

在Kubernetes上部署SpringCloud

使用 Kuboard 在 K8S 上部署 OCP

本系列文章将描述如何使用 Kuboard 在 Kubernetes 上部署 OCP 的如下组件：

- eureka-server
- auth-server
- user-center
- api-gateway
- back-center

该系列文章的目录如下：

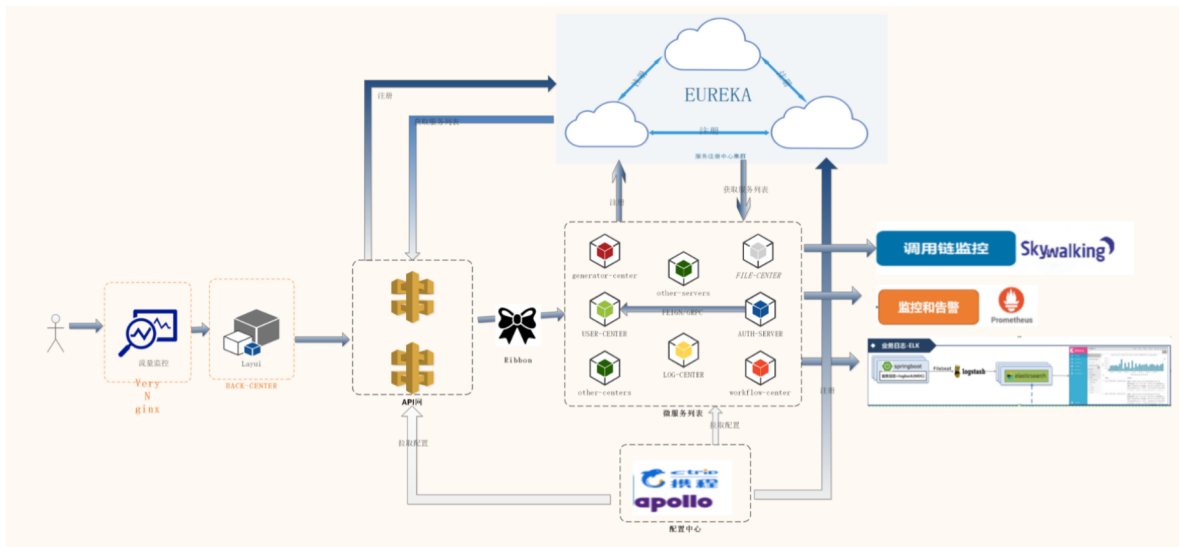
- 准备
 - [准备OCP的构建环境和部署环境](#)
 - [构建docker镜像并推送到仓库](#)
- 部署
 - [部署顺序](#)
 - [在K8S上部署eureka-server](#)
 - [在K8S上部署mysql](#)
 - [在K8S上部署redis](#)
 - [在K8S上部署auth-server](#)
 - [在K8S上部署user-center](#)
 - [在K8S上部署api-gateway](#)
 - [在K8S上部署back-center](#)
 - [重新审视配置信息](#)
- 多环境
 - [导出部署配置](#)
 - [导入部署配置](#)

OCP 介绍

简称ocp是基于layui+springcloud的企业级微服务框架(用户权限管理, 配置中心管理, 应用管理, ...), 其核心的设计目标是分离前后端, 快速开发部署, 学习简单, 功能强大, 提供快速接入核心接口能力, 其目标是帮助企业搭建一套类似百度能力开放平台的框架;

- 基于layui前后端分离的企业级微服务架构
- 兼容spring cloud netflix & spring cloud alibaba
- 优化Spring Security内部实现, 实现API调用的统一出口和权限认证授权中心
- 提供完善的企业微服务流量监控, 日志监控能力
- 提供完善的压力测试方案
- 提供完善的灰度发布方案
- 提供完善的微服务部署方案

技术介绍



功能介绍

- 统一安全认证中心
 - 支持oauth的四种模式登录
 - 支持用户名、密码加图形验证码登录
 - 支持第三方系统单点登录
- 微服务架构基础支撑
 - 服务注册发现、路由与负载均衡
 - 服务熔断与限流
 - 统一配置中心
 - 统一日志中心
 - 分布式锁
 - 分布式任务调度器
- 系统服务监控中心
 - 服务调用链监控
 - 应用吞吐量监控
 - 服务降级、熔断监控
 - 微服务服务监控
- 能力开放平台业务支撑
 - 网关基于应用方式API接口隔离
 - 下游服务基于RBAC权限管理，实现细粒度控制
 - 代码生成器中心
 - 网关聚合服务内部Swagger接口文档
 - 统一跨域处理
 - 统一异常处理
- docker容器化部署
 - 基于rancher的容器化部署
 - 基于docker的elk日志监控
 - 基于docker的服务动态扩容

代码结构

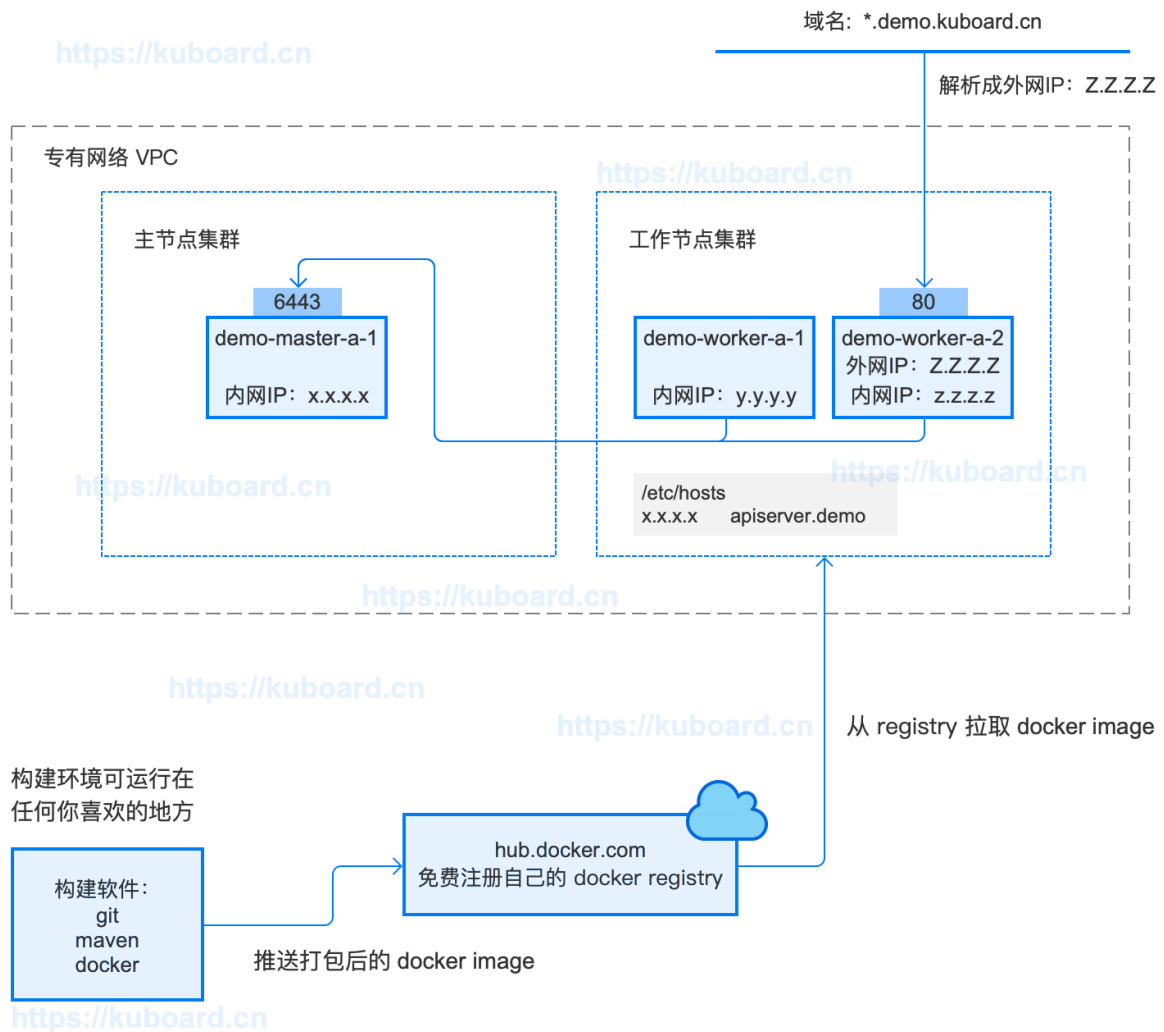
inner-intergration	-- 内部支撑工具	
db-spring-boot-starter		-- 数据库逻辑封装
redis-spring-boot-starter		-- redis逻辑封装
common-spring-boot-starter		-- common逻辑封装
swagger-spring-boot-starter		-- API文档逻辑封装
uaa-client-spring-boot-starter		-- 资源服务器逻辑封装
uaa-server-spring-boot-starter		-- 认证服务器逻辑封装
log-spring-boot-starter		-- 日志逻辑封装
register-center	-- 注册中心	
eureka-server		-- eureka服务注册中心[1111]
oauth-center	--认证中心	
auth-server		--认证服务[8000]
auth-sso		--oauth sso样例工程[9997]
api-gateway	-- 基于zuul服务网关[9200]	
new-api-gateway	-- 基于sc gateway服务网关[9200]	
business-center	-- 业务中心	
user-center		-- 用户中心[7000]
file-center		-- 文件中心[5000]
sms-center		-- 短信中心[4000]
generator-center		-- 代码生成中心[6503]
workflow-center		-- 工作流中心[7010]
web-portal	-- 页面门户	
back-center		-- 后台中心[8066]
job-center	-- 分布式定时任务中心	
job-core		--核心库
job-admin		--job管理器[8080]
job-demo		--job执行器
monitor-center	-- 监控中心	
log-center		-- 日志中心[5006]
transaction-center		-- 分布式事物中心[7970]
admin-server		-- springboot监控[9001]

准备OCP的构建环境和部署环境

环境要求

Open Capacity Platform 是基于 Java Spring Cloud 的微服务架构，为了将其部署到 Kubernetes 上，我们需要准备如下环境：

- 硬件
 - 1 台 Linux 服务器，配置不低于2核4G，CentOS 7.6，（本系列教程称该机器为 master 节点）用途：
 - Kubernetes master 节点
 - 编译 OCP 源码并打包 docker 镜像的构建机
 - 2 台 Linux 服务器，配置不低于2核4G，CentOS 7.6，（本系列教程称该机器为 worker 节点，前期可以只有一台，随着负载增加再增加节点）用途：
 - Kubernetes worker 节点
- 软件
 - Kubernetes 集群及管理软件
 - Kubernetes 最新版本，参考 [安装Kubernetes单Master节点](#)
 - Kuboard 最新版本，参考 [安装Kuboard](#)
 - Master节点
 - Docker 已在安装 Kubernetes 时完成安装
 - JDK 1.8
 - maven
 - 镜像仓库



本文假设您已经完成了 Kubernetes 集群的安装，假设您准备在 Kubernetes master 节点上执行构建过程，并将使用简短的篇幅介绍如何在 master 节点上做好构建环境准备

- 安装 JDK 1.8
- 安装 maven 3.6.2
- 安装 git
- 下载 open-capability-platform 的代码仓库

在 master 节点上安装 JDK1.8

以root身份在 master 节点上执行：

```
1 | yum install java-1.8.0-openjdk\* -y
```

在 master 节点上安装 maven

- 在 [maven 官网](#)

获取最新版 maven 的 binary 文件下载链接，例如 apache-maven-3.6.2-bin.tar.gz 的下载地址为 <http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-3/3.6.2/binaries/apache-maven-3.6.2-bin.tar.gz>

以 root 身份在 master 节点上执行：

```
1 #切换到 /root 用户目录
2 cd /root
3 # 下载 tar.gz
4 wget http://mirrors.tuna.tsinghua.edu.cn/apache/maven/maven-
3/3.6.2/binaries/apache-maven-3.6.2-bin.tar.gz
5 # 解压 tar.gz
6 tar -xvf apache-maven-3.6.2-bin.tar.gz
```

以root身份在 master 节点上执行 `vim /root/.bash_profile` 修改 `.bash_profile` 文件, 向 `PATH=` 所在行的行尾增加 `:/root/apache-maven-3.6.2/bin` 如下所示:

```
1 # User specific environment and startup programs
2
3 PATH=$PATH:$HOME/bin:/root/apache-maven-3.6.2/bin
4
5 export PATH
```

TIP

您可以把 `apache-maven-3.6.2` 放在您自己喜欢的位置

检查安装结果: 退出 master 节点的 shell 终端, 并重新以 root 用户登录 master 节点的 shell 终端, 执行命令 `mvn -version`, 输出结果如下所示:

```
1 Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-
27T23:06:16+08:00)
2 Maven home: /root/apache-maven-3.6.2
3 Java version: 1.8.0_222, vendor: Oracle Corporation, runtime:
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.222.b10-1.e17_7.x86_64/jre
4 Default locale: en_US, platform encoding: UTF-8
5 OS name: "linux", version: "3.10.0-957.21.3.e17.x86_64", arch: "amd64",
family: "unix"
```

在 master 节点上安装 git

- 以root身份在 master 节点执行:

```
1 # 安装 git
2 yum install -y git
3 # 查看已安装版本
4 git version
```

在 master 节点上克隆 ocp 代码仓库

- 以root身份在 master 节点上执行:

```
1 # 切换到 /root 用户目录
2 cd /root
3 # 克隆 ocp 代码仓库
4 git clone https://gitee.com/owenwangwen/open-capacity-platform.git
```

以root身份在 master 节点上执行：

```
1 # 安装 tree
2 yum install tree -y
3 # 查看 ocp 代码目录
4 tree /root/open-capacity-platform -L 2
```

输出结果如下所示：

```
1 /root/open-capacity-platform
2 |─ api-gateway
3 | |─ pom.xml
4 | |─ src
5 |─ business-center
6 | |─ file-center
7 | |─ generator-center
8 | |─ pom.xml
9 | |─ sms-center
10 | |─ user-center
11 | |─ workflow-center
12 |─ inner-intergration
13 | |─ common-spring-boot-starter
14 | |─ db-spring-boot-starter
15 | |─ log-spring-boot-starter
16 | |─ pom.xml
17 | |─ rabbitmq-spring-boot-starter
18 | |─ redis-spring-boot-starter
19 | |─ swagger-spring-boot-starter
20 | |─ uaa-client-spring-boot-starter
21 | |─ uaa-server-spring-boot-starter
22 |─ job-center
23 | |─ doc
24 | |─ job-admin
25 | |─ job-core
26 | |─ job-demo
27 | |─ pom.xml
28 |─ LICENSE
29 |─ monitor-center
30 | |─ admin-server
31 | |─ log-center
32 | |─ pom.xml
33 | |─ transaction-center
34 | |─ zipkin-center
35 |─ new-api-gateway
36 | |─ pom.xml
37 | |─ src
38 |─ oauth-center
```

```
39 | | └─ auth-server
40 | | └─ auth-sso
41 | | └─ pom.xml
42 | └─ pom.xml
43 | └─ README.en.md
44 | └─ README.md
45 | └─ register-center
46 | | └─ eureka-server
47 | | └─ pom.xml
48 | └─ sql
49 | | └─ 01.user-center.sql
50 | | └─ 02.oauth-center.sql
51 | | └─ 03.file-center.sql
52 | | └─ 04.sms-center.sql
53 | | └─ 05.log-center.sql
54 | | └─ 06.job-center.sql
55 | | └─ 07.workflow-center.sql
56 | | └─ 08.transaction-center.sql
57 | | └─ 09.batch-center.sql
58 | └─ tuning-center
59 | | └─ pom.xml
60 | | └─ test-common-spring-boot-starter
61 | | └─ test-log-spring-boot-starter
62 | | └─ test-redis-spring-boot-starter
63 | | └─ test-spring-boot-starter
64 | └─ web-portal
65 | | └─ back-center
66 | | └─ pom.xml
67
68 42 directories, 23 files
```

