该笔记主要内容为本周阅读k8s文档及Spring cloud相关资料后，结合本周的实践结果对所学内容进行的简要整理。

1 k8s学习笔记

1.1 简述

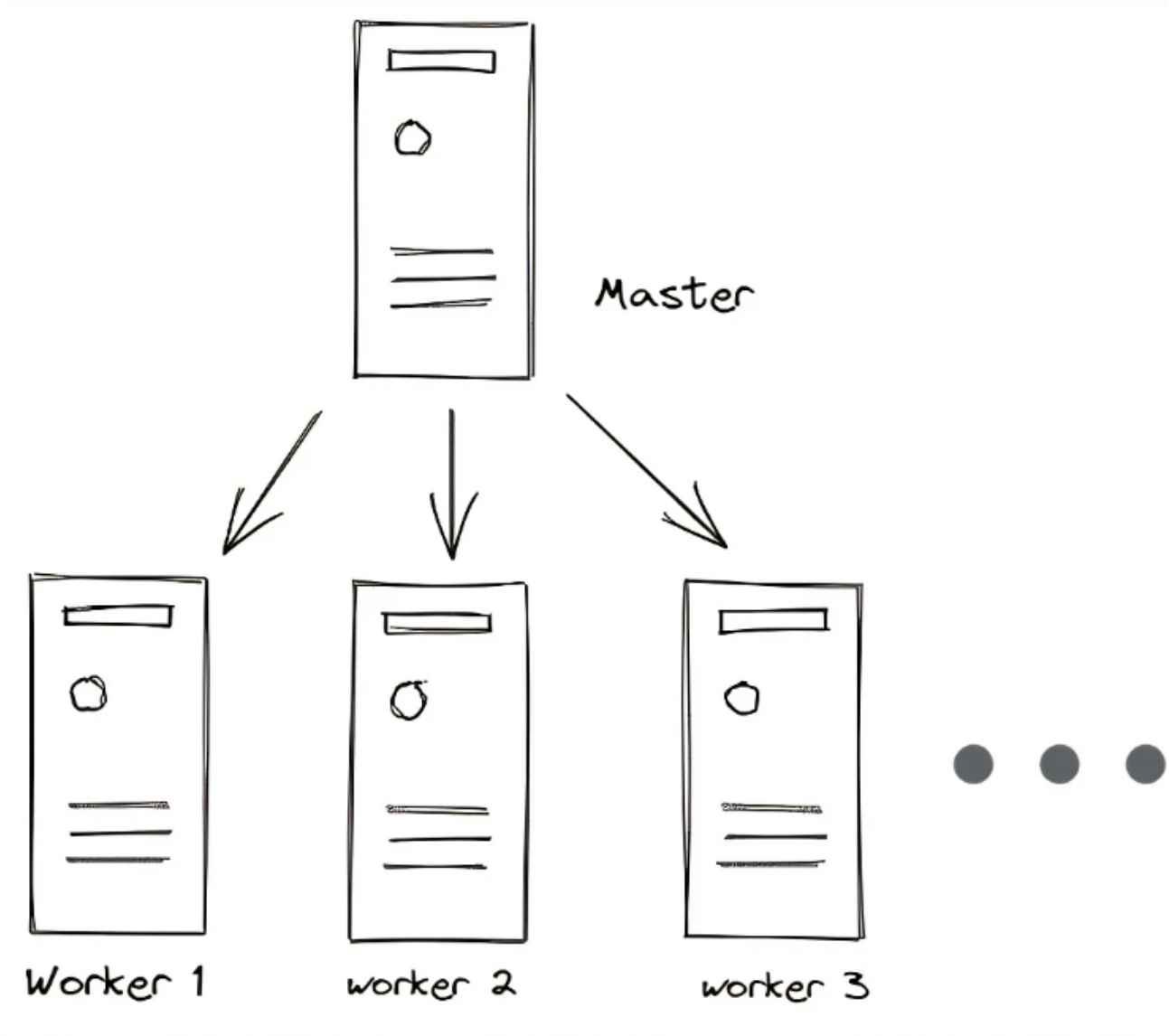
简而言之，k8s是一个对众多机器进行分布式管理的开源工具，能够实现服务发现；负载均衡；自动部署与回滚；自我修复（在实现过程中，我发现k8s确实在不断的重启和修复启动失败的pod，而且这一过程不需要人为干预）等操作。

个人体验相比Spring cloud这种每个小功能都要自己配置组件的框架来讲，k8s确实使用起来更加方便，使开发人员可以更多关心功能本身的开发工作上。而且k8s无疑能为大公司省下更多的运维成本，节约服务器资源。

