，对外服务的端口为80，映射到容器8080的端口。

## 4.2.2 在Kubernetes控制台找到服务的内网SLB地址

现在我们成功创建了一个携带内网SLB的服务，我们可以在Kubernetes控制台的“路由与负载均衡”菜单的“服务”子页面找到这个内网SLB的内网IP：




找到内网SLB之后，就可以像4.1.4中一样，去SLB的控制台找到这个SLB的VPC ID和SLB ID，并且使用这个SLB的VPC ID和SLB ID到API网关去授权了。

## 5 总结

让我们总结一下本文的内容，在前三节中，我们描述了Kubernetes集群和API网关的各项能力，并且画出了结合他俩作为后端应用服务生产的架构图。我们结合API网关+Kubernetes集群的架构，让系统具备了动态伸缩，动态路由，支持多协议接入，SDK自动生成，双向通信等各项能力，成为可用性更高，更灵活，更可靠的一套架构。

### 5.1 配置总结

在最后一节，我们描述了在阿里云的公有云如何创建一整套API网关加Kubernetes的流程，关键点在于Kubernetes集群通过Ingress Control组件服务发现，在API网关对Ingress Control组件的内网SLB进行授权后，将所有请求发送到SLB上。下面我们再总结一下通过Ingress结合API网关与Kubernetes的步骤：

1. 创建一个带有Ingress Control组件的Kubernetes的集群；
2. 使用资源编排命令，在Kubernetes集群中创建两个运行这最新版本的Tomcat的容器，并且基于这两个容器创建对应的服务；
3. 在控制台上给Ingress Control加上一条服务路由；
4. 到SLB控制台找到Ingress Control内网SLB的实例ID，在API网关创建一个VPC授权；
5. 在API网关创建API，后端服务使用刚才创建的VPC授权，并且设定一个名为host的参数，参数值使用Ingress Control服务路由设置的域名。API的请求将发送到Kubernetes集群的Ingress Control的SLB上，由Ingress Control将请求路由到容器内部的Tomcat服务上。

在高并发场景或者只有一个服务的场景，我们可以跳过Ingress Control，直接在服务前面架设一个内网SLB，并且将这个内网SLB授权给API网关，供API网关进行访问。

### 5.2 Ingress Control和SLB两种方案对比

1. 使用SLB+Ingress Control。Ingress Control可以做Kubernetes集群的服务发现和七层代理工作，如果Kubernetes集群中有多个服务，可以统一使用Ingress Control进行路由，同时只需要一个内网SLB就可以对外暴露多个服务。便于运维管理和Kubernetes集群的服务扩充。推荐使用此种方式。

2. 直接使用SLB。如果集群中某个服务的业务压力很大，可以考虑为此服务单独建立一个SLB，API网关直接连接此SLB，从而达到更高的通信效率。此种方式的弊端也比较明显，如果Kubernetes集群内有多个服务，需要为每个服务配置一个SLB，因此给运维管理带来较大工作量。

# 5.混合云API集中管理

本文介绍了如何结合云企业网（CEN），构建跨Region的集中式API管理方案。同时您也可以参考本例中的步骤，结合CEN、高速通道，构建在VPC间、VPC与本地数据中心间的集中式API管理，从而实现全网资源都可以发布在API网关、并通过API网关进行调用。

## 概述

API网关默认只能和同Region的VPC相互连通。本文中的API网关为创建在杭州Region的专享实例，对三个不同的场景API进行管理。

由于API网关是托管型的产品，因此需要在杭州Region创建一个VPC（本例中为vpc-1）用于和北京VPC以及本地数据中心IDC进行互访。

> 说明
>
> 本文的架构设计仅为更清晰的说明如何跨VPC之间的API调用，您在类似场景下使用时，只需要为API网关提供一个VPC（本例中为vpc-1）用于与其他环境（如其他Region的VPC、或本地数据中心）进行互通，此VPC除了为API网关提供接入外，并不限制其他用途。