构建docker镜像并推送到仓库

本文假设您已经完成了 [准备OCP的构建环境和部署环境](#)，在该文档的最后，我们将 Open Capacity Platform 的代码仓库克隆到了 master 节点的 /root/open-capacity-platform。

TIP

- 假设您在 <https://hub.docker.com>

-

完成了注册，并获得了账号（本教程中，作者使用的 hub.docker.com 的账号为 `ocpsample`）

从完成后续教程的角度来看，您也可以不注册自己的 hub.docker.com 账号，不执行构建和镜像推送的操作，直接使用 ocpsample 已经发布到 hub.docker.com 上的 docker 镜像完成本系列教程后面的部分。匿名在 <https://hub.docker.com>

-

- 搜索 `ocpsample`

修改配置

修改 `/root/open-capacity-platform/pom.xml` 文件，修改其中的字段：

- `project --> properties --> docker.host` 修改为 `unix:///var/run/docker.sock`
- `project --> properties --> docker.image.prefix` 修改为 `ocpsample` (此处使用你在 <https://hub.docker.com> 的账号)

TIP

如果您使用自己的 docker 镜像仓库, 您的 `docker.image.prefix` 要复杂一些, 请参考 [使用私有仓库中的docker镜像](#)。具体而言, 前缀应该由您的docker镜像仓库的多个参数组成, 例如:
`my-registry.example.com:5000/example`

- 蓝色部分: registry 地址
- 绿色部分: registry 端口
- 紫色部分: repository 名字

```
1 <properties>
2   <java.version>1.8</java.version>
3   <core.version>2.0.1</core.version>
4   <log4j2.version>2.1</log4j2.version>
5   <jasypt.version>1.14</jasypt.version>
6   <hutool.version>4.1.13</hutool.version>
7   <fastjson.version>1.2.60</fastjson.version>
8   <disruptor.version>3.4.1</disruptor.version>
9   <maven.compiler.source>1.8</maven.compiler.source>
10  <maven.compiler.target>1.8</maven.compiler.target>
11  <docker.host>unix:///var/run/docker.sock</docker.host>
12  <docker.image.prefix>ocpsample</docker.image.prefix>
13  <spring-boot.version>2.0.4.RELEASE</spring-boot.version>
14  <net-devh-grpc.version>2.0.1.RELEASE</net-devh-grpc.version>
15  <spring-platform.version>Cairo-SR3</spring-platform.version>
16  <spring.social.version>1.1.6.RELEASE</spring.social.version>
17  <spring-security.version>5.1.1.RELEASE</spring-security.version>
18  <security-oauth2.version>2.3.4.RELEASE</security-oauth2.version>
19  <commons-collections4.version>4.1</commons-collections4.version>
20  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
21  <hibernate-validator.verion>5.0.2.Final</hibernate-validator.verion>
22  <flowable.version>6.4.1</flowable.version>
23  <spring-cloud-dependencies.version>Finchley.SR2</spring-cloud-
dependencies.version>
24 </properties>
```

构建 jar 包

- 在 master 节点执行 `cd /root/open-capacity-platform`
- 在 master 节点执行 `mvn clean package -DskipTests`, 输出结果如下所示

```
1 [INFO] zipkin-center ..... SUCCESS [
  0.009 s]
2 [INFO] zipkin-center-es ..... SUCCESS [
  0.003 s]
3 [INFO] es-server ..... SUCCESS [
  1.156 s]
4 [INFO] es-client ..... SUCCESS [
  0.515 s]
5 [INFO] new-api-gateway ..... SUCCESS [
  1.956 s]
6 [INFO] web-portal ..... SUCCESS [
  0.004 s]
7 [INFO] back-center ..... SUCCESS [
  1.735 s]
8 [INFO] -----
  -----
9 [INFO] BUILD SUCCESS
10 [INFO] -----
  -----
11 [INFO] Total time: 01:09 min
12 [INFO] Finished at: 2019-09-24T13:23:44+08:00
13 [INFO] -----
  -----
```

TIP

- 构建时间

:

- 在首次执行时 `mvn clean package -DskipTests` 时，由于需要从 maven 中央仓库拉取 OCP 所依赖的 jar 包以及 maven 所需要的 plugin 等资源，耗时比较长，作者第一次执行该命令时等候了 32 分钟。
- 在第二次以及以后的执行中，只需要 1 分钟左右的时间即可完成构建

- 构建结果

: 完成构建后，jar包存在于 java 模块的

```
1 | ./target/
```

目录下，例如 eureka-server 子模块的jar包位于路径

```
1 | /root/open-capacity-platform/register-center/eureka-server/target/eureka-
  server.jar
```

- jar文件所在目录是 java 项目的一种约定
- jar文件的名字则在由子模块 pom.xml 文件中 `build.finalName` 字段指定。例如 `/root/open-capacity-platform/register-center/eureka-server/pom.xml` 中 `build.finalName` 字段为 `${project.artifactId}`，该变量的取值为 `eureka-server`，因此jar文件的名字为 `eureka-server.jar`
- 其他jar文件可在对应子模块的 `./target/` 目录下找到

构建docker镜像并推送到registry

eureka-server

- 执行命令 `cd /root/open-capacity-platform/register-center/eureka-server`
- 执行命令 `mvn docker:build` 输出结果如下所示:

```
1  ProgressMessage{id=null, status=null, stream=null, error=null,
2  progress=null, progressDetail=null}
3  Successfully built 2fa9cf75f7ba
4  Successfully tagged ocpsample/eureka-server:latest
5  [INFO] Built ocpsample/eureka-server
6  [INFO] -----
7  [INFO] BUILD SUCCESS
8  [INFO] -----
9  [INFO] Total time: 17.900 s
10 [INFO] Finished at: 2019-09-24T13:45:45+08:00
    [INFO] -----
    [INFO] -----
```

OCP中由于全是 java 项目，因此使用了 docker 的 maven 插件 `com.spotify.docker-maven-plugin` 进行 docker 镜像的构建

- 该插件的配置在子模块目录 pom.xml 文件的 `build.plugins` 字段
- OCP 将该插件的两个全局配置项作为属性提取到了 `/root/open-capacity-platform/pom.xml` 文件中，即我们在 [修改配置](#) 章节提前修改好的属性 `docker.image.prefix` 和 `docker.host`

`docker-maven-plugin` 在 maven 环境中给我们带来了很大的便利，您仍然有必要熟悉 `docker build` 命令，以便您能够轻松地处理 php/nodejs/python 等类型的项目

```
1  <!-- 首先加入pom ${docker.image.prefix} : 这个是你的dockerhub注册上面的名字
2  gitgeek 这个是我注册的
3  ${project.artifactId} : 项目的名称 dockerDirectory : dockerfile的文件路径
4  -->
5  <plugin>
6  <groupId>com.spotify</groupId>
7  <artifactId>docker-maven-plugin</artifactId>
8  <version>0.4.13</version>
9  <configuration>
10 <imageName>${docker.image.prefix}/${project.artifactId}</imageName>
11 <dockerDirectory>src/main/docker</dockerDirectory>
12 <!-- docker远程服务器地址 -->
13 <dockerHost>${docker.host}</dockerHost>
14 <resources>
15 <resource>
16 <targetPath></targetPath>
17 <directory>${project.build.directory}</directory>
18 <include>${project.build.finalName}.jar</include>
19 </resource>
20 </resources>
    </configuration>
  </plugin>
```

- 执行命令 `docker login` 输入您在 `hub.docker.com` 的用户名密码，输出结果如下所示

```
1 | Login with your Docker ID to push and pull images from Docker Hub. If you
  | don't have a Docker ID, head over to https://hub.docker.com to create one.
2 | Username: ocpsample
3 | Password:
4 | WARNING! Your password will be stored unencrypted in
  | /root/.docker/config.json.
5 | Configure a credential helper to remove this warning. See
  | https://docs.docker.com/engine/reference/commandline/login/#credentials-store
6 |
7 |
8 | Login Succeeded
```

执行命令 `docker push ocpsample/eureka-server:latest`

如果推送成功，将显示

```
1 | The push refers to repository [docker.io/ocpsample/eureka-server]
2 | daf5b9e5336a: Layer already exists
3 | b59179c58f56: Layer already exists
4 | ceaf9e1ebef5: Layer already exists
5 | 9b9b7f3d56a0: Layer already exists
6 | f1b5933fe4b5: Layer already exists
7 | latest: digest:
  | sha256:7fb0f1b566e9d9183c65f2ceed3740d210072b2317523b6399ef2745fe87b367 size:
  | 1371
```

- 推送速度取决于您与 `hub.docker.com` 之间的网络连接速度，时间有点儿长，您也可以直接使用本教程已经推送上去的镜像，直接进入在 Kubernetes 上部署的环节

auth-server

过程与 `eureka-server` 相同，因此只写命令，不再写过程

```
1 | # 切换到目录
2 | cd /root/open-capacity-platform/oauth-center/auth-server
3 | # docker build
4 | mvn docker:build
5 | # docker push
6 | docker push ocpsample/auth-server:latest
```

user-center

```
1 # 切换到目录
2 cd /root/open-capacity-platform/business-center/user-center
3 # docker build
4 mvn docker:build
5 # docker push
6 docker push ocpsample/user-center:latest
```

api-gateway

```
1 # 切换到目录
2 cd /root/open-capacity-platform/api-gateway
3 # docker build
4 mvn docker:build
5 # docker push
6 docker push ocpsample/api-gateway:latest
```

back-center

```
1 # 切换到目录
2 cd /root/open-capacity-platform/web-portal/back-center
3 # docker build
4 mvn docker:build
5 # docker push
6 docker push ocpsample/back-center:latest
```

查看镜像推送结果

使用 ocpsample 的用户名和密码登录 <https://hub.docker.com>

-

的 `Repositories` 菜单下可查看推送上去的镜像，如下图所示：

TIP


- 请使用您自己在 `hub.docker.com` 的用户名和密码

hub.docker.com

dockerhub Search for great content (e.g., mysql) Explore Repositories Organizations

ocpsample Search by repository name... Create Repository +

ocpsample / back-center Updated 24 minutes ago	☆ 0	↓ 1	🌐 PUBLIC
ocpsample / api-gateway Updated 27 minutes ago	☆ 0	↓ 2	🌐 PUBLIC
ocpsample / user-center Updated 4 hours ago	☆ 0	↓ 1	🌐 PUBLIC
ocpsample / auth-server Updated 5 hours ago	☆ 0	↓ 1	🌐 PUBLIC
ocpsample / eureka-server Updated 5 hours ago	☆ 0	↓ 1	🌐 PUBLIC

 Tip: Not finding your repository? Try switching namespace via the top left dropdown.

深度解析

OCP的项目结构

Open Capacity Platform 中，使用一个三层结构在组织java项目：

- 【项目根路径】：

对应 open-capacity-platform 的项目根路径，包含：

- pom.xml，所有 java 项目的 parent，该 pom.xml 中需要以 module 的形式定义子模块的名称

- 【模块分类】：

对应 api-gateway / business-center / ... register-center 等模块

- pom.xml，需要定义该模块的所有子模块
- 某些情况下，第二层直接就是子模块代码，如 api-gateway / new-api-gateway 等

- 【子模块代码】：

对应（以 business-center 为例）file-center / generator-center / sms-center / user-center / workflow-center 等模块

- pom.xml
- 子模块的代码

这种代码组织结构的优点:

- OCP 可以将所有模块有机组织起来一个命令 `mvn clean build` 就可以完成所有项目的构建
- 模块之间的依赖设定比较便捷

缺点

- 所有模块全都在一个 gitee (github) 项目中, 不能够按模块设定代码查看权限

另一种项目结构

相对于类似 OCP 的多层结构, 还可以用扁平的结构组织 java 项目, 即每一个模块一个 gitee (github) 项目, 如果这样, OCP 的项目结构将如下所示:

- **ocp-root** 用于存放原 `open-capacity-platform/pom.xml` 文件
- **ocp-\$(category)-root** 用于存放原 `open-capacity-platform/register-center/pom.xml` 等第二层 **【模块分类】** 的 pom.xml
- **ocp-(category)-(modules)** 用于存放原 `open-capacity-platform/register-center/eureka-server` 等第三层 **【子模块代码】** 的项目文件

这种代码结构的优点:

- 可以按照模块进行权限控制
- 可以将 DevOps 的 pipeline 文件直接存在 项目的根路径, 并按模块控制 pipeline

缺点

- 需要先将 **ocp-root / ocp-category-root** 以及被其他模块依赖的 **ocp-modules** 提前构建并推送到私有 maven 仓库中 (此时你还要自己搭建 maven 仓库)
- 模块的构建顺序需要人为控制

如何选择

不同的结构, 没有最好, 只有最合适:

- OCP 作为开源项目, 将所有模块放在同一个 gitee 项目中, 便于管理和维护, 同时也是为了让大家快速上手
- 大型团队按服务领域拆分模块之后, 每个团队负责一个到多个模块, 同时要控制代码的可见范围, 可能倾向于操作起来更复杂的扁平式结构
- 先熟悉 OCP 的分层结构, 在对 maven 以及 java 的包依赖有深入了解之后, 在必要的时候再尝试扁平式的结构, 对初学者来说, 是一条友好的学习路径