1.2 架构

1.2.1 node

k8s架构主要由节点（node）构成，节点又分为主节点（master）和工作节点（worker），其关系如下图所示：



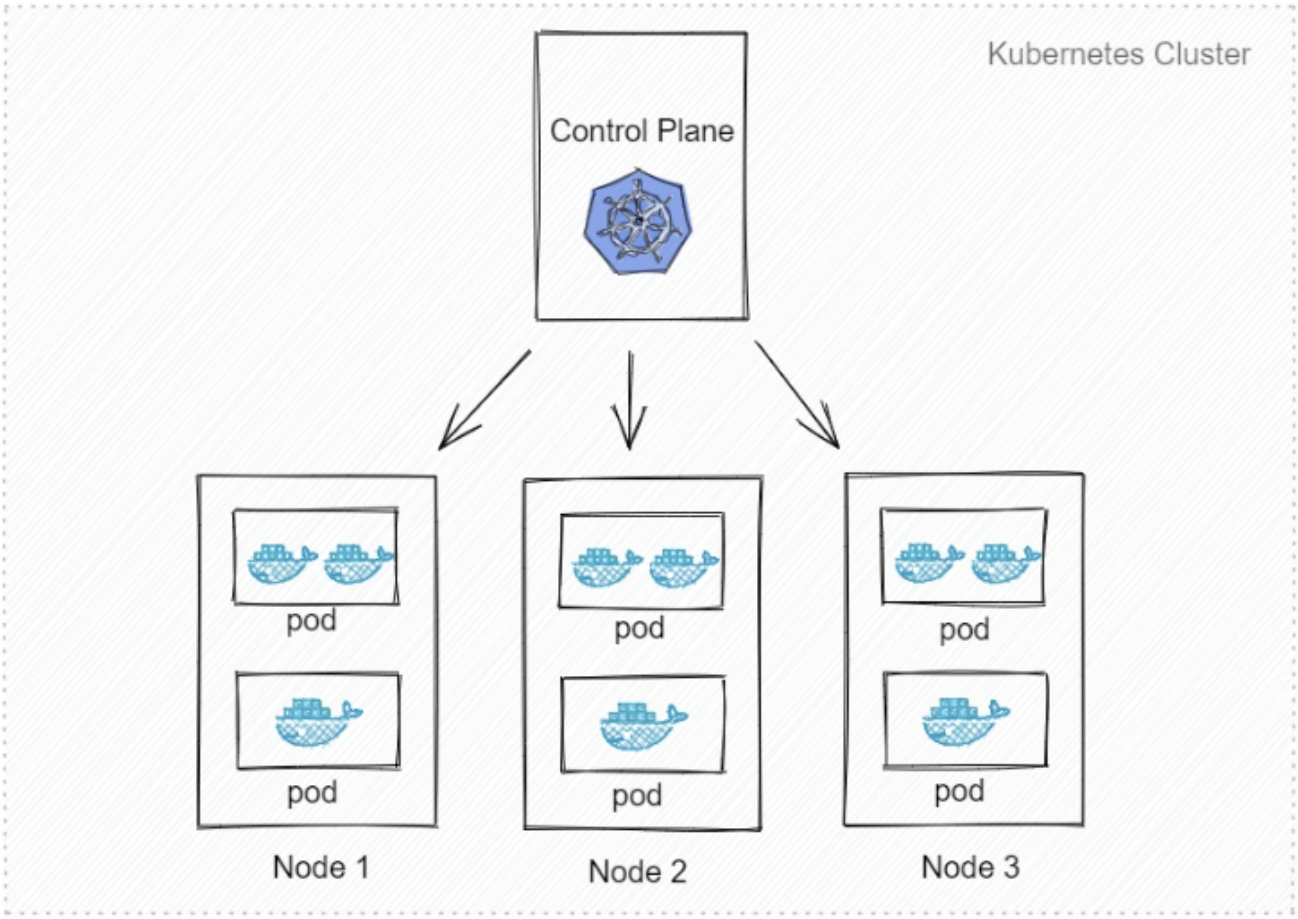
其中，master作为主节点起到控制平台的作用，用来管理手下的worker，进行负载均衡等操作。如果需要多集群分布式管理，也可以用多个master来管理不同的集群。 worker作为工作节点，可以是虚拟机/实体机器，用来跑任务，由master进行任务分配。

1.2.2 pod

pod是封装在node里的调度单元，是k8s进行调度和管理的最小单位。一个pod包含多个容器，主节点会考量自动调度pod到哪个节点运行。

同时，每个pod有自己的独立虚拟ip，因此，pod里的容器可以共享pod的存储空间和网络资源，个人理解类似于进程和线程的关系。

node,pod和容器的关系如下图所示：



1.3 使用minikube进行架构层面的实践与观察

需要说明的是，minikube只有一个节点，且该节点集合了master和worker的功能。所以使用minikube进行的本地实践只能观察到k8s在pod层面的调度和运行情况，后续我会考虑本地多开虚拟机（感觉好麻烦）/云服务器租赁多集群的方式进行节点层面的实践。

1.3.1 安装minikube

minikube安装：

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

快速使用：

```
minikube start
```

停止服务：

```
minikube stop
```

删除minikube镜像：

```
minikube delete
```

版本验证：

```
minikube version
```

1.3.2 安装kubectl

kubectl是kube的命令管理工具，使用kube绕不开这个工具的安装。

下载安装包：

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

检查校验和：

```
curl -LO "https://dl.k8s.io/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
```

验证：

```
echo "$(cat kubectl.sha256) kubectl" | sha256sum --check

kubectl version --client
```