主要场景如下：

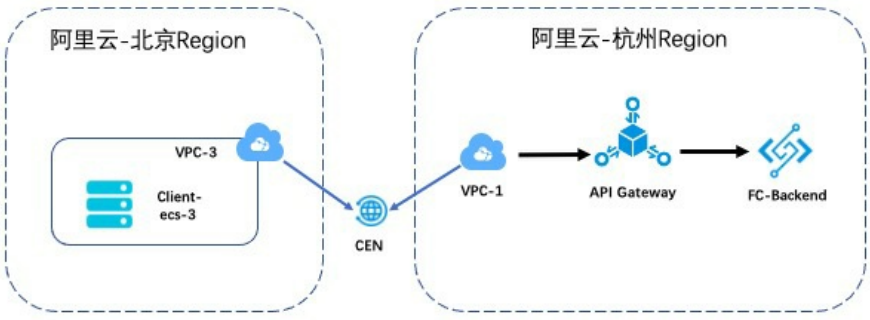
场景一：阿里云跨region内网VPC调用API网关

场景二：本地数据中心IDC机房内网调用APi网关

场景三：API网关内网访问部署在IDC机房的后端服务

## 场景一 阿里云跨region内网VPC调用API网关

本场景的客户端为北京vpc-3专有网络下的ecs-3服务器；API网关为杭州专享实例；后端服务为杭州的函数计算。架构图如下：




配置主要过程如下：

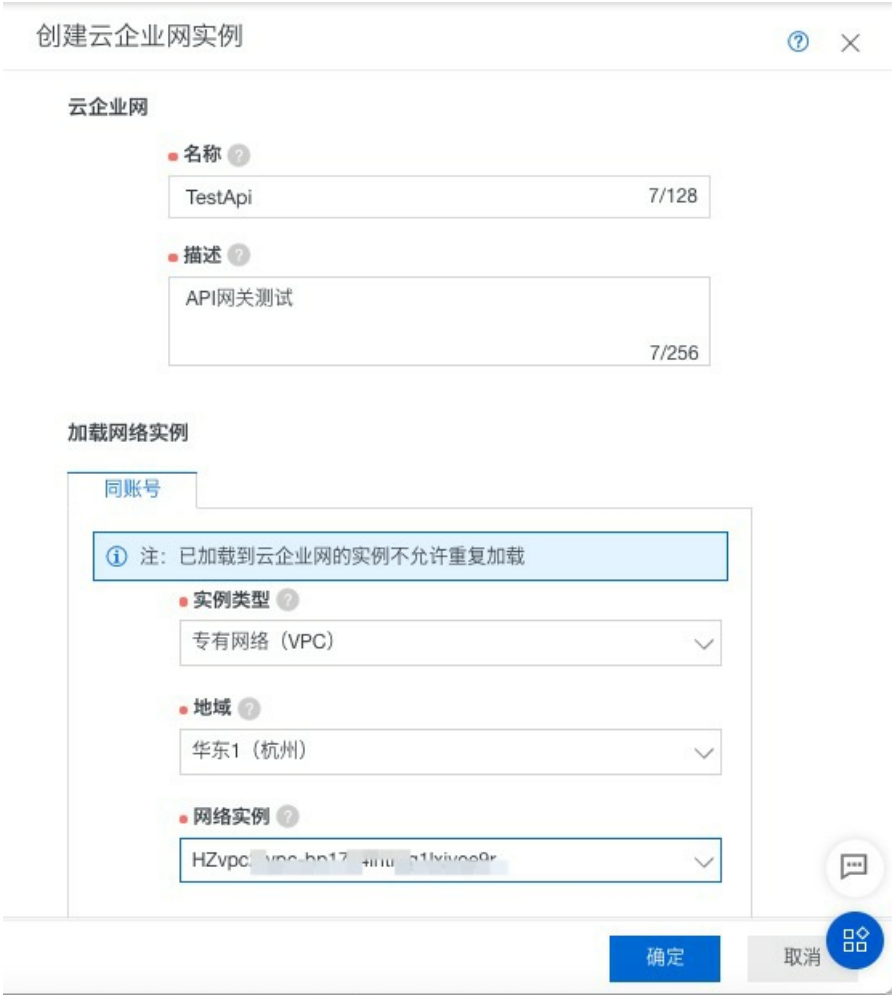
- 配置API;
- 配置云企业网（CEN），实现北京vpc-3和杭州vpc-1的内网互通；
- 配置入访VPC，添加VPC到API网关的访问权限，使得在内网需要调用API的ECS实例可以访问到API网关；