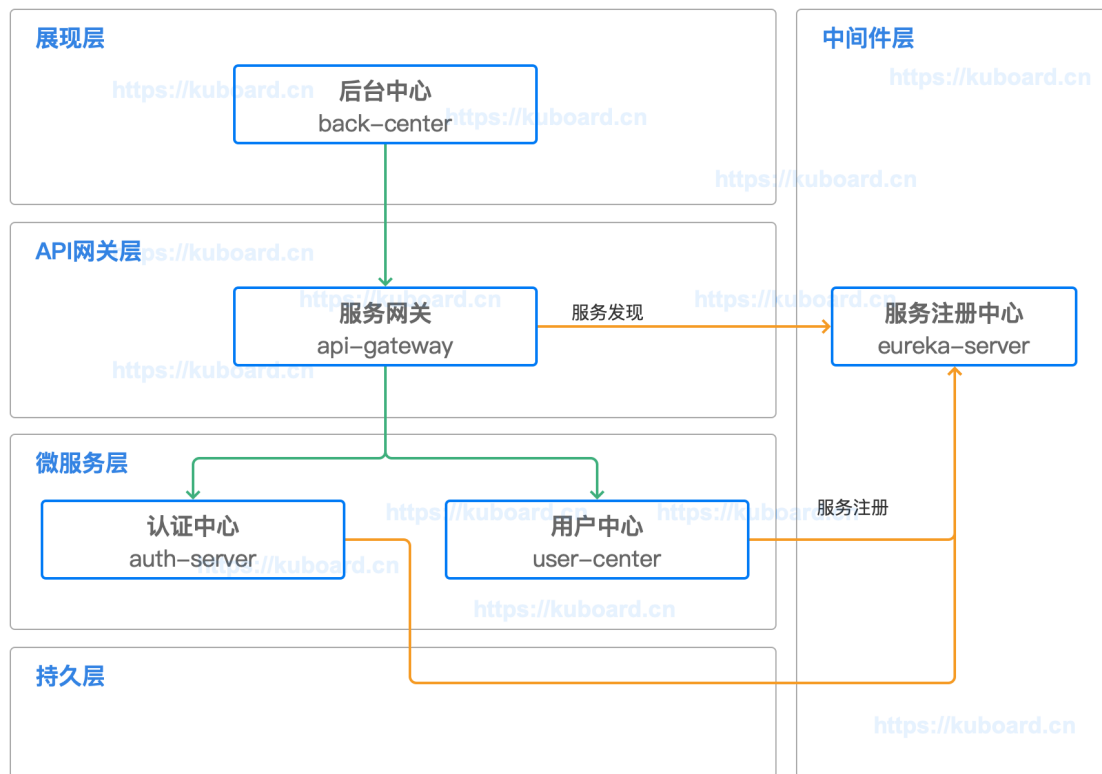
部署顺序

部署顺序的确定

本教程将在 KUbernet是上部署 Spring Cloud - Open Capacity Platform 微服务架构的如下模块:

- 服务注册中心 eureka-server -- **【中间件层】**
- 用户中心 user-center -- **【微服务层】**
- 认证中心 auth-server -- **【微服务层】**
- 服务网关 api-gateway -- **【API网关层】**
- 后台中心 back-center -- **【展现层】**

他们之间的依赖关系如下图所示:



本教程将按照如下顺序部署这些微服务模块：

1. 服务注册中心 eureka-server
2. 用户中心 user-center
3. 认证中心 auth-server
4. 服务网关 api-gateway
5. 后台中心 back-center

在决定按照什么顺序部署这些微服务组件时，主要考虑的因素有：

- 依赖关系
 - 0 依赖（不依赖任何其他模块）的最先部署
 - 按依赖链条顺序部署
 - 上图的依赖顺序为：eureka-server <-- (auth-server / user-center) <-- api-gateway <-- back-center
 - 依赖链条末端的最后部署
- 分层关系
 - 一个模块可以依赖处于下一层的模块
 - 一个模块可以依赖处于同一层的模块
 - 尽量避免跨层依赖，例如 back-center 直接依赖 auth-server
 - 展现层、API网关层、微服务层、持久层的组件都有可能依赖中间件层，在本例中，API网关层的 api-gateway 和 微服务层的 auth-server、user-center 都依赖于中间件层的 eureka-server
- 解耦关系
 - 如果从服务注册/服务发现的角度来审视服务调用者 api-gateway、服务提供者 auth-server/user-center、服务注册中心 eureka-server 三者的关系：
 - 服务注册中心 eureka-server 必须先于服务调用者和服务提供者存在，否则调用者和提供者都有可能启动失败
 - 服务提供者先于服务调用者存在，遵循了依赖关系

- 服务调用者可以先于服务提供者存在并正常启动，此时，如果服务提供者完成启动并向注册中心注册，服务调用者后续才发现提供者的存在，并进一步向服务提供者发送接口调用请求。此时服务注册中心使得我们可以向一个已经运行多时（api-server已存在）的情况下添加新的微服务（比方说 product-center）

TIP

按照解耦关系这几个微服务模块的部署顺序也可以调整为：

1. 服务注册中心 eureka-server
2. 服务网关 api-gateway
3. 用户中心 user-center
4. 认证中心 auth-server
5. 后台中心 back-center

服务编排工具的特点

docker-compose

需要严格定义不同模块之间的依赖关系，依赖链条前序的模块没有完成启动，将不会尝试后续模块的启动。

Kubernetes

Kubernetes 中，不定义模块之间的依赖关系。你把控制器（Deployment/StatefulSet/DaemonSet 等）按任意顺序部署到 Kubernetes 中，Kubernetes 就开始尝试为你维持期望的 Pod 副本数。如果依赖模块不存在，被依赖模块会启动失败？是的，但是，没关系，Kubernetes 会先等一下然后再尝试启动，直到被依赖模块出现以后，依赖模块正常启动为止。请参考 [重启策略](#)

TIP

在测试验证阶段，我们仍然严格按照期望的顺序进行部署，否则，虽然 Kubernetes 会不知疲倦地尝试维持期望的 Pod 副本数，但是开发者仍然期望快速看到正常运行的结果。

在K8S上部署eureka-server

本文假设您已经完成了 [在Kubernetes上部署SpringCloud-OCP](#) 教程的前序步骤：

- [准备OCP的构建环境和部署环境](#)
- [构建docker镜像并推送到仓库](#)

也可以使用 `ocpsample/eureka-server:latest` 镜像

- 理解 Spring Cloud Eureka 组件，请参考 [Eureka服务注册与发现](#)

理解eureka-server

本章节参考 eureka-server 的 [代码仓库](#)

，并着重从容器化部署的角度来理解 Spring Cloud eureka-server 以及 OCP 中 eureka-server 的配置文件。

`open-capacity-platform/register-center/eureka-server/src/main/resources` 目录中包含了 eureka-server 的配置文件，如下所示：

```
1 | └─ application-slave0.yml
2 | └─ application-slave1.yml
3 | └─ application-slave2.yml
4 | └─ application-slave3.yml
5 | └─ application.yml
6 | └─ bootstrap.yml
```

其中，`application-slave0.yml`，`application-slave1.yml`，`application-slave2.yml`，`application-slave3.yml` 为 spring boot 的 4 个 profile 配置，他们之间最重要的差异在于以下两个字段：

- `server.port`
- `eureka.client.defaultZone`

而 `application.yml` 文件中则指定了 `application-slave0.yml` 为默认 profile。通过 `eureka.client.service-url.defaultzone` 字段不难看出：

- 配置文件 `slave0` 为一组，在运行 eureka-server 单节点时使用（该文件中还有一些特定于测试环境的配置项）
- 配置文件 `slave1/slave2/slave3` 为一组，在运行 eureka-server 高可用时使用，为了避免端口冲突，为每一个实例单独定义了 `server.port` 字段，通过 `--spring.profiles.active` 启动参数为 eureka-server 的实例指定激活的配置文件

确定部署方案

在 Kubernetes 中部署多个 eureka-server 的实例组成集群时，主要有如下考虑因素：

- 每个 eureka-server 需要被赋予一个唯一的 id，通过字段

```
1 | eureka.instance.instance-id
```

指定。OCP 中，该字段的配置为

```
1 | ${spring.application.name}:${spring.cloud.client.ip-
  | address}:${spring.application.instance_id:${server.port}}
```

- Kubernetes 为每一个 Pod 分配一个 IP 地址，此要求可以满足
- eureka-server 的每一个实例需要知道集群中其他实例的地址和端口号，通过字段

```
1 | eureka.client.serviceUrl.defaultzone
```

指定

- 请参考 [StatefulSet的使用场景](#) 以理解为何选择 StatefulSet 部署 eureka
- 请参考 [StatefulSet稳定的网络ID](#) 以理解 StatefulSet 如何为其中的 Pod 分配 DNS name
- eureka-server 的多个实例之间，不能存在端口冲突
 - 请参考 [Kubernetes的网络模型](#) 以理解 Kubernetes 中如何避免端口冲突

我们在 Kubernetes 上部署 eureka-server 时：

- 使用 StatefulSet 部署 eureka-server，副本数量为 3
- 使用 OCP eureka-server 的 application-slave0.yml 这个 profile
- 使用环境变量覆盖

```
1 eureka.client.service-url.defaultZone
```

取值，将其设置为：

```
1 http://cloud-eureka-0.cloud-
  eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-
  eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-
  eureka.ocp.svc.cluster.local:1111/eureka
```

- TIP

通过 cloud-eureka-0.cloud-eureka 也可以访问到对应的 POD，但是此处必须使用完整域名，否则 eureka-server 将不被认为是 available

- 使用环境变量覆盖 `eureka.instance.prefer-ip-address` 取值，将其设置为：`false`
- 为 eureka-server 创建 Ingress，并分配域名

```
1 cloud-eureka.ocp.demo.kuboard.cn
```

- 关于 Ingress，请参考 [Ingress通过互联网访问您的应用](#)
- 该域名由 `工作负载名.名称空间.集群名字.一级域名` 组成，这种命名规则下，只需要将 `*.demo.kuboard.cn` 的域名解析指向集群 Ingress Controller 的地址就可以，在测试环境中配置新的模块时非常方便。

部署eureka-server

本教程将 eureka-server 及其他 OCP 组件部署到 `ocp` 名称空间，并假设您已经创建好了该名称空间，参考 [创建名称空间](#)

- 在 Kuboard 界面中进入 `ocp` 名称空间，并点击页头的按钮 `创建工作负载`，如下图所示：
填写表单：

字段名称	填写内容	备注
------	------	----

字段名称	填写内容	备注
服务类型	StatefulSet	
服务分层	中间件	
服务名称	eureka	
服务描述	服务注册中心	
副本数量	3	
容器名称	eureka-server	
镜像	ocpsample/eureka-server:latest	也可以使用自己构建的镜像
抓取策略	Always	
环境变量	eureka.client.service-url.defaultZone= http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka , http://cloud-eureka-1.cloud-eureka.ocp.svc.cluster.local:1111/eureka , http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka eureka.instance.prefer-ip-address=false	填入 kubernetes 环境变量名后面不带 <code>=</code>
Service	NodePort: 协议 TCP 服务端口 1111 节点端口 31111 容器端口 1111	可从节点端口访问
Ingress	域名: cloud-eureka.ocp.demo.kubernetes.cn 路由配置: 映射URL / 服务端口 1111	可通过域名访问

Kuboard 设置 ▾
ocp
ocp 创建工作负载 取消编辑

事件 < 后返

基本信息

* 服务类型

* 服务分层

* 服务名称

* 标签

服务描述 6/50

* 副本数量

存储卷声明模板

数据卷 Volume 帮助 ▾

运行容器组 Pod

Docker 仓库的用户名密码 帮助 ▾

ServiceAccount

容器组重启策略 帮助 ▾

节点选择 帮助 ▾

自动分配 指定节点 匹配节点

由 Kubernetes 根据各节点的运行时状态自动分配

工作容器

* 容器名称

* 镜像

* 抓取策略

Command

Args

环境变量 =

=

挂载点

就绪检查 无

存活检查 无

帮助 ▾

访问方式 Service

不配置 ClusterIP (集群内访问) NodePort (VPC内访问)

协议	服务端口	节点端口 (可空)	容器端口	操作
TCP	1111	31111	1111	<input type="button" value="删除"/>

节点端口如果为空, 则由集群自动分配

互联网入口 Ingress 帮助 ▾

标签

* 域名

HTTPS

* 路由配置

映射URL	服务端口	操作
/	1111	<input type="button" value="删除"/>

- 点击 **保存** 按钮
- 点击 **应用** 按钮
- 点击 **完成** 按钮
 - 等待 eureka-server 完成部署
 - 根据您服务器到 hub.docker.com 的网速不同, 等候的时间约 1-5 分钟

查看部署结果

按照上面的部署方式, 有如下两种方式可以从浏览器访问 eureka-server 的界面:

- 使用域名: <http://cloud-eureka.ocp.demo.kuboard.cn/>

- 使用节点端口: `http://${任意节点的IP地址}:31111`

eureka-server 界面如下图所示:

The screenshot shows the Spring Eureka server interface. At the top, there is a navigation bar with the Spring Eureka logo and links for '主页' (Home) and '最近启动的1000个服务' (Recently started 1000 services). The main content is divided into several sections:

- 系统状态 (System Status):** A table showing environment details.

环境	test	当前时间	2019-09-27T06:08:36 +0000
数据中心	default	运行	02:34
		启用租约到期时间	true
		续订阈值	0
		续订 (最后一分钟)	6
- 自保存模式已关闭。如果出现网络/其他问题, 这可能不会保护实例过期。** (Self-preservation mode is disabled. If network/other issues occur, this may not protect instances from expiring.)
- 服务副本 (Service Replicas):** A list of service replicas with their IDs: `cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local` and `cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local`.
- 当前注册的服务实例 (Currently registered service instances):** A table showing instance details.

应用	申请	可用性区域	状态
EUREKA-SERVER	n/a (3)	(3)	UP (3) - eureka-server:192.168.144.153:1111, eureka-server:192.168.144.130:1111, eureka-server:192.168.144.189:1111
- 一般信息 (General Information):** A table showing system metrics.

名称	值
total-avail-memory	104mb
environment	test
num-of-cpus	1
current-memory-usage	83mb (79%)
server-uptime	02:34
registered-replicas	http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka/, http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka/
unavailable-replicas	
available-replicas	http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka/, http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka/
- 示例信息 (Example Information):** A table showing instance details.