1.3.3 利用minikube部署sock shop

将sock shop文件clone到本地：

```
git clone https://github.com/microservices-demo/microservices-demo
cd microservices-demo
```

开启minikube：

```
minikube start
```

(可选)部署dashboard监控微服务的运行情况：

```
minikube dashboard
```

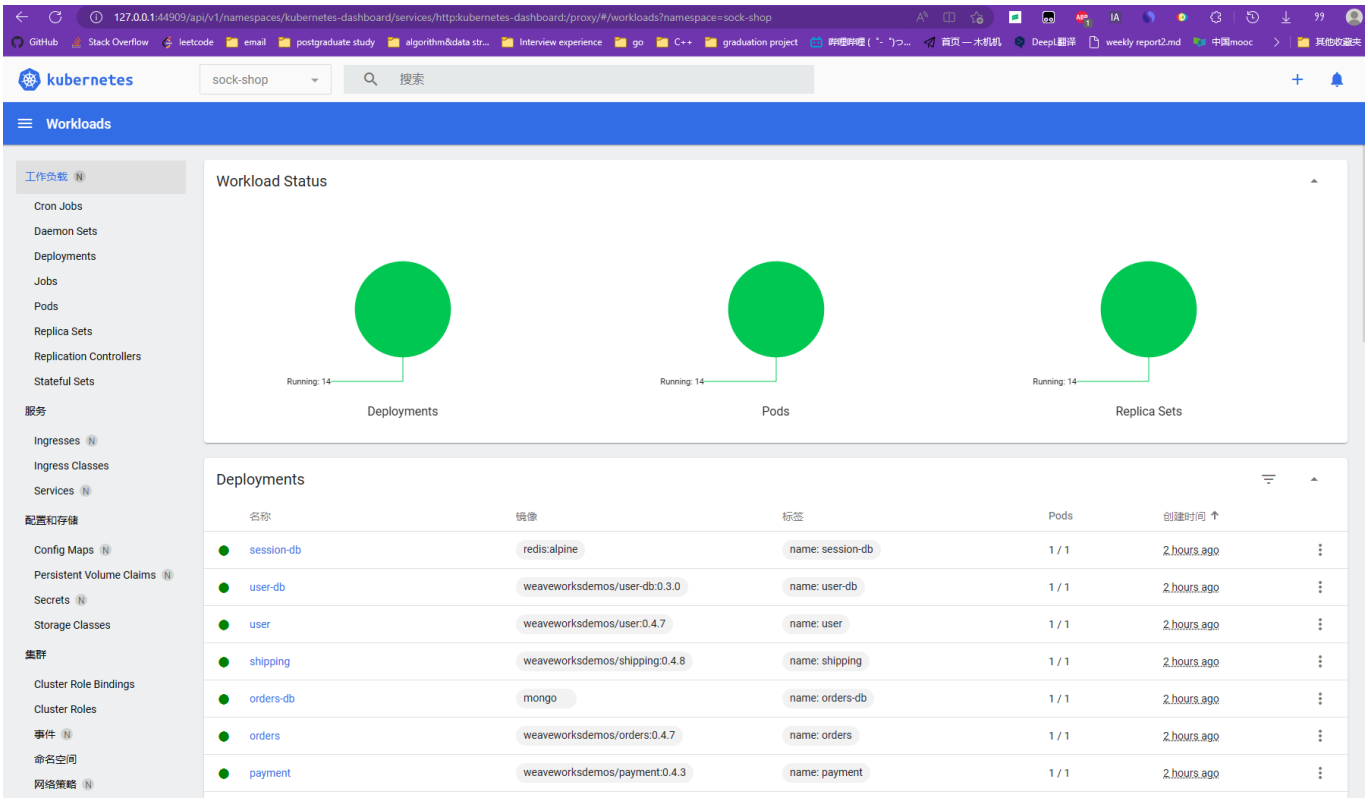
- Tips:这一步要在新的终端中运行并且始终挂载

部署Sock Shop：

```
kubectl create -f deploy/kubernetes/manifests/00-sock-shop-ns.yaml -f
deploy/kubernetes/manifests
```