**步骤1：配置API**

参考创建后端为函数计算的API文档创建一个后端为函数计算的API。

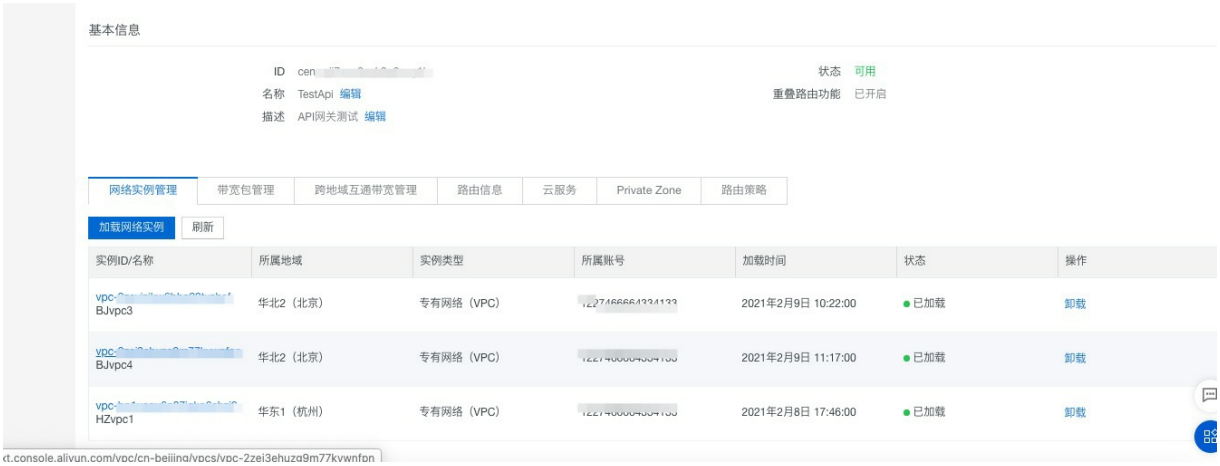
### 步骤2：创建云企业网实例

登录云企业网控制台，创建一个云企业网实例。




### 步骤3：加载网络实例

将vpc-1和vpc-3都添加到云企业网中，完成后如下图所示。




### 步骤4：配置跨地域带宽

购买一个带宽包，作为云企业网内通信需要。本例购买了一个最低的2M的带宽，您可以根据实际需要按需购买。


配置跨地域带宽设置，指定的互通地域配置带宽值，还可以将1个带宽包拆分到多个互通地域中。

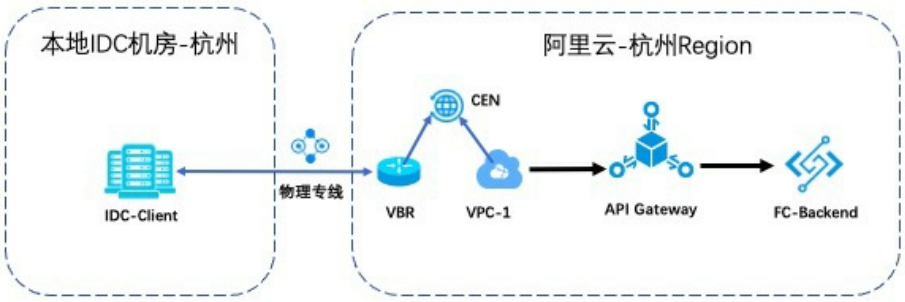

### 步骤5：配置入访VPC

在API网关实例页开通专享实例的入访VPC，点击“入访VPC”后的选择VPC，选择杭州vpc-1的vpc id，表示可以通过vpc-1内网访问到API网关。


在分组详情页面，开通内网二级域名，点击开通后，API网关会给分组分配一个内网VPC二级域名。之后专有网络vpc-3内的vpc资源就可以内网调用该分组下的API。


## 场景二 本地数据中心IDC机房内网调用APi网关

本场景客户端为杭州IDC机房，API网关为杭州专享实例，后端服务为杭州的函数计算。访问链路均为内网。架构图如下：




主要配置过程如下：

- 配置API;
- 实现本地数据中心到阿里云接入点vpc-1的内网互通；
- 配置入访VPC，添加vpc-1到API网关的访问权限，使得在IDC机房可以通过vpc-1内网调用API;

**步骤1：配置API**

参考创建后端为函数计算的API文档创建一个后端为函数计算的API。

**步骤2：实现IDC机房到阿里云接入点vpc-1的内网互通。**

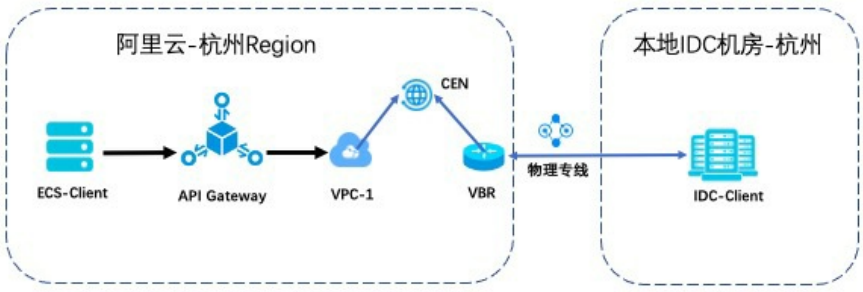
本地数据中心与阿里云可通过接入物理专线的方式打通内网，具体操作可以参考本地IDC通过专线访问云服务器ECS。

**步骤3：配置入访VPC**

参考场景一的*步骤5*，入访VPC需选择杭州vpc-1的vpc id。之后IDC机房可通过vpc-1来调用API网关分组的内网VPC域名。

## 场景三 API网关内网访问部署在IDC机房的后端服务

本场景客户端为杭州ecs，API网关为杭州专享实例，后端服务部署在杭州的IDC机房。访问链路均为内网。架构图如下：




主要配置过程如下：

- 实现本地数据中心到阿里云接入点vpc-1的内网互通；
- 创建VPC授权，即添加API网关到vpc-1的访问权限，从而API网关可以访问到VPC内的IDC机房；
- 配置API；

**步骤1：实现本地数据中心到阿里云接入点的内网互通**

本地数据中心与阿里云可通过接入物理专线的方式打通内网，具体操作可以参考本地IDC通过专线访问云服务器ECS。

**步骤2：创建VPC授权**

在配置API之前，需要先在API网关控制台创建API网关到vpc-1的访问授权，如下图所示，点击右上角的创建授权，然后填写以下内容：

— 授权名称：自定义授权名称；

— Vpc Id：填写vpc-1的VpcId；

— 实例id/ip：填写IDC机房提供服务的内网ip；

— 端口：服务端口。


**步骤3：配置API**

参考创建后端服务为VPC内资源的API文档配置并调试API。

## 8 使用限制

- 仅限API网关专享实例。