名称	值
ipAddr	192.168.144.153
status	UP

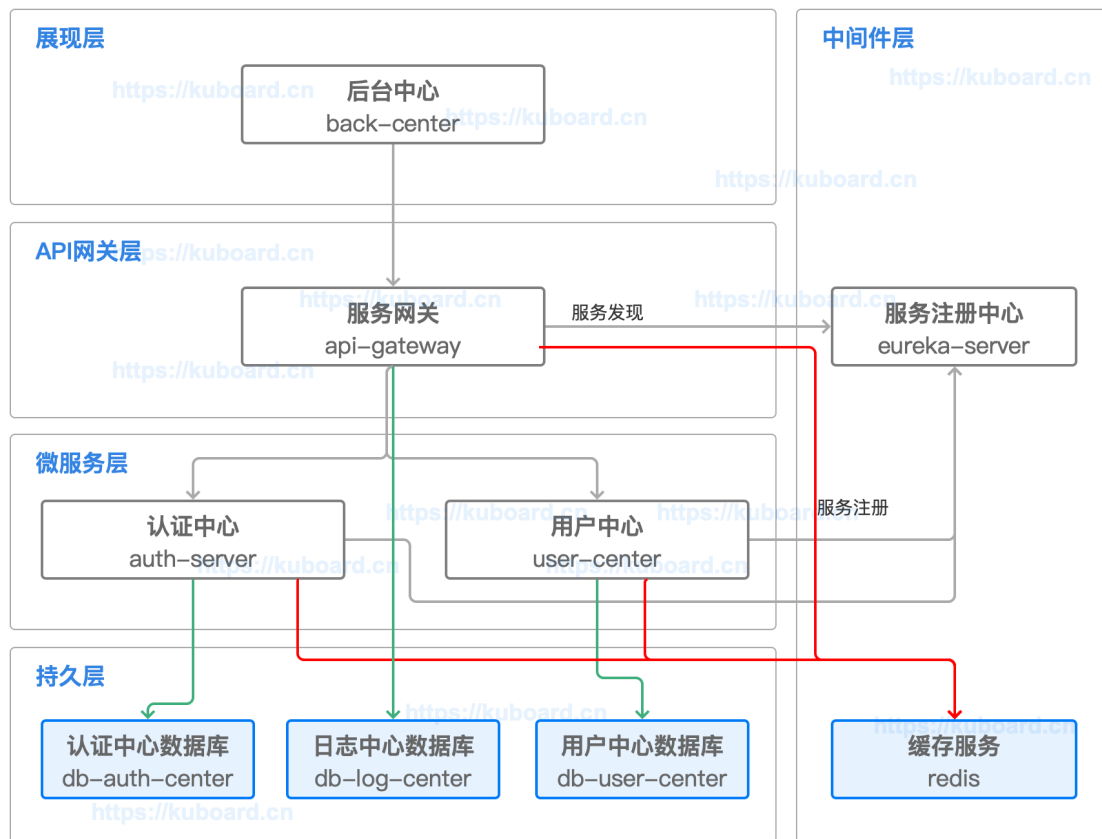
在K8S上部署mysql

OCP 的 auth-server 和 user-center 都有自己的数据库, 同时 auth-server、user-center 和 api-gateway 又都依赖于 redis 缓存服务和 log-center 数据库。这使得我们的部署结构演变成下图所示:

我们必须先完成 db-auth-center、db-user-center、db-log-center 和 redis 的部署, 才能继续部署 auth-server 和 user-center。本文描述了如何部署 db-auth-center、db-user-center、db-log-center。

WARNING

将 mysql 部署到 K8S 中, 可以非常便捷地搭建一套测试环境, 但是, 在生产环境里, 并不建议直接将 mysql 部署到 K8S 上。



构建并推送mysql镜像

OCP 要求 mysql 版本 5.7 以上，当我们在 K8S 上部署 mysql 时，将选择 [mysql 官方镜像](#)

-

并基于此镜像，构建自己的 mysql 镜像，以便：

- 把数据库初始化脚本打包到镜像中
 - 这样每次部署一个新的 mysql 实例时，可以自动初始化 OCP 所需要的表结构
- 把自定义的数据库配置文件 my.cnf 打包到镜像中
- 设置环境变量 本文档不涉及

auth-center-mysql

- 在 master 节点上，执行命令 `cd /root/open-capacity-platform/sql` 切换当前目录。

假设您已经完成了 [准备OCP的构建环境和部署环境](#)

- 执行命令 `vim auth-center-my.cnf` 以创建文件，其内容如下：

此配置用于解决 mysql 使用时的乱码问题。

```

1 [mysqld]
2
3 init_connect='SET collation_connection = utf8_unicode_ci'
4 init_connect='SET NAMES utf8'
5 character-set-server=utf8
6 collation-server=utf8_unicode_ci
7 skip-character-set-client-handshake

```

执行命令 `vim dockerfile_auth-center` 以创建文件，其内容如下：

```
1 FROM mysql:5.7.26
2 ADD auth-center-my.cnf /etc/mysql/conf.d/my.cnf
3 ADD 02.oauth-center.sql /docker-entrypoint-initdb.d/02.oauth-center.sql
4 EXPOSE 3306
```

执行命令 `docker build -f dockerfile_auth-center -t ocpsample/auth-center-mysql:latest`

.

TIP

如果使用私有仓库，则应该按照如下格式填写镜像名：

`my-registry.example.com:5000/example/auth-center-mysql:latest`

- 蓝色部分：registry 地址
- 绿色部分：registry 端口
- 紫色部分：repository 名字
- 红色部分：image 名字
- 棕色部分：image 标签

执行命令 `docker login` 登录镜像仓库

执行命令 `docker push ocpsample/auth-center-mysql:latest`

大约2-5分钟，可完成镜像推送

user-center-mysql

- 在 master 节点上，执行命令 `cd /root/open-capacity-platform/sql` 切换当前目录。（与上一个步骤目录相同）
- 执行命令 `vim user-center-my.cnf` 以创建文件，其内容如下：

```
1 [mysqld]
2
3 init_connect='SET collation_connection = utf8_unicode_ci'
4 init_connect='SET NAMES utf8'
5 character-set-server=utf8
6 collation-server=utf8_unicode_ci
7 skip-character-set-client-handshake
```

执行命令 `vim dockerfile_user-center` 以创建文件，其内容如下：

```
1 FROM mysql:5.7.26
2 ADD user-center-my.cnf /etc/mysql/conf.d/my.cnf
3 ADD 01.user-center.sql /docker-entrypoint-initdb.d/01.user-center.sql
4 EXPOSE 3306
```


SQL脚本问题

01.user-center.sql 脚本中包含一些 create FUNCTION 的语句，已经确认这些 FUNCTION 并不被用到。在执行 coker build 之前，请确保这些 create FUNCTION 的语句被删除，否则您将在 mysql 初始化时碰到如下错误：

```
1 | ERROR 1064 (42000) at line 246: You have an error in your SQL syntax; check  
  | the manual that corresponds to your MySQL server version for the right syntax  
  | to use near '' at line 3
```

- 执行命令 `docker build -f dockerfile_user-center -t ocpsample/user-center-mysql:latest .`
- 执行命令 `docker push ocpsample/user-center-mysql:latest`
大约 20 秒，可完成镜像推送

log-center-mysql

- 在 master 节点上，执行命令 `cd /root/open-capacity-platform/sql` 切换当前目录。（与上一个步骤目录相同）
- 执行命令 `vim log-center-my.cnf` 以创建文件，其内容如下：

```
1 | [mysqld]  
2 |  
3 | init_connect='SET collation_connection = utf8_unicode_ci'  
4 | init_connect='SET NAMES utf8'  
5 | character-set-server=utf8  
6 | collation-server=utf8_unicode_ci  
7 | skip-character-set-client-handshake
```

执行命令 `vim dockerfile_log-center` 以创建文件，其内容如下：

```
1 | FROM mysql:5.7.26  
2 | ADD log-center-my.cnf /etc/mysql/conf.d/my.cnf  
3 | ADD 05.log-center.sql /docker-entrypoint-initdb.d/05.log-center.sql  
4 | EXPOSE 3306
```

执行命令 `docker build -f dockerfile_log-center -t ocpsample/log-center-mysql:latest .`

执行命令 `docker push ocpsample/log-center-mysql:latest`

大约 20 秒，可完成镜像推送

部署mysql

部署auth-center-mysql

- 在 Kuboard 界面中进入 `ocp` 名称空间
- 点击 **创建工作负载** 按钮

填写表单，如下图所示：

字段名称	填写内容	说明
服务类型	StatefulSet	
服务分层	持久层	
服务名称	auth-center	
服务描述	认证中心数据库	
副本数量	1	请填写1
容器名称	auth-center-mysql	
镜像	ocpsample/auth-center-mysql:latest	
抓取策略	Always	
环境变量	MYSQL_ROOT_PASSWORD=root	参考 mysql官方镜像

Service	ClusterIP (集群内访问) 协议: TCP 服务端: 3306 容器端口: 3306	

持久化

- 将 `mysql` 的容器内路径 `/var/lib/mysql` 映射到外部数据卷，可以使数据持久保存，请参考 [数据卷](#)
- 为了保持教程的简洁，此处并没有为 `auth-center-mysql` 挂载外部存储，存入 `mysql` 的数据在每次容器重启后都将丢失，并重新执行初始化脚本 `02.auth-center.sql`

访问方式

为该 StatefulSet 配置了 ClusterIP (集群内访问) 的访问方式，Kuboard 将创建一个与 StatefulSet 同名 (db-auth-center) 的 Kubernetes Service。您可以在集群内同名称空间 `ocp` 下任何容器组中通过 `db-auth-center:3306` 访问 `auth-center-mysql` 数据库，用户名为 `root`，密码为 `root`。

参考 [Service连接应用程序](#)

Kuboard 设置 事件 后退 ocp ocp 创建工作负载 取消编辑

基本信息

* 服务类型: StatefulSet

* 服务分层: 持久层

* 服务名称: db- auth-center

* 标签: k8s.eip.work/layer: db
k8s.eip.work/name: db-auth-center
+ 标签

服务描述: 认证中心数据库 7/50

* 副本数量: 1

存储卷声明模板

添加

数据卷 Volume

帮助 添加

运行容器组 Pod

Docker 仓库的用户名密码 帮助 删除

请选择 删除

添加 创建

ServiceAccount 请选择

容器组重启策略 帮助 删除

Always

节点选择 帮助 删除

自动分配 指定节点 匹配节点

由 Kubernetes 根据各节点的运行时状态自动分配

工作容器 删除

* 容器名称: auth-center-mysql

* 镜像: ocp/sample/auth-center-mysql:latest

* 抓取策略: Always

Command 添加

Args 添加

环境变量: _ROOT_PASSWORD = 值 删除

+ 名值对 + 配置

挂载点 添加

就绪检查 编辑 无

存活检查 编辑 无

资源限制 帮助 删除

添加初始化容器 添加工作容器

访问方式 Service

不配置 ClusterIP (集群内访问) NodePort (VPC内访问)

协议	服务端口	容器端口	操作
TCP	3306	3306	删除

添加

互联网入口 Ingress

帮助 删除

互联网入口

为 web 应用或 接口网关配置互联网入口 (K8S Ingress), 以便用户可以通过互联网访问应用程序

Ingress

Ingress 将集群外部的 HTTP / HTTPS 请求路由到集群内部的 Service。

保存 取消

- 点击 **保存**
- 点击 **应用**
- 点击 **完成**

验证auth-center-mysql

- 在 Kuboard 中进入 `auth-center-mysql` 的终端界面, 执行如下命令:

```
1 mysql -uroot -proot
2 > show databases;
3 > use oauth-center;
4 > show tables;
```

可以验证，oauth-center 的数据库表结构已经完成初始化，输出结果如下图所示：

```
返回名称空间 切换到 /bin/sh ocp / db-auth-center-0 auth-center-mysql 控制台 已连接
root@db-auth-center-0:/# mysql -uroot -proot
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.26 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| oauth-center |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> use oauth-center;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_oauth-center |
+-----+
| oauth_client_details |
| sys_client_service |
| sys_gateway_routes |
| sys_service |
+-----+
4 rows in set (0.00 sec)

mysql>
```

部署user-center-mysql

按照同样的方式部署 user-center-mysql，因此，本章节不在截图，只将必要的步骤和参数进行罗列：

- 点击 **创建工作负载** 按钮
- 填写表单，如下表所示：

字段名称	填写内容	说明
服务类型	StatefulSet	
服务分层	持久层	
服务名称	user-center	
服务描述	用户中心数据库	
副本数量	1	请填写1
容器名称	user-center-mysql	
镜像	ocpsample/user-center-mysql:latest	
抓取策略	Always	
环境变量	MYSQL_ROOT_PASSWORD=root	参考 mysql官方镜像

Service	ClusterIP (集群内访问) 协议: TCP 服务端口: 3306 容器端口: 3306	

- 点击 **保存**
- 点击 **应用**

- 点击 **完成**

验证user-center-mysql

- 在 Kuboard 中进入 `user-center-mysql` 的终端界面，执行如下命令：

```
1 mysql -uroot -proot
2 > show databases;
3 > use user-center;
4 > show tables;
```

部署log-center-mysql

按照同样的方式部署 log-center-mysql，因此，本章节不在截图，只将必要的步骤和参数进行罗列：

- 点击 **创建工作负载** 按钮
- 填写表单，如下表所示：

字段名称	填写内容	说明
服务类型	StatefulSet	
服务分层	持久层	
服务名称	log-center	
服务描述	日志中心数据库	
副本数量	1	请填写1
容器名称	log-center-mysql	
镜像	ocpsample/log-center-mysql:latest	
抓取策略	Always	
环境变量	MYSQL_ROOT_PASSWORD=root	参考 mysql官方镜像

Service	ClusterIP (集群内访问) 协议: TCP 服务端: 3306 容器端口: 3306	

- 点击 **保存**
- 点击 **应用**
- 点击 **完成**

验证log-center-mysql

- 在 Kuboard 中进入 `log-center-mysql` 的终端界面，执行如下命令：

```
1 mysql -uroot -proot
2 > show databases;
3 > use log-center;
4 > show tables;
```

在K8S上部署redis

如 [在K8S上部署mysql](#) 所述，auth-server、user-center、api-gateway 都需要使用 redis 服务，本文描述如何使用 Kuboard 在 Kubernetes 上部署 redis。

本文将使用 [redis官方镜像](#)

进行部署。

部署redis

- 在 Kuboard 界面进入名称空间 `ocp`，点击 **创建工作负载** 按钮，并填写表单，如下图所示：

字段名称	填写内容	备注
服务类型	StatefulSet	
服务分层	中间件	
服务名称	redis	
服务描述	Redis缓存	
容器名称	redis	
镜像	redis:4.0.14	
抓取策略	Always	
Service	ClusterIP (集群内访问) 协议: TCP 端口: 6379 容器端口: 6379	

- 点击 **保存**
- 点击 **应用**
- 点击 **完成**

稍等片刻，即可完成 redis 的部署

检查redis

- 在 Kuboard 界面进入 `c1oud-redis-0` 容器组的日志界面，可查看到 redis 可用的输出信息，如下所示：

```
返回名称空间 ocp / cloud-redis-0 日志 已连接
l:C 28 Sep 05:25:07.295 # o000c000c000c Redis is starting o000c000c000c
l:C 28 Sep 05:25:07.296 # Redis version=4.0.14, bits=64, commit=00000000, modified=0, pid=1, just started
l:C 28 Sep 05:25:07.296 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
l:M 28 Sep 05:25:07.301 * Running mode=standalone, port=6379.
l:M 28 Sep 05:25:07.302 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
l:M 28 Sep 05:25:07.302 # Server initialized
l:M 28 Sep 05:25:07.302 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
l:M 28 Sep 05:25:07.302 * Ready to accept connections
```

初始行数 500 暂停追踪 自动追踪 Tab键可逐页查看

已在 K8S 上完成了 Redis 的部署

在K8S上部署auth-server

本文假设您已经完成了 [在Kubernetes上部署 Spring Cloud - OCP](#) 系列教程的前面部分，并已经完成了 eureka-server、auth-center-mysql、redis 在 K8S 上的部署。

理解auth-server

auth-server 是一个 Spring Boot 项目，其配置文件位于路径 `oauth-center/auth-server/src/main/resources`，该目录内容如下所示：

```
1 | └─ application.yml
2 | └─ bootstrap.yml
3 | └─ mybatis.cfg.xml
```