查看微服务的运行情况：

- 在dashboard的页面中直接查看

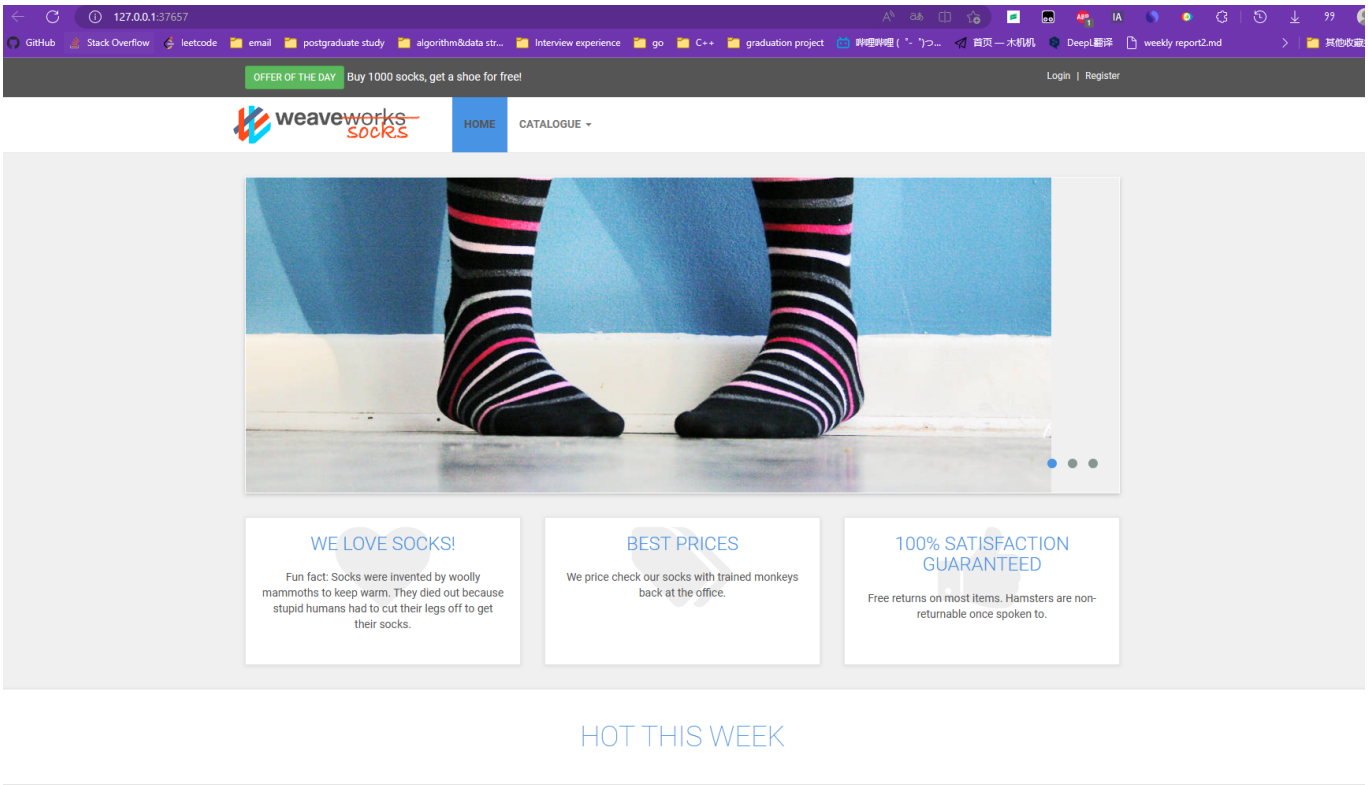


- 或者使用命令`kubectl get pods --namespace="sock-shop"`观察每一个pod的运行状态

```
master@LAPTOP-BTG77E4C:~/microservices-demo$ kubectl get pods --namespace="sock-shop"
NAME                            READY   STATUS             RESTARTS   AGE
carts-db-9f479f97d-mxjcr        0/1     ImagePullBackOff   0          14m
carts-f97d4b77b-hhp7c           1/1     Running            0          14m
catalogue-b879d7855-5tkkc       1/1     Running            0          14m
catalogue-db-6f8f687666-pfrp2   1/1     Running            0          14m
front-end-64c558644d-trt8q      1/1     Running            0          14m
orders-975b98f6d-n2rsr          1/1     Running            0          14m
orders-db-7b9776477b-84r8l      0/1     ImagePullBackOff   0          14m
payment-5479985fc6-j8c9g        1/1     Running            0          14m
queue-master-5f47c5967-crwbx    1/1     Running            0          14m
rabbitmq-7d58795496-tpkhc       0/2     ContainerCreating  0          14m
session-db-6b78bc9658-rmm6m     1/1     Running            0          14m
shipping-b7967ddb-blr27         0/1     ContainerCreating  0          14m
user-5c8564b446-5t5ls           0/1     ContainerCreating  0          14m
user-db-6979dbc4cd-t62ql        0/1     ContainerCreating  0          14m
```

访问Sock Shop前端页面：

```
minikube service front-end -n sock-shop
```



1.4 遇到的问题：

- Sock Shop是部署在“sock shop”命名空间中而不是部署在默认的命名空间中，因此在执行各种操作命令的时候要注意加上sock shop命名空间（在dashboard监控部署状态时也要注意切换到对应的命名空间）。刚开始入门时不了解命名空间的概念，浪费了很多时间
- 部署Sock Shop时官网给出的yaml文件地址不存在，要自己找一下对应的文件地址
- Sock Shop界面并不是部署在官网所说的位置，你需要使用`minikube service`指令（后面记得用`-n sock-shop`指定命名空间）运行你的服务

1.5 命名空间(namespace)

我在实践过程中发现一般的微服务应用在部署时都会给自己声明一个命名空间（namespace），用一个个命名空间来隔绝不同的微服务应用。因此也在这里做一个补充。

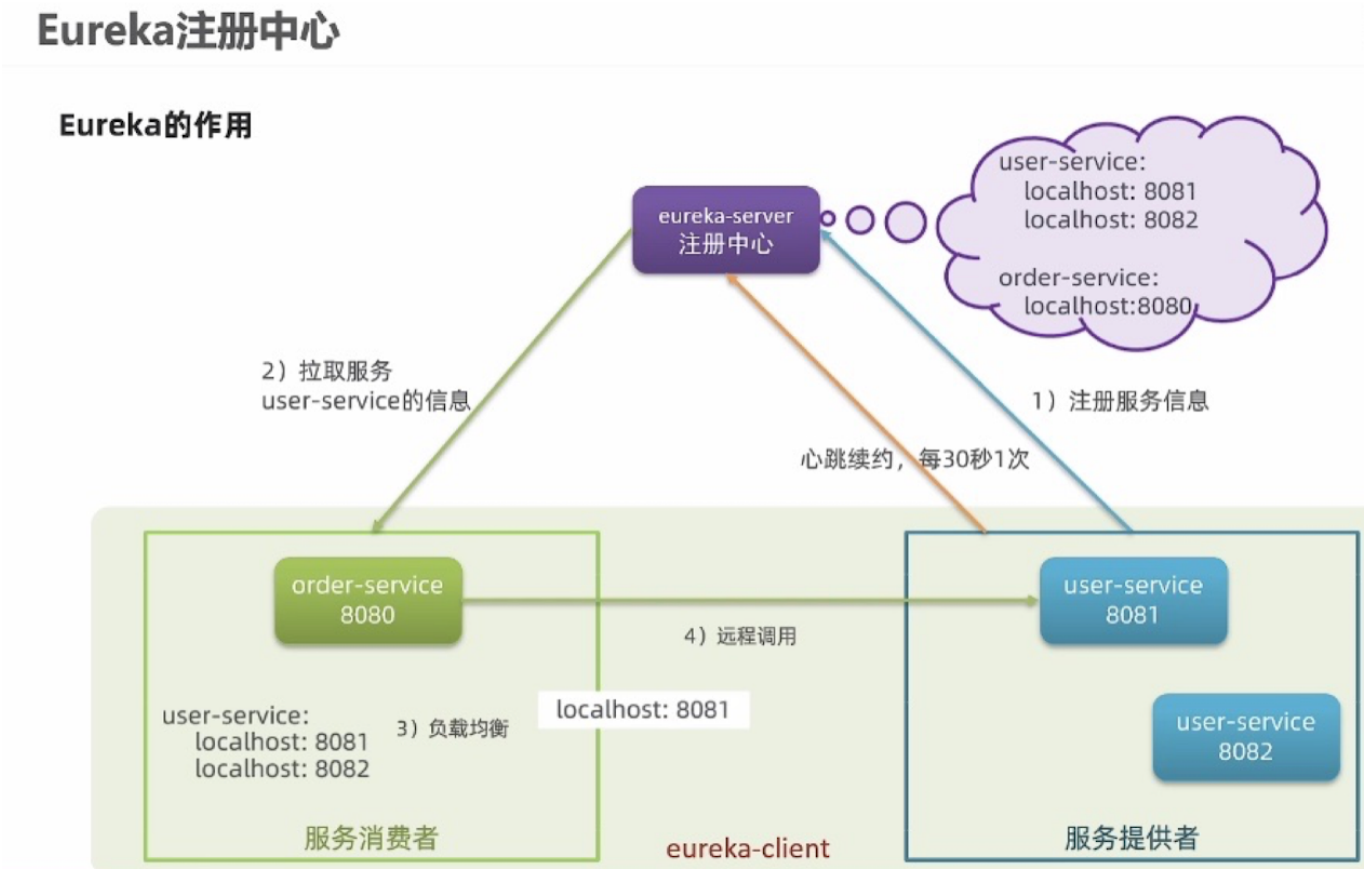
k8s的namespace类似于c++的namespace概念。namespace将同一集群中的资源划分为相互隔离的组。 同一namespace内的资源名称唯一，但跨namespace时没有这个要求。 namespace作用域仅针对带有namespace的对象， （例如 Deployment、Service 等），这种作用域对集群范围的对象 （例如 StorageClass、Node、PersistentVolume 等）不适用。

2 Spring cloud学习笔记

学而不思则罔，结合实际的微服务去部署和实践docker+k8s或许会增加我对这些工具的理解。因此我也在进行微服务框架的学习工作，先从Spring cloud开始吧！