监听端口

参考 `bootstrap.yml` 的如下代码片段，auth-server 监听 8000 端口

```
1 | #端口
2 | server:
3 |   port: 8000
4 | # port: ${randomServerPort.value[8000,8000]} #随机端口
```

依赖项

auth-server 的部署依赖有：

- eureka-server
- mysql
- redis

上述依赖在教程的前面部分都已经完成部署。

- **eureka-server** 依赖项

参考 `bootstrap.yml` 的如下代码片段，`auth-server` 中默认配置的 `eureka-server` 的地址为 `http://127.0.0.1:1111/eureka`

```
1 #eureka client 配置
2 eureka:
3   client:
4     serviceUrl:
5       defaultZone: http://127.0.0.1:1111/eureka
6       #http://130.75.131.241:8761/eureka,http://130.75.131.248:8762/eureka
7       #http://134.224.249.33:1111/eureka/ 正式库
8       #http://134.224.249.33:1111/eureka/ 测试库
```

- **mysql** 依赖项

参考 `application.yml` 的如下代码片段，`auth-server` 中默认配置的 `mysql` 的连接参数如下：

```
1 spring:
2   session:
3     store-type: redis
4   datasource:
5     dynamic:
6       enable: true
7     druid:
8       # JDBC 配置(驱动类自动从url的mysql识别,数据源类型自动识别)
9     core:
10      url: jdbc:mysql://59.110.164.254:3306/oauth-center?
11      useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
12      username: root
13      password: root
14      driver-class-name: com.mysql.jdbc.Driver
15     log:
16      url: jdbc:mysql://59.110.164.254:3306/log-center?
17      useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
18      username: root
19      password: root
20      driver-class-name: com.mysql.jdbc.Driver
```

- **redis** 依赖项

参考 `application.yml` 的如下代码片段，`auth-server` 中默认配置的 `redis` 的连接参数如下：


```
1 | spring:
2 |   # . . . . .
3 |   redis:
4 |     ##### redis 单机版 start #####
5 |     host: 59.110.164.254
6 |     port: 6379
7 |     timeout: 6000
8 |     database: 3
```

确定部署方案

auth-server 为无状态服务，使用 Deployment 部署。

根据 [在K8S上部署eureka-server](#)、[在K8S上部署mysql](#)、[在K8S上部署redis](#) 的部署结果，我们应该通过环境变量覆盖 auth-server 的如下参数：

- eureka.client.serviceUrl.defaultZone

```
1 | http://cloud-eureka-0.cloud-
   | eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-
   | eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-
   | eureka.ocp.svc.cluster.local:1111/eureka
```

spring.datasource.druid.core.url

```
1 | jdbc:mysql://db-auth-center:3306/oauth-center?
   | useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

spring.datasource.druid.core.username

`root` 与默认配置相同

spring.datasource.druid.core.password

`root` 与默认配置相同

spring.datasource.druid.log.url

```
1 | jdbc:mysql://db-log-center:3306/log-center?
   | useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

spring.datasource.druid.log.username

`root` 与默认配置相同

spring.datasource.druid.log.password

`root` 与默认配置相同

spring.redis.host

1 | cloud-redis

spring.redis.port

6379 与默认配置相同

部署auth-server

- 在 Kubeboard 界面进入 `ocp` 名称空间，点击 **创建工作负载** 按钮，并填写表单，如下图所示：

字段名称	填写内容	备注
服务类型	Deployment	
服务分层	服务层	
服务名称	auth-server	
服务描述	认证中心	
副本数	1	
容器名称	auth-server	
镜像	ocpsample/auth-server:latest	
抓取策略	Always	
环境变量	<pre>eureka.client.serviceUrl.defaultZone=http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka spring.datasource.druid.core.url=jdbc:mysql://db-auth-center:3306/oauth-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false spring.datasource.druid.log.url=jdbc:mysql://db-log-center:3306/log-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false spring.redis.host=cloud-redis</pre>	填入 kubeboard 时 环境变量名后面不带 <code>=</code> 此处的内容与前面的mysql和redis的部署匹配，请谨慎修改
Service	ClusterIP (集群内访问) 协议 TCP 服务端 8000 容器端口 8000	
Ingress	域名 <code>svc-auth-server.ocp.demo.kubeboard.cn</code> URL / 服务端 8000	请使用您自己的 Ingress 域名

域名

- 该域名由 工作负载名.名称空间.集群名字.一级域名 组成，这种命名规则下，只需要将 *.demo.kuboard.cn 的域名解析指向集群 Ingress Controller 的地址就可以
- 在测试环境，为了更好地测试，才为 svc 配置 Ingress
- 关于 Ingress，请参考 [Ingress通过互联网访问您的应用](#)
- 服务层的服务通过 eureka-server 进行服务发现，因此，原则上不需要为 Spring Cloud 在服务层的 Pod 配置 Kubernetes Service，此处是为了配置 Ingress 才启用 Kubernetes Service

Kuboard 设置 事件 ← 后退
ocp 创建工作负载 取消编辑

基本信息

* 服务类型

* 服务分层

* 服务名称

* 标签 k8s.eip.work/layer: svc k8s.eip.work/name: svc-auth-server + 标签

服务描述 4/50

* 副本数量

数据卷 Volume 帮助

添加

运行容器组 Pod

Docker 仓库的用户名密码 帮助

请选择 删除

添加 创建

工作容器 删除

* 容器名称

* 镜像

* 抓取策略

Command 添加

Args 添加

环境变量

viceUrl.defaultZone =	<input type="text" value="值"/>	删除
:ource.druid.core.url =	<input type="text" value="值"/>	删除
source.druid.log.url =	<input type="text" value="值"/>	删除
spring.redis.host =	<input type="text" value="值"/>	删除

+ 名值对 + 配置

挂载点 添加

就绪检查 编辑 无

存活检查 编辑 无

资源限制 帮助

节点选择 帮助

自动分配
 指定节点
 匹配节点

由 Kubernetes 根据各节点的运行时状态自动分配

添加初始化容器
添加工作容器

访问方式 Service

不配置
 ClusterIP (集群内访问)
 NodePort (VPC内访问)

协议	服务端口	容器端口	操作
<input type="text" value="TCP"/>	<input type="text" value="8000"/>	<input type="text" value="8000"/>	删除

添加

互联网入口 Ingress 帮助

添加

标签 + 标签

* 域名 删除

HTTPS

* 路由配置

映射URL	服务端口	操作
<input type="text" value="/"/>	<input type="text" value="8000"/>	删除

添加

保存
取消

检查部署结果

- 在浏览器访问 <http://svc-auth-server.ocp.demo.kuboard.cn/swagger-ui.html>

[](<http://svc-auth-server.ocp.demo.kuboard.cn/swagger-ui.html>)

此处请使用您自己的 url

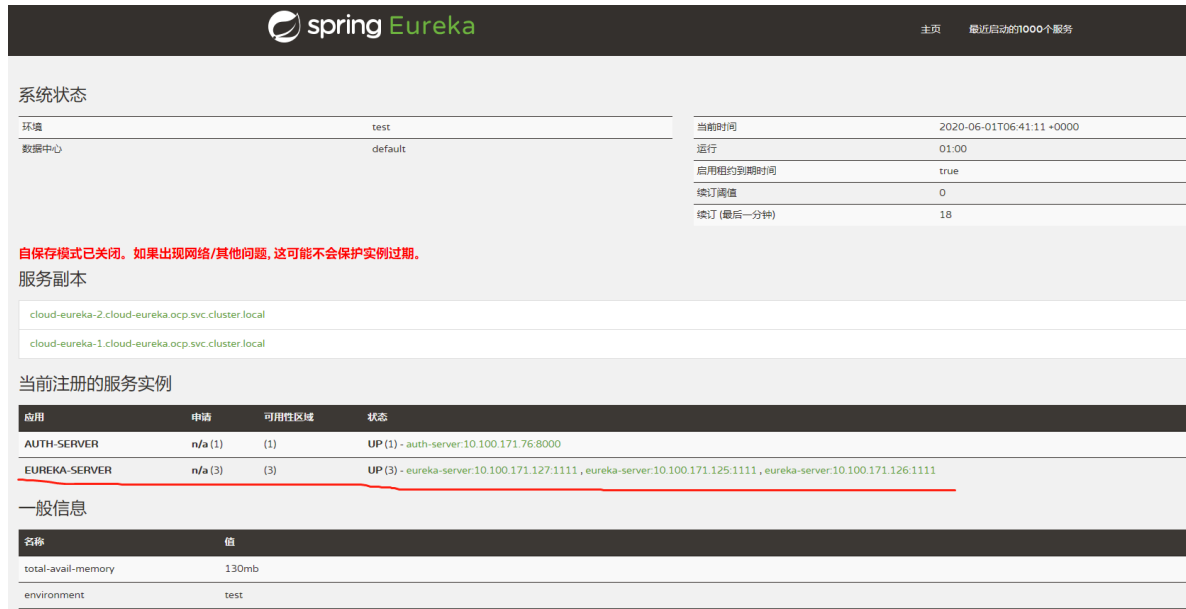


The image shows the Swagger UI for the authentication center. The title is "认证中心swagger接口文档" (Authentication Center Swagger API Documentation). The base URL is "http://svc-auth-server.ocp.demo.kuboard.cn/v2/api-docs". The API is titled "CLIENT API Sys Client Controller". The endpoints listed are:

- GET /clients: 根据用户id获取拥有的角色
- GET /clients/{id}: 根据id获取应用
- DELETE /clients/{id}: 删除应用
- GET /clients/all: 所有应用
- POST /clients/saveOrUpdate: 保存或者修改应用
- PUT /clients/updateEnabled: 修改状态

已在 K8S 上完成了 auth-server 的部署

若是无法通过域名访问的话，可以在eureka中查看是否已经注册到注册中心中



The image shows the Spring Eureka dashboard. The system status is "test" and the data center is "default". The dashboard displays the following information:

- 系统状态: 环境 (test), 数据中心 (default), 当前时间 (2020-06-01T06:41:11 +0000), 运行 (01:00), 启用租约到期时间 (true), 续订阈值 (0), 续订 (最后一分钟) (18).
- 自保存模式已关闭。如果出现网络/其他问题, 这可能不会保护实例过期。
- 服务副本: cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local, cloud-eureka-1.cloud-eureka.ocp.svc.cluster.local.
- 当前注册的服务实例: AUTH-SERVER (n/a (1) (1) UP (1) - auth-server:10.100.171.76:8000), EUREKA-SERVER (n/a (3) (3) UP (3) - eureka-server:10.100.171.127:1111, eureka-server:10.100.171.125:1111, eureka-server:10.100.171.126:1111).
- 一般信息: total-avail-memory (130mb), environment (test).

在K8S上部署user-center

本文假设您已经完成了 [在Kubernetes上部署 Spring Cloud - OCP](#) 系列教程的前面部分，并已经完成了 eureka-server、user-center-mysql、log-center-mysql、redis 在 K8S 上的部署。

理解user-center

user-center 是一个 Spring Boot 项目，其配置文件位于路径 `business-center/user-center/src/main/resources`，该目录内容如下所示：

```
1 | └─ application.yml
2 | └─ bootstrap.yml
3 | └─ mybatis.cfg.xml
```

监听端口

参考 `bootstrap.yml` 的如下代码片段，user-center 监听 7000 端口

```
1 | #端口配置
2 | server:
3 |   port: 7000 #固定端口
4 | # port: ${randomServerPort.value[7000,7005]} #随机端口
```

依赖项

user-center 的部署依赖有：

- eureka-server
- mysql
- redis

上述依赖在教程的前面部分都已经完成部署。

这些依赖项的情况与 [auth-server依赖项](#) 的情况大致相同，此处不再重复描述

确定部署方案

user-center 为无状态服务，使用 Deployment 部署。

根据 [在K8S上部署eureka-server](#)、[在K8S上部署mysql](#)、[在K8S上部署redis](#) 的部署结果，我们应该通过环境变量覆盖 user-center 的如下参数：

- eureka.client.serviceUrl.defaultZone

```
1 | http://cloud-eureka-0.cloud-
   | eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-
   | eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-
   | eureka.ocp.svc.cluster.local:1111/eureka
```

spring.datasource.druid.core.url

```
1 | jdbc:mysql://db-user-center:3306/user-center?
   | useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

spring.datasource.druid.core.username

root 与默认配置相同

spring.datasource.druid.core.password

root 与默认配置相同

spring.datasource.druid.log.url

```
1 jdbc:mysql://db-log-center:3306/log-center?  
  useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

- spring.datasource.druid.log.username

root 与默认配置相同

- spring.datasource.druid.log.password

root 与默认配置相同

- spring.redis.host

c1oud-redis

- spring.redis.port

6379 与默认配置相同

部署user-center

- 在 Kuboard 界面进入 ocp 名称空间，点击 **创建工作负载** 按钮，并填写表单，如下图所示：

字段名称	填写内容	备注
服务类型	Deployment	
服务分层	服务层	
服务名称	user-center	
服务描述	用户中心	
副本数	1	
容器名称	user-center	
镜像	ocpsample/user-center:latest	
抓取策略	Always	
环境变量	eureka.client.serviceUrl.defaultZone= http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka , http://cloud-eureka-1.cloud-eureka.ocp.svc.cluster.local:1111/eureka , http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka spring.datasource.druid.core.url=jdbc:mysql://db-user-center:3306/user-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false spring.datasource.druid.log.url=jdbc:mysql://db-log-center:3306/log-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false spring.redis.host=cloud-redis	填入 kubernetes 时 环境变量名后面不带 <code>=</code> 此处的内容与前面的mysql和redis的部署匹配, 请谨慎修改
Service	ClusterIP (集群内访问) 协议 TCP 服务端 7000 容器端口 7000	
Ingress	域名 <code>svc-user-center.ocp.demo.kubernetes.cn</code> URL / 服务端 7000	请使用您自己的 Ingress 域名

域名

- 该域名由 工作负载名.名称空间.集群名字.一级域名 组成, 这种命名规则下, 只需要将 `*.demo.kubernetes.cn` 的域名解析指向集群 Ingress Controller 的地址就可以
- 在测试环境, 为了更好地测试, 才为 svc 配置 Ingress
- 关于 Ingress, 请参考 [Ingress通过互联网访问您的应用](#)
- 服务层的服务通过 eureka-server 进行服务发现, 因此, 原则上不需要为 Spring Cloud 在服务层的 Pod 配置 Kubernetes Service, 此处是为了配置 Ingress 才启用 Kubernetes Service

Kuboard 设置 事件 ← 后退
ocp 创建工作负载 取消编辑

基本信息

* 服务类型 Deployment

* 服务分层 服务层

* 服务名称 svc- user-center

* 标签 k8s.eip.work/layer: svc k8s.eip.work/name: svc-user-center + 标签

服务描述 用户中心 4/50

* 副本数量 - 1 +

数据卷 Volume 帮助

添加

运行容器组 Pod

Docker 仓库的用户名密码 帮助

请选择 删除

添加 创建

工作容器 删除

* 容器名称 user-center

* 镜像 ocpsample/user-center:latest

* 抓取策略 Always

Command 添加

Args 添加

环境变量

viceUrl.defaultZone =	值	删除
iource.druid.core.url =	值	删除
isource.druid.log.url =	值	删除
spring.redis.host =	值	删除