本周学习到的Spring cloud组件主要有nacos和eureka两个，这些组件可以实现微服务的注册，发现和管理。

2.1 eureka



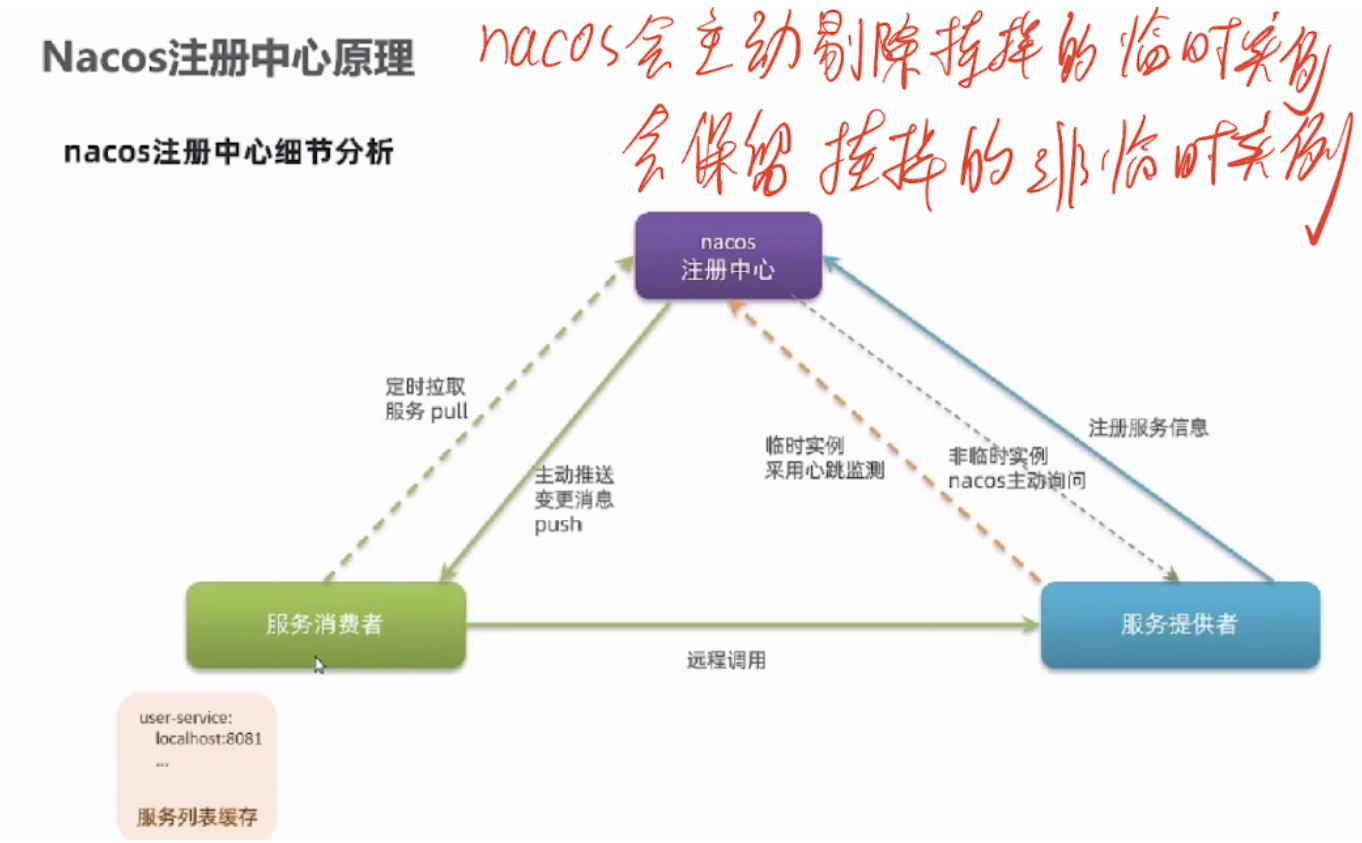
服务的消费者只需要从注册中心拉取消费者端的消息，然后根据消费者的信息实现对消费者服务的远程调用。

这里要做一个概念辨析，服务的提供者和消费者并不对应服务端/客户端这个概念，只是根据服务应用过程中的实际需要。每一个微服务都可以既是服务消费者又是提供者：在微服务A需要别的微服务B服务时，微服务A就是消费者；而在A给别的微服务C提供服务时，A就是消费者了。

实际使用中，我们通过：搭建eureka注册中心（先启动eureka程序）——>服务注册（在微服务端配置eureka依赖，端口等信息）——>服务发现（微服务消费者从注册中心远程调用服务提供者的信息并对其进行远程调用）三部曲来实现eureka的使用。