+ 名值对 + 配置

挂载点 添加

就绪检查 编辑 无

存活检查 编辑 无

资源限制 帮助

添加初始化容器
添加工作容器

访问方式 Service

不配置 ClusterIP (集群内访问) NodePort (VPC内访问)

协议	服务端口	容器端口	操作
TCP	7000	7000	删除

添加

互联网入口 Ingress 帮助

标签 + 标签

* 域名 svc-user-center.ocp.demo.kuboard.cn 删除

HTTPS

* 路由配置

映射URL	服务端口	操作
/	7000	删除

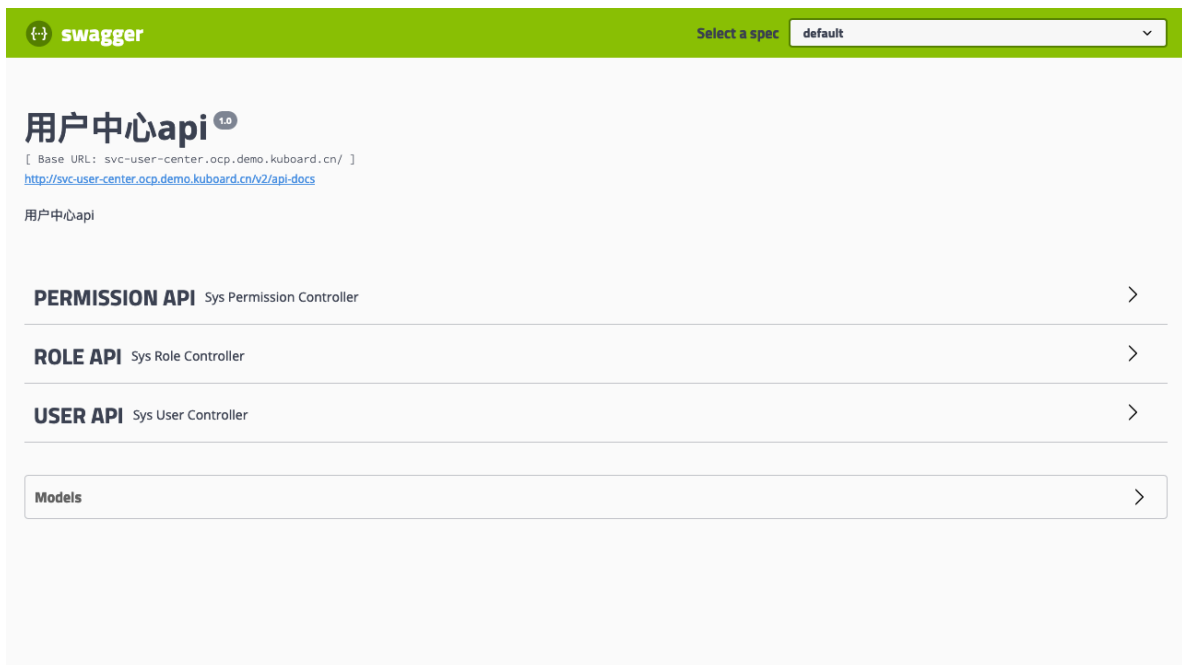
添加

保存
取消

检查部署结果

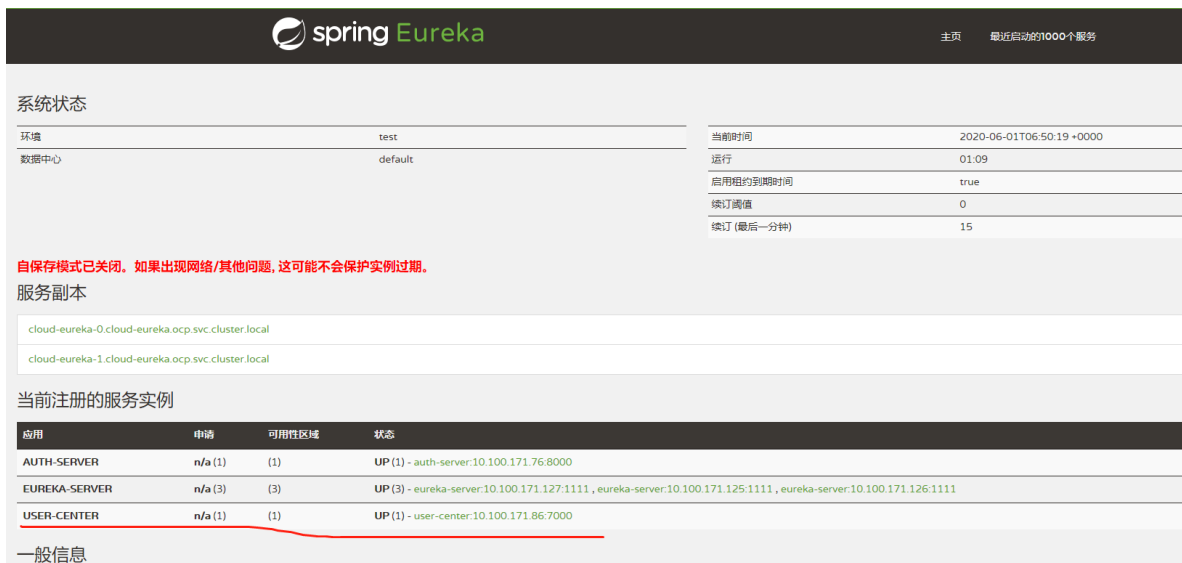
- 在浏览器访问 <http://svc-user-center.ocp.demo.kuboard.cn/swagger-ui.html>

此处请使用您自己的 url



已在 K8S 上完成了 user-center的部署

若是无法通过域名访问的话，可以在eureka中查看是否已经注册到注册中心中



在K8S上部署api-gateway

本文假设您已经完成了 [在Kubernetes 上部署 Spring Cloud - OCP](#) 系列教程的前面部分，并已经完成了 eureka-server、api-gateway-mysql、log-center-mysql、redis、auth-server、user-center 在 K8S 上的部署。

理解api-gateway

api-gateway 是一个 Spring Boot 项目，其配置文件位于路径 `api-gateway/src/main/resources`，该目录内容如下所示：

```
1 | application.yml
2 | bootstrap.yml
3 | mybatis.cfg.xml
```

监听端口

参考 `bootstrap.yml` 的如下代码片段，`api-gateway`监听 7000 端口

```
1 #端口配置
2 server:
3   port: 9200
```

依赖项

`api-gateway` 的部署依赖有：

- `eureka-server`
- `mysql`
- `redis`

上述依赖在教程的前面部分都已经完成部署。

这些依赖项的情况与 [auth-server依赖项](#) 的情况大致相同，此处不再重复描述

确定部署方案

`api-gateway` 为无状态服务，使用 `Deployment` 部署。

根据 [在K8S上部署eureka-server](#)、[在K8S上部署mysql](#)、[在K8S上部署redis](#) 的部署结果，我们应该通过环境变量覆盖 `api-gateway` 的如下参数：

- `eureka.client.serviceUrl.defaultZone`

```
1 http://cloud-eureka-0.cloud-
  eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-
  eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-
  eureka.ocp.svc.cluster.local:1111/eureka
```

`spring.datasource.druid.core.url`

```
1 jdbc:mysql://db-auth-center:3306/oauth-center?
  useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

`spring.datasource.druid.core.username`

`root` 与默认配置相同

`spring.datasource.druid.core.password`

`root` 与默认配置相同

`spring.datasource.druid.log.url`

```
1 jdbc:mysql://db-log-center:3306/log-center?  
useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

- spring.datasource.druid.log.username
root 与默认配置相同
- spring.datasource.druid.log.password
root 与默认配置相同
- spring.redis.host
cloud-redis
- spring.redis.port
6379 与默认配置相同

部署api-gateway

- 在 Kubeboard 界面进入 `ocp` 名称空间，点击 **创建工作负载** 按钮，并填写表单，如下图所示：

字段名称	填写内容	备注
服务类型	Deployment	
服务分层	网关层	
服务名称	api	
服务描述	接口网关	
副本数	1	
容器名称	api-gateway	
镜像	ocpsample/api-gateway:latest	
抓取策略	Always	
环境变量	eureka.client.serviceUrl.defaultZone= http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka , http://cloud-eureka-1.cloud-eureka.ocp.svc.cluster.local:1111/eureka , http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka spring.datasource.druid.core.url=jdbc:mysql://db-auth-center:3306/oauth-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false spring.datasource.druid.log.url=jdbc:mysql://db-log-center:3306/log-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false spring.redis.host=cloud-redis	填入 kubernetes 时 环境变量名后面不带 <code>=</code> 此处的内容与前面的mysql和redis的部署匹配, 请谨慎修改
Service	ClusterIP (集群内访问) 协议 TCP 服务端 9200 容器端口 9200	
Ingress	域名 <code>api-gateway.ocp.demo.kubernetes.cn</code> URL / 服务端 9200	请使用您自己的 Ingress 域名

域名

- 该域名由 工作负载名.名称空间.集群名字.一级域名 组成, 这种命名规则下, 只需要将 `*.demo.kubernetes.cn` 的域名解析指向集群 Ingress Controller 的地址就可以
- 关于 Ingress, 请参考 [Ingress通过互联网访问您的应用](#)

Kuboard 设置 ▾
ocp
ocp 创建工作负载 取消编辑

基本信息

- * 服务类型 ▾
Deployment
- * 服务分层 ▾
网关层
- * 服务名称 ▾
gateway- api
- * 标签

k8s.eip.work/layer: gateway
k8s.eip.work/name: gateway-api

+ 标签
- 服务描述 4/50
接口网关
- * 副本数量 ▾ 1 ▸

数据卷 Volume 帮助 ⓘ

添加

运行容器组 Pod

Docker 仓库的用户名密码 帮助 ⓘ

请选择 ▾ 删除

添加
创建

ServiceAccount

请选择 ▾

容器组重启策略 帮助 ⓘ

Always ▾

节点选择 帮助 ⓘ

自动分配
 指定节点
 匹配节点

由 Kubernetes 根据各节点的运行时状态自动分配

工作负载 删除

- * 容器名称 ▾
api-gateway
- * 镜像 ▾
ocpsample/api-gateway:latest
- * 抓取策略 ▾
Always

Command 添加

Args 添加

环境变量

viceUrl.defaultZone	=	值	删除
source.druid.core.url	=	值	删除
source.druid.log.url	=	值	删除
spring.redis.host	=	值	删除

+ 名值对
+ 配置

挂载点 添加

就绪检查 编辑 无

存活检查 编辑 无

资源限制 帮助 ⓘ

添加初始化容器
添加工作容器

访问方式 Service

不配置
 ClusterIP (集群内访问)
 NodePort (VPC内访问)

协议	服务端口	容器端口	操作
TCP ▾	9200	9200	删除

添加

互联网入口 Ingress 帮助 ⓘ

标签 + 标签

- * 域名 ▾ 删除
api-gateway.ocp.demo.kuboard.cn
- HTTPS
- * 路由配置

映射URL	服务端口	操作
/	9200 ▾	删除

添加

保存
取消

检查部署结果

在浏览器访问 <http://api-gateway.ocp.demo.kuboard.cn/doc.html>

此处请使用您自己的 url

已在 K8S 上完成了 api-gateway 的部署

若是无法通过域名访问的话，可以在eureka中查看是否已经注册到注册中心中

在K8S上部署back-center

本文假设您已经完成了 [在Kubernetes 上部署 Spring Cloud - OCP](#) 系列教程的前面部分，并已经完成了 eureka-server、user-center-mysql、log-center-mysql、redis、auth-server、user-center 在 K8S 上的部署。

理解back-center

back-server 是一个前端项目，OCP中，将其运行在一个 SpringBoot 中。

该项目中，与服务端连接的参数，请参考 `web-portal/back-center/src/main/view/static/module/config.js` 中的如下代码片段，OCP需要在前端项目中引用：

- `base_server` 即 api-gateway 的 URL 地址, `http://api-gateway.ocp.demo.kuboard.cn`
- `eureka_server` 即 eureka-server 的 URL 地址 `http://cloud-eureka.ocp.demo.kuboard.cn`
(经确认, 实际并没用到, 我们在此处仍然修改该取值)

依赖项

back-center 的部署依赖有:

- api-gateway

上述依赖在教程的前面部分都已经完成部署。

确定部署方案

back-center 为无状态服务, 使用 Deployment 部署。

前端项目中需要替换的两个参数都是 js 代码中的内容, 执行点是客户端浏览器, 因此, 不能够通过环境变量注入该参数。由于我们只需要 `web-portal/back-center/src/main/view/static` 目录下的静态内容, 且 js 中的变量不能像 java 一样, 通过环境变量覆盖, 此时, 我们使用 nginx 部署该项目显得更为合理一些。

因此, 我们为其构建一个 nginx 的 docker 镜像。

构建docker镜像

- 在 master 节点上执行本章节“构建docker镜像”的内容
- 创建文件 `/root/open-capacity-platform/web-portal/back-center/entry-point.sh`, 内容如下所示:

该文件在启动 nginx 前, 从环境变量获得参数, 并以此为依据, 使用 sed 命令修改 config.js 中对应字段的取值, 以便我们能够获得一个可以适应测试环境、生产环境部署的 docker 镜像。

```
1  #!/bin/sh
2  echo "GATEWAY_API_URL 为 ${GATEWAY_API_URL}"
3  echo "CLOUD_EUREKA_URL 为 ${CLOUD_EUREKA_URL}"
4  sed -i "s#base_server.*#base_server: '${GATEWAY_API_URL}',#g"
   /usr/share/nginx/html/module/config.js
5  sed -i "s#eureka_server.*#eureka_server: '${CLOUD_EUREKA_URL}',#g"
   /usr/share/nginx/html/module/config.js
6  echo "参数修改完毕, 如下所示: "
7  cat /usr/share/nginx/html/module/config.js
8  echo ""
9  echo "启动 nginx"
10 nginx -g "daemon off;"
```

创建文件 `/root/open-capacity-platform/web-portal/back-center/dockerfile`, 内容如下所示:

```

1 FROM nginx:1.17.1
2 LABEL maintainer="kuboard.cn"
3
4 ADD ./entry-point.sh /entry-point.sh
5 RUN chmod +x /entry-point.sh && rm -rf /usr/share/nginx/html
6
7 # 创建环境变量的默认内容，防止 sed 脚本出错
8 ENV GATEWAY_API_URL http://gateway_api_url_not_set/
9 ENV CLOUD_EUREKA_URL http://cloud_eureka_url_not_set/
10 ADD ./src/main/view/static /usr/share/nginx/html
11
12 EXPOSE 80
13 CMD ["/entry-point.sh"]

```

构建 docker 镜像并推送

```

1 # 切换到 back-center 目录
2 cd /root/open-capacity-platform/web-portal/back-center
3 # 使用你自己的 docker 用户名登录
4 docker login
5 # 构建镜像
6 docker build -t ocpsample/back-center:latest .
7 # 推送镜像
8 docker push ocpsample/back-center:latest