但是eureka相对来说服务功能很简陋，而且还需要借助ribbon组件实现负载均衡，因此我更喜欢nacos组件来实现微服务的管理。

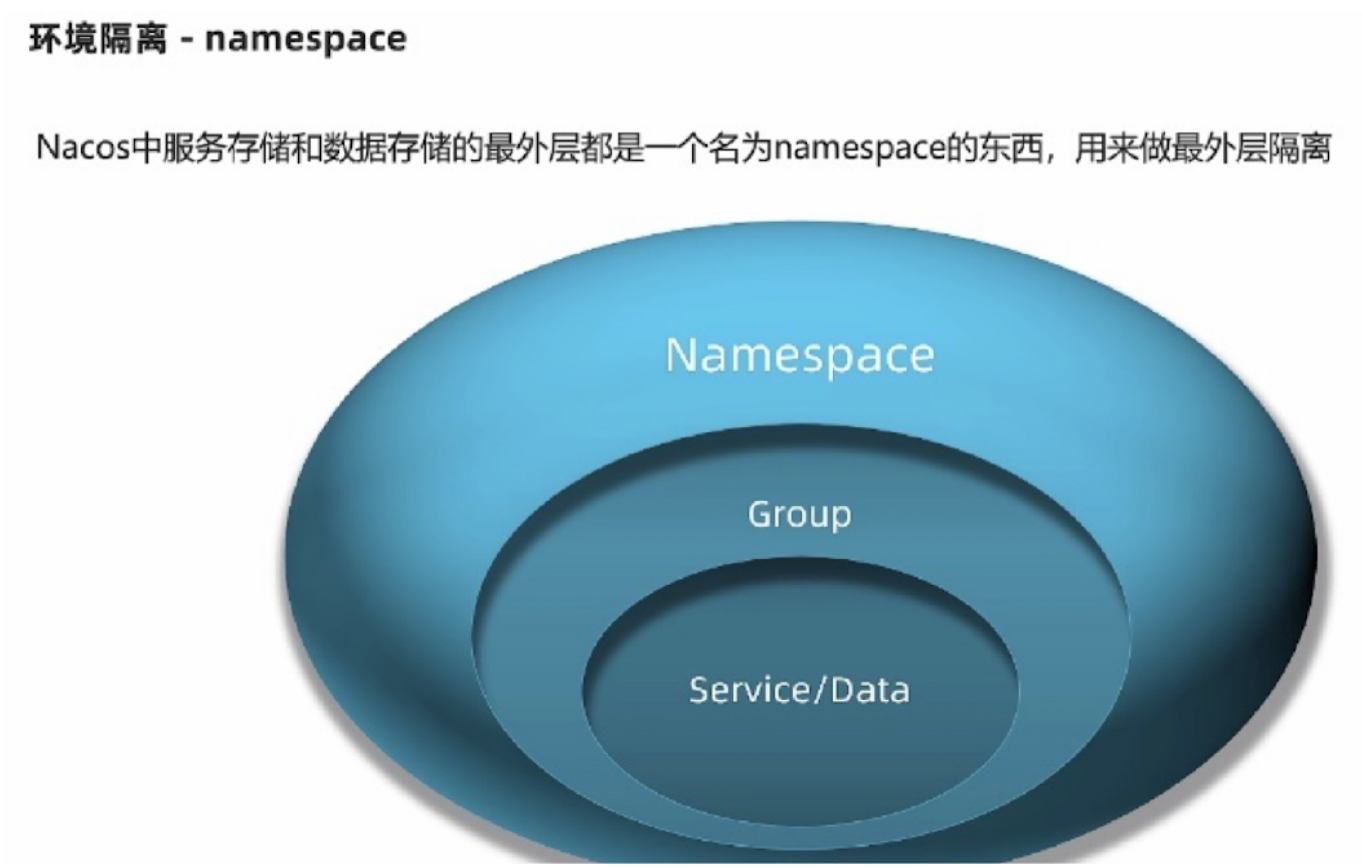
2.2 nacos



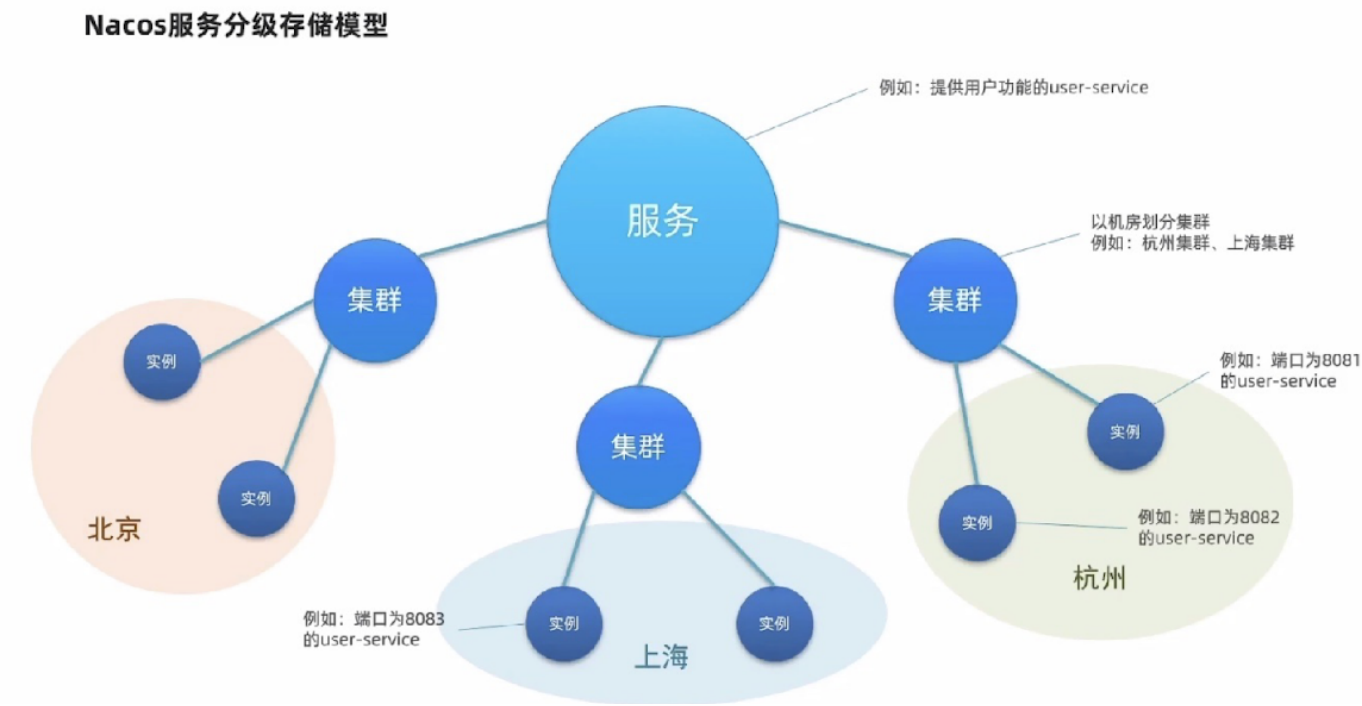
nacos的工作方式和eureka相近，如上图所示。但是nacos引入了非临时实例的概念。可以在服务提供者暂时挂掉的时候仍然保留其信息，方便运维人员下线维护出问题的服务。