```

部署back-center

- 在 Kuboard 界面进入 `ocp` 名称空间，点击 **创建工作负载** 按钮，并填写表单，如下图所示：

字段名称	填写内容	备注
服务类型	Deployment	
服务分层	展现层	
服务名称	back-center	
服务描述	后台中心	
副本数	1	
容器名称	back-center	
镜像	ocpsample/back-center:latest	
抓取策略	Always	

字段名称	填写内容 GATEWAY_API_URL= http://api-gateway.ocp.demo.kuboard.cn/ CLOUD_EUREKA_URL= http://cloud-eureka.ocp.demo.kuboard.cn/	备注 kuboard 时 环境变量名后面 不带 =
Service	ClusterIP (集群内访问) 协议 TCP 服务端口 80 容器端口 80	
Ingress	域名 <code>back-center.ocp.demo.kuboard.cn/</code> URL / 服务端口 <code>80</code>	请使用您自己的 Ingress域名

域名

- 该域名由 工作负载名 . 名称空间 . 集群名字 . 一级域名 组成，这种命名规则下，只需要将 `*.demo.kuboard.cn` 的域名解析指向集群 Ingress Controller 的地址就可以
- 关于 Ingress，请参考 [Ingress通过互联网访问您的应用](#)

Kuboard 设置 事件 ← 后退
ocp 创建工作负载 取消编辑

基本信息

* 服务类型

* 服务分层

* 服务名称

* 标签 k8s.eip.work/layer: web k8s.eip.work/name: web-back-center

+ 标签

服务描述 4/50

* 副本数量

数据卷 Volume 帮助

添加

运行容器组 Pod

Docker 仓库的用户名密码 帮助

删除

添加 创建

ServiceAccount

容器组重启策略 帮助

节点选择 帮助

自动分配 指定节点 匹配节点

由 Kubernetes 根据各节点的运行时状态自动分配

工作容器 删除

* 容器名称

* 镜像

* 抓取策略

Command 添加

Args 添加

环境变量

GATEWAY_API_URL = 删除

LOUD_EUREKA_URL = 删除

+ 名值对 + 配置

挂载点 添加

就绪检查 编辑 无

存活检查 编辑 无

资源限制 帮助

添加初始化容器 添加工作容器

访问方式 Service

不配置 ClusterIP (集群内访问) NodePort (VPC内访问)

协议	服务端口	容器端口	操作
TCP	80	80	删除

添加

互联网入口 Ingress 帮助

标签 + 标签

* 域名 删除

HTTPS

* 路由配置

映射URL	服务端口	操作
/	80	删除

添加

保存
取消

检查部署结果

- 在浏览器访问 <http://back-center.ocp.demo.kuboard.cn/>

此处请使用您自己的 url

使用默认用户名密码 admin/admin 登录系统，可查看到登录后的结果如下所示：



这个在eureka中无法查看,但是可以通过其他办法查看

Kuboard 当前用户: kuboard-user 名称空间: [切换] ocp 设置

服务名称: ocp / web-back-center Deployment YAML

服务类型: Deployment 副本数量: 1 / 1

访问方式: Service Service YAML

类型: 集群内访问 [ClusterIP]

访问端口:

协议	服务端口	容器端口	访问
TCP	80	80	代理 提示

运行信息:

Type	Status	message
Available	True	Deployment has liability.
Progressing	True	ReplicaSet "web-65f59cbbc" has progressed.

互联网入口 Ingress Ingress YAML

域名: http://back-center.ocp.de

路由配置: 映射URL

Deployment Processing

通过 Kuboard 代理访问

代理配置信息 修改 了解代理

用户名添加到 Header: 未设置 Cookie TTL / 单位: 秒: 未设置

组名添加到 Header: 未设置 禁用 Rebase: false

代理目标信息:

proxy target kind: Service	proxy_set_header X-WEBAUTH-USER: kuboard-user
namespace: ocp	proxy_set_header X-WEBAUTH-GROUPS: kube-system
name: web-back-center	disable_rebase: false
ip: 10.96.7.72	有效时间: 36000 秒
port: 80	有效时间截止到: 2020-06-02 01:09:09

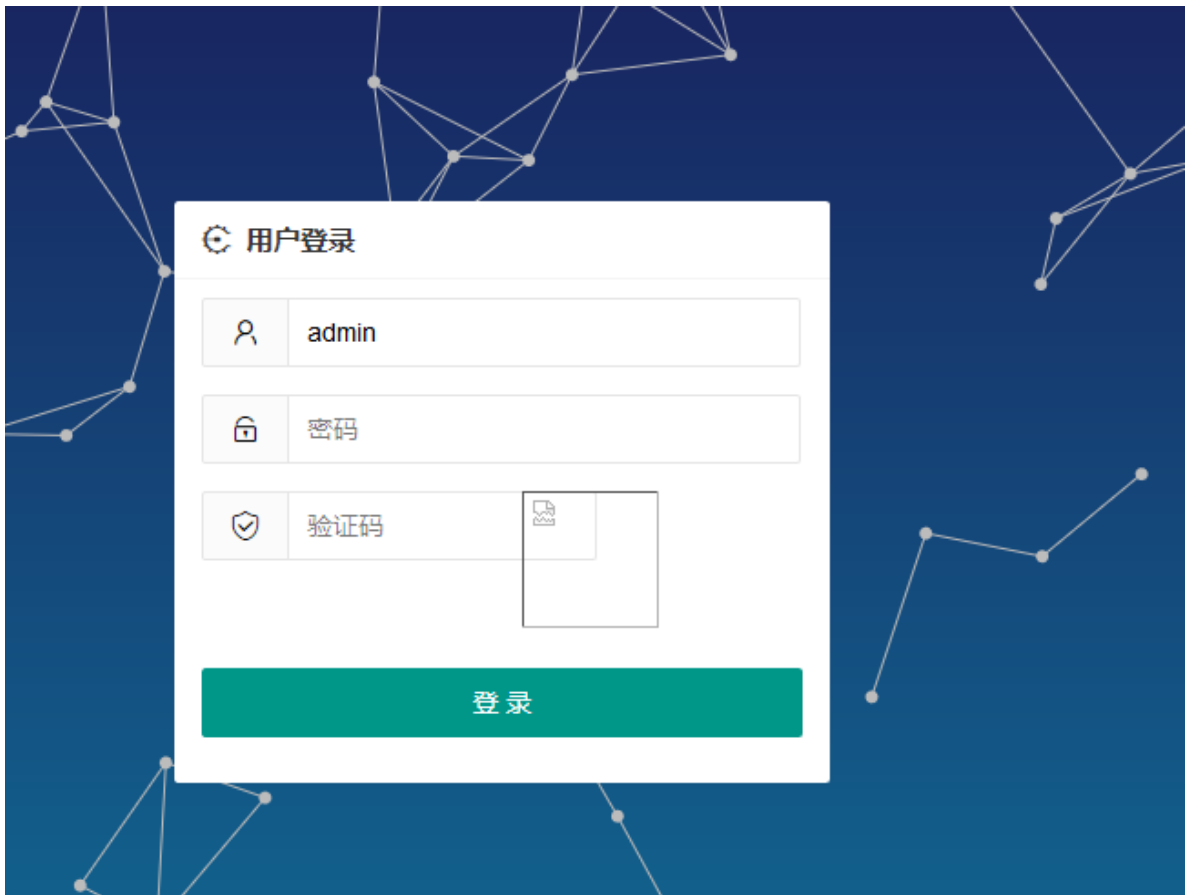
访问代理目标:

1 请选择代理协议 (容器内应用所接受的协议): http https

2 请选择访问方法: 在浏览器窗口中打开 使用 curl/postman 访问代理目标

您容器内对应端口必须能够处理 http 协议, 否则 KuboardProxy 将不能正常工作!

在 2020-06-02 01:09:09 之后, 或者浏览器 Session 失效之后, KuboardProxy 将拒绝您的请求。



重新审视配置信息

在本系列文章的前面部分，我们已经完成了在 Kubernetes 上部署 Spring Cloud OCP 的主要组件：eureka-server、auth-server、user-center、api-gateway、back-center。

提取相同的参数

各模块中的环境变量

部署过程中，主要使用环境变量向容器内注入具体环境相关的信息，以便容器内应用程序可以使用特定于环境的配置。具体来说，主要有如下信息通过环境变量替换：

- eureka-server
 - 使用环境变量覆盖

```
1 eureka.client.service-url.defaultZone
```

取值，将其设置为：

```
1 http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka
```

使用环境变量覆盖 `eureka.instance.prefer-ip-address` 取值，将其设置为： `false`

auth-server

部署auth-server时，通过环境变量覆盖了如下参数：

- eureka.client.serviceUrl.defaultZone 覆盖为

```
1 http://cloud-eureka-0.cloud-  
eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-  
eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-  
eureka.ocp.svc.cluster.local:1111/eureka
```

spring.datasource.druid.core.url 覆盖为

```
1 jdbc:mysql://db-auth-center:3306/oauth-center?  
useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

spring.datasource.druid.log.url 覆盖为

```
1 jdbc:mysql://db-log-center:3306/log-center?  
useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

- spring.redis.host 覆盖为 `cloud-redis`

user-center

部署user-center时，通过环境变量覆盖了如下参数：

- eureka.client.serviceUrl.defaultZone 覆盖为

```
1 http://cloud-eureka-0.cloud-  
eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-  
eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-  
eureka.ocp.svc.cluster.local:1111/eureka
```

spring.datasource.druid.core.url 覆盖为

```
1 jdbc:mysql://db-user-center:3306/user-center?  
useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

spring.datasource.druid.log.url 覆盖为

```
1 jdbc:mysql://db-log-center:3306/log-center?  
useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

- spring.redis.host 覆盖为 `ccloud-redis`

api-gateway

部署api-gateway时，通过环境变量覆盖了如下参数：

- eureka.client.serviceUrl.defaultZone 覆盖为

```
1 http://ccloud-eureka-0.ccloud-  
eureka.ocp.svc.cluster.local:1111/eureka,http://ccloud-eureka-1.ccloud-  
eureka.ocp.svc.cluster.local:1111/eureka,http://ccloud-eureka-2.ccloud-  
eureka.ocp.svc.cluster.local:1111/eureka
```

spring.datasource.druid.core.url 覆盖为

```
1 jdbc:mysql://db-auth-center:3306/oauth-center?  
useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

spring.datasource.druid.log.url 覆盖为

```
1 jdbc:mysql://db-log-center:3306/log-center?  
useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
```

- ○ spring.redis.host 覆盖为 `ccloud-redis`
- back-center
部署back-center时，通过环境变量覆盖了如下参数：
 - GATEWAY_API_URL
 - CLOUD_EUREKA_URL

相同的参数

回顾一下，可以发现如下相同的参数：

- eureka.client.service-url.defaultZone
使用到此参数，且取值相同的模块有：
 - eureka-server
 - auth-server
 - user-center
 - api-gateway
- spring.datasource.druid.core.url
使用到此参数的模块有，（但是他们的参数值不同）
 - auth-server
 - user-center
- spring.datasource.druid.log.url
使用到此参数，且取值相同的模块有：
 - auth-server

- user-center
- api-gateway
- spring.redis.host

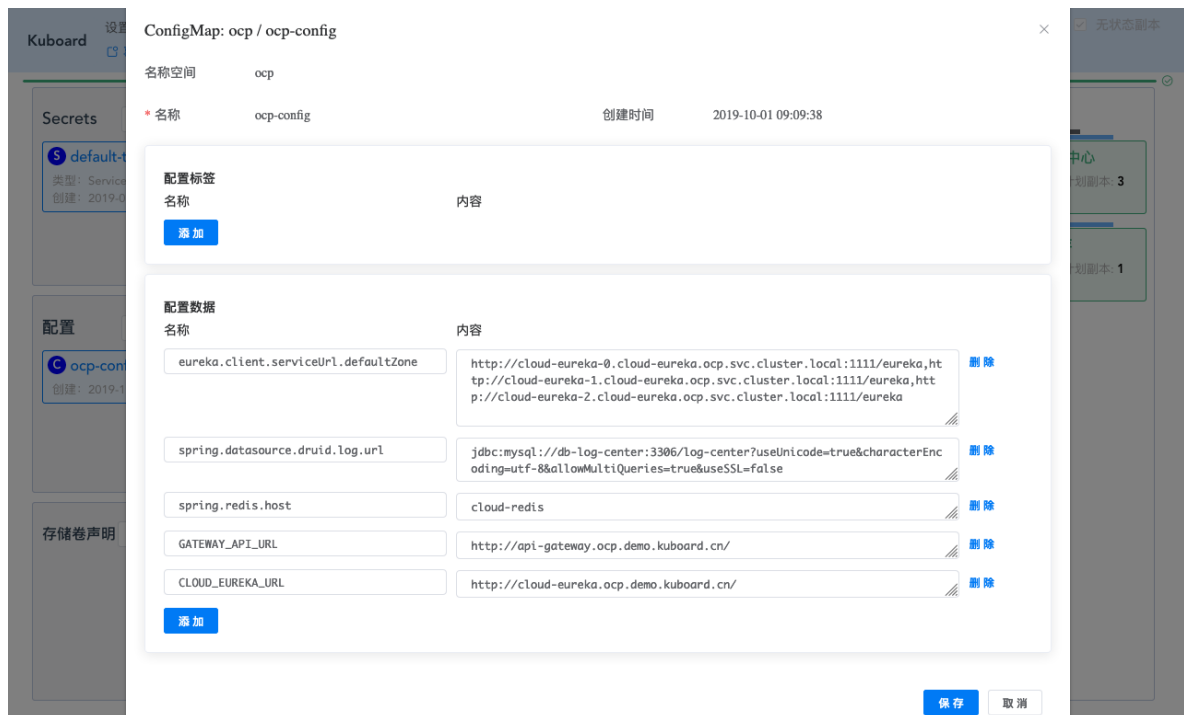
使用到此参数，且取值相同的模块有：

- auth-server
- user-center
- api-gateway

提取参数到ConfigMap

可参考文档 [使用ConfigMap配置您的应用程序](#)

- 在 Kuboard 界面中进入 `ocp` 名称空间
- 创建ConfigMap，并填入五个名值对：
 - `eureka.client.serviceUrl.defaultZone` = `http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-eureka.ocp.svc.cluster.local:1111/eureka`
 - `spring.datasource.druid.log.url` = `jdbc:mysql://db-log-center:3306/log-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false`
 - `spring.redis.host` = `cloud-redis`
 - `GATEWAY_API_URL` = `http://api-gateway.ocp.demo.kuboard.cn/`
 - `CLOUD_EUREKA_URL` = `http://cloud-eureka.ocp.demo.kuboard.cn/`



- 修改 eureka-server、auth-server、user-center、api-gateway 的部署信息，将上面创建的 ConfigMap 中所有名值对注入到容器的环境变量，并去除已经在 ConfigMap 中包含的环境变量。

以 eureka-server 为例，编辑该工作负载的界面截图如下所示：

工作容器
删除

* 容器名称

* 镜像

抓取策略

工作目录

Command

Args

Ports

环境变量 删除

+ 名值对
+ 配置

挂载点

postStart 无

preStop 无

就绪检查 无

存活检查 无

安全设定

CPU

TIP

将相同的参数提炼到 ConfigMap 可以使配置更简洁。

为谁定义变量

在本教程中，为了避免对 OCP 已有代码的修改，因此以直接注入 `spring.datasource.druid.log.url` 类似的环境变量的方式，使docker镜像适应不同的环境（开发环境、测试环境、生产环境等）。这种做法就会碰到一个比较尴尬的情况，例如，对于参数 `spring.datasource.druid.core.url` 键值相同，而不同模块中（auth-server、user-center、api-gateway）取值却不同。这是从参数使用者视角来看不可避免的现象。

一个建议的方式是，从参数提供者的视角来定义环境变量参数，并由参数使用者引用。例如，我们定义如下几个ConfigMap属性：

- EUREKA_URLS = `http://cloud-eureka-0.cloud-eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-`

- ```
eureka.ocp.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-
eureka.ocp.svc.cluster.local:1111/eureka
```
- DB\_AUTH\_CENTER\_URL = jdbc:mysql://db-auth-center:3306/auth-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
  - DB\_USER\_CENTER\_URL = jdbc:mysql://db-user-center:3306/user-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
  - DB\_LOG\_CENTER\_URL = jdbc:mysql://db-log-center:3306/log-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false
  - REDIS\_HOST = cloud-redis
  - GATEWAY\_API\_URL = http://api-gateway.ocp.demo.kuboard.cn/
  - CLOUD\_EUREKA\_URL = http://cloud-eureka.ocp.demo.kuboard.cn/

然后在参数使用者的 `application.xml` 中引用这些环境变量参数，以 `auth-center` 的 `application.xml` 为例：

```
1 spring:
2 session:
3 store-type: redis
4 datasource:
5 dynamic:
6 enable: true
7 druid:
8 # JDBC 配置(驱动类自动从url的mysql识别,数据源类型自动识别)
9 core:
10 url: ${DB_AUTH_CENTER_URL}
11 username: root
12 password: root
13 driver-class-name: com.mysql.jdbc.Driver
14 log:
15 url: ${DB_LOG_CENTER_URL}
16 username: root
17 password: root
18
19 redis:
20 host: ${REDIS_HOST}
21 port: 6379
22 timeout: 6000
```

## 导出部署配置

通过本系列文章前面的部分，我们终于完整了 Spring Cloud OCP 的核心组件在 Kubernetes 上的部署。此时，Kuboard 的名称空间 `ocp` 界面的截图如下所示：



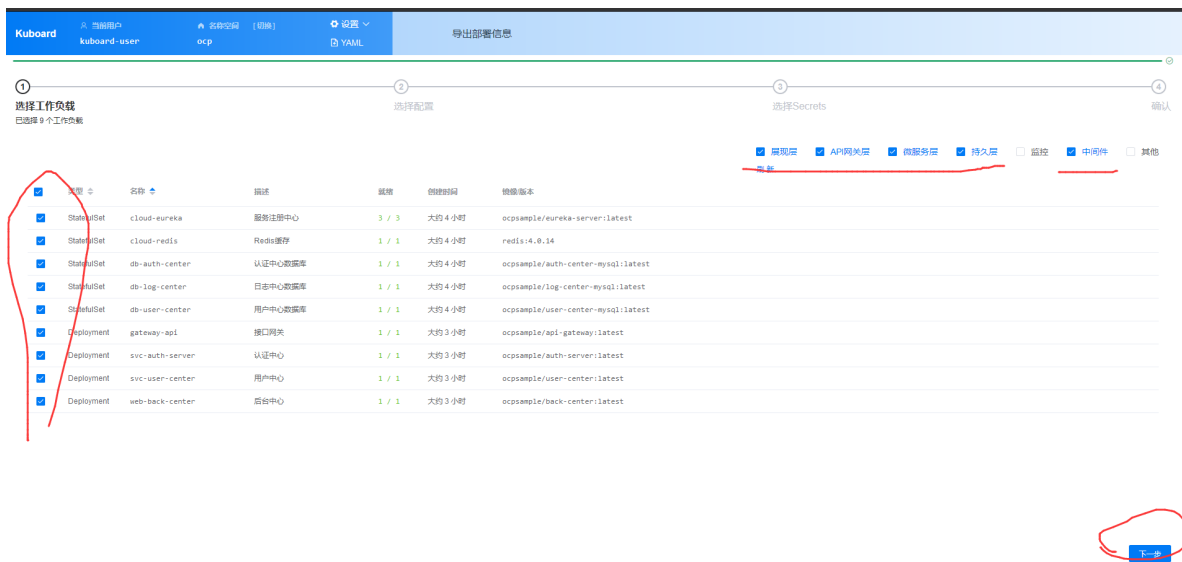
接下来，立刻要面临的问题是：假设刚部署完的环境是用于开发的，那我们如何快速部署一个新的环境用于测试呢？

Kuboard 支持您将此名称空间界面导出到一个 yam1 文件，并在另一个名称空间（或Kubernetes集群）中导入所有的配置。本文档后面的部分将描述导出配置的过程：

- 点击 **导出工作负载** 按钮

选择 **展现层**、**网关层**、**服务层**、**持久层**、**中间件** 五个分层，点击刷新，然后全选所有的工作负载，如下图所示：





然后下一步，最后点击 **确定** 按钮

并保存文件，文件名格式为 `kuboard_名称空间_年_月_日_时_分_秒.yaml`，例如 `kuboard_ocp_2019_10_01_12_56_14.yaml`

## 导入部署配置

腾讯云 [【腾讯云】云产品采购季，助力行业复工。](#) 1核2G云服务器，首年99元 [去抢](#) [广告](#)

### # 获得yaml文件

在 [导出部署配置](#) 中，我们将 Spring Cloud OCP 部署相关的所有信息导出到一个 yaml 文件，本文描述如何在一个新的名称空间（或者新的 Kubernetes 集群）中导入该部署信息，并形成一个新的独立的部署环境。

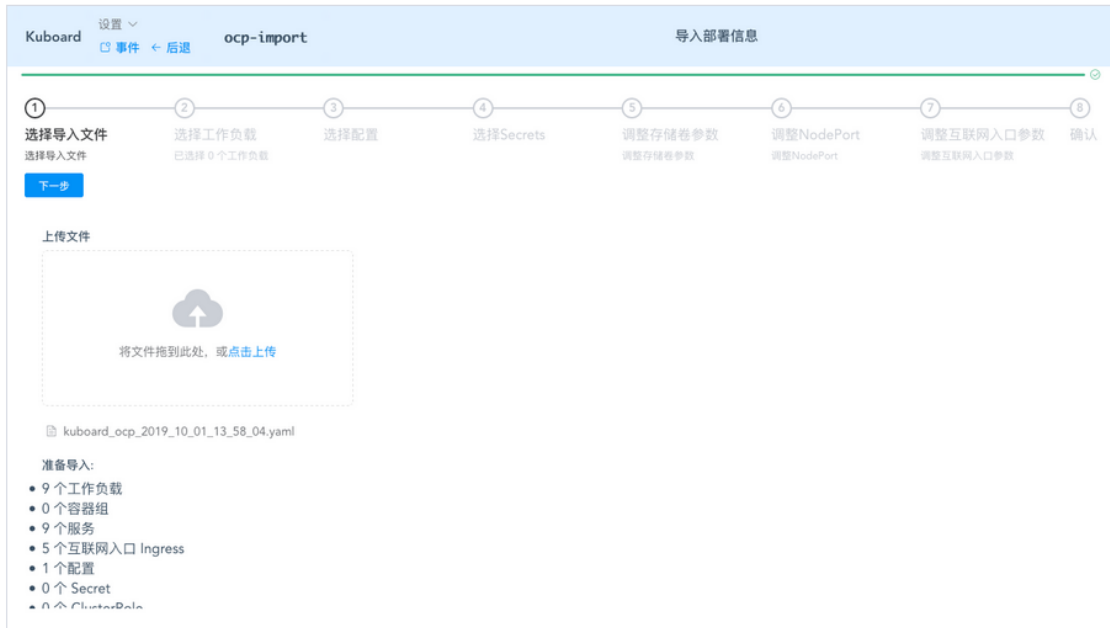
如果您还没有该 yaml 文件，可从此处 [下载 OCP部署yaml文件](#)

### # 导入yaml文件

假设您已创建了名称空间 `ocp-import` 用来导入部署配置。请参考接下来的导入步骤：

- 点击 **导入工作负载** 按钮

在此界面中上传前一个步骤导出的（或从 [www.kuboard.cn](http://www.kuboard.cn) 下载的）yaml 文件。如下图所示：



- 点击 **下一步** 按钮

全选所有的工作负载，如下图所示：

- 点击 **下一步** 按钮

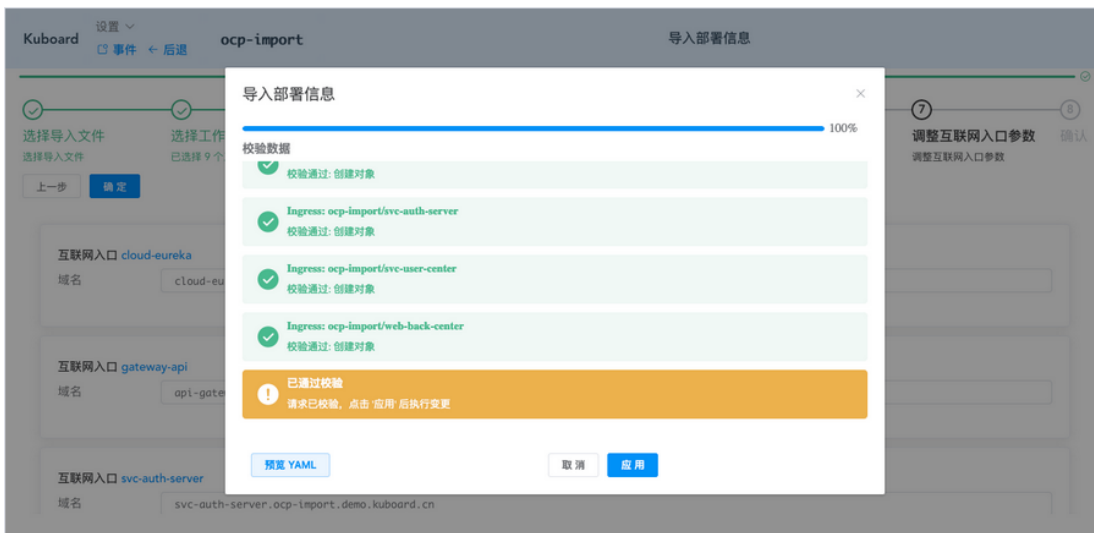
勾选 **使用随机端口** 此处由于我们在原集群的新名称空间中导入配置，因此，需要修改节点端口号，以避免冲突。如果您不知道该怎么分配端口号，可以在节点端口字段填写 **0**，集群将自动为您分配可用的节点端口。



- 点击 **下一步** 按钮

在此界面中调整对外的域名，以避免和原名称空间的部署产生域名冲突，如下图所示：

- cloud-eureka.ocp-import.demo.kuboard.cn
- api-gateway.ocp-import.demo.kuboard.cn
- svc-auth-server.ocp-import.demo.kuboard.cn
- svc-user-center.ocp-import.demo.kuboard.cn
- back-center.ocp-import.demo.kuboard.cn



点击 **应用** 按钮



- 点击 **完成** 按钮
- 此时进入名称空间，可看到所有的配置和部署都已经完成导入。

## # 导入后调整

- 导入到新的名称空间以后，ConfigMap中配置参数受到影响的部分需要调整，如下图所示：
  - `eureka.client.serviceUrl.defaultZone = http://cloud-eureka-0.cloud-eureka.ocp-import.svc.cluster.local:1111/eureka,http://cloud-eureka-1.cloud-eureka.ocp-import.svc.cluster.local:1111/eureka,http://cloud-eureka-2.cloud-eureka.ocp-import.svc.cluster.local:1111/eureka`
  - `spring.datasource.druid.log.url = jdbc:mysql://db-log-center:3306/log-center?useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true&useSSL=false`
  - `spring.redis.host = cloud-redis`
  - `GATEWAY_API_URL = http://api-gateway.ocp-import.demo.kuboard.cn`
  - `CLOUD_EUREKA_URL = http://cloud-eureka.ocp-import.demo.kuboard.cn`

由于 ConfigMap 中的参数发生了变化，此时必须删除所以引用该 ConfigMap 中的容器组，Kubernetes 将自动创建新的容器组以替换被删除的容器组，新的容器组中，ConfigMap 的变更将生效。

点击名称空间上方的 **容器组列表** 按钮，全选，并删除，如下图所示：




Kuboard 设置 o 事件 ← 后退 **ocp-import** 容器组列表  展现层  网关层  服务层  持久层  中间件  监控层  其他 刷新

删除

| <input checked="" type="checkbox"/> | 名称 <span>↕</span>                                                                                                  | 就绪    | 主机                                      | 节点              | 状态      | 已重启 | 已创建 <span>↕</span> |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------|-------|-----------------------------------------|-----------------|---------|-----|--------------------|
| <input checked="" type="checkbox"/> |  cloud-eureka-0                   | 1 / 1 | 172.17.216.105 / demo-worker-temp-01    | 192.168.199.163 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  cloud-redis-0                    | 1 / 1 | 172.17.216.105 / demo-worker-temp-01    | 192.168.199.164 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  db-auth-center-0                 | 1 / 1 | 172.17.76.199 / iz2zedgghv14j4zfl06z3lz | 192.168.144.131 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  db-log-center-0                  | 1 / 1 | 172.17.76.199 / iz2zedgghv14j4zfl06z3lz | 192.168.144.176 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  db-user-center-0                 | 1 / 1 | 172.17.76.199 / iz2zedgghv14j4zfl06z3lz | 192.168.144.145 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  gateway-api-5db4fb564d-z16gb     | 1 / 1 | 172.17.216.105 / demo-worker-temp-01    | 192.168.199.165 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  svc-auth-server-547cf78d78-1fpvh | 1 / 1 | 172.17.76.199 / iz2zedgghv14j4zfl06z3lz | 192.168.144.149 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  svc-user-center-65cdf75b-ln4db   | 1 / 1 | 172.17.216.105 / demo-worker-temp-01    | 192.168.199.166 | running | 0   | 3 分钟               |
| <input checked="" type="checkbox"/> |  web-back-center-bc58d86c6-rqvh5  | 1 / 1 | 172.17.216.105 / demo-worker-temp-01    | 192.168.199.167 | running | 0   | 3 分钟               |

## 验证配置

- 在浏览器打开 <http://back-end.ocp-import.demo.kuboard.cn> (此域名已失效, 以节省演示服务器空间, 请使用您自己的域名), 可登录 OCP 后台中心的界面。

   您成功学会了如何使用 Kuboard 快速复制一份 Spring Cloud 微服务架构的部署环境。