除此之外，相比eureka，nacos还有几个更多的功能：

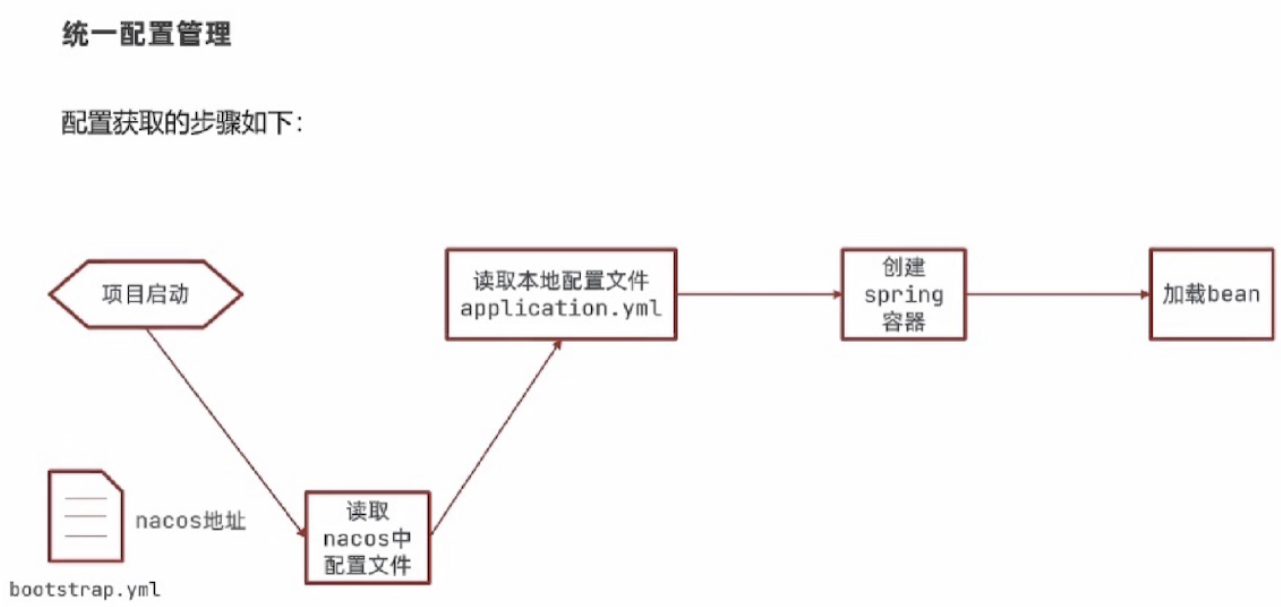
- 环境隔离：nacos引入了命名空间概念，可以将微服务进行不同层次的服务进行环境隔离，具体如下所示：



- 服务分级：nacos将服务分为服务，集群和实例三级，实现服务的灵活管理。同时，nacos在实现负载均衡时也会根据服务所在的集群是否相同而对不同服务进行不同的负载处理。



- 配置热管理：nacos的配置获取步骤如下图所示。我们可以看到，nacos的实时配置文件优先级更高，可以在读取服务的本地配置文件之前读取。因此，我们可以通过在nacos网页端直接编辑配置文件，来实现实时的配置变更。不同命名空间配置互相隔离，同一命名空间也可以根据环境来配置不同的服务配置。服务读取配置时，会根据环境配置->服务配置->本地配置的顺序读取配置。（因此我们可以把不同环境的相同配置写到服务配置层，而把各个环境的独特配置写在环境配置层，实现配置的分级管理）



- 负载均衡：nacos可以通过配置服务的负载权值，以及根据服务的不同分级（比如优先调用统一集群的服务）来进行简单的负载均衡工作，相比eureka而言nacos直接网页端配置就可以实现负载均衡，方便快捷。

3 下周学习计划

下周我主要计划继续学习Spring cloud的相关组件，同时根据学习进度将已经实现的微服务尝试迁移到docker上运行。我也会继续学习k8s的相关文档，理解k8s具体组件的实现细节和源码等。