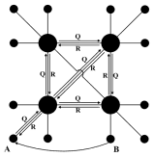
DATAGURU 炼数成金

Leader-us系列



从Docker到Kubernetes 第11周

法律声明

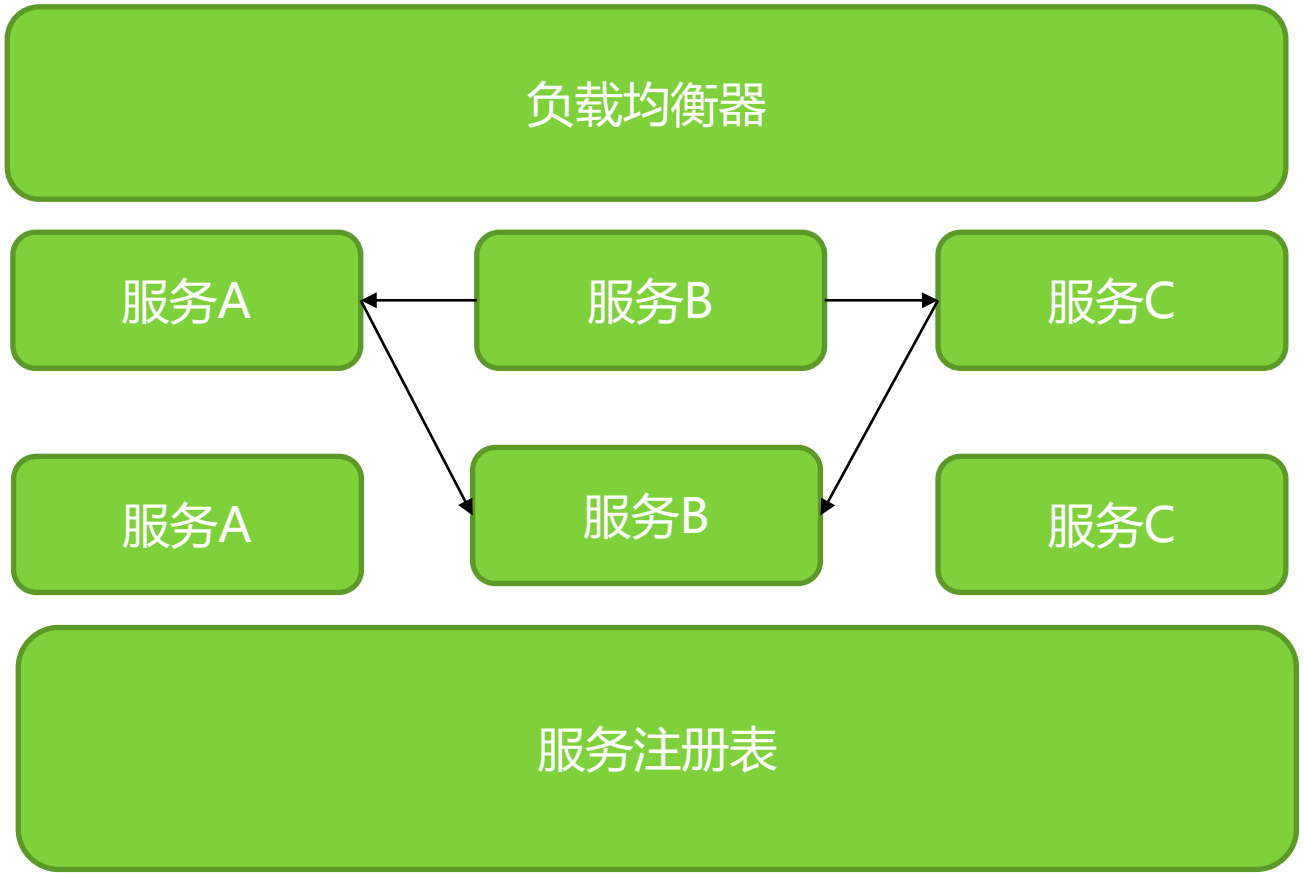
【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

- Kubernetes分布式集群架构
- Kubernetes 集群架构例子
- 集群运维常见问题

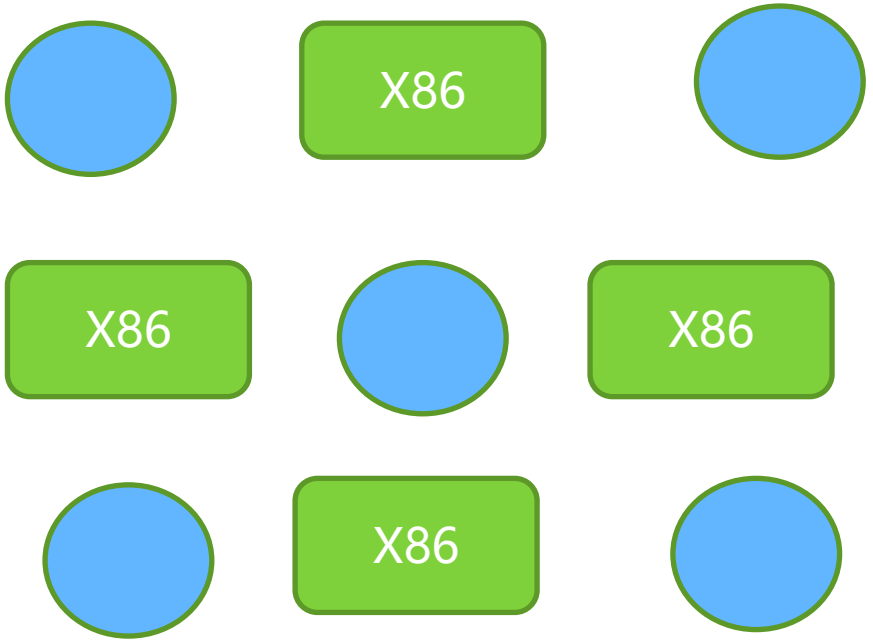
Kubernetes分布式集群架构



传统的架构

- 总体上零散，缺乏架构的整体性与系统性
- 服务注册表的方式，依然是入侵性的，缺乏直观性
- “服务”这个概念本身并没有被作为架构的一等公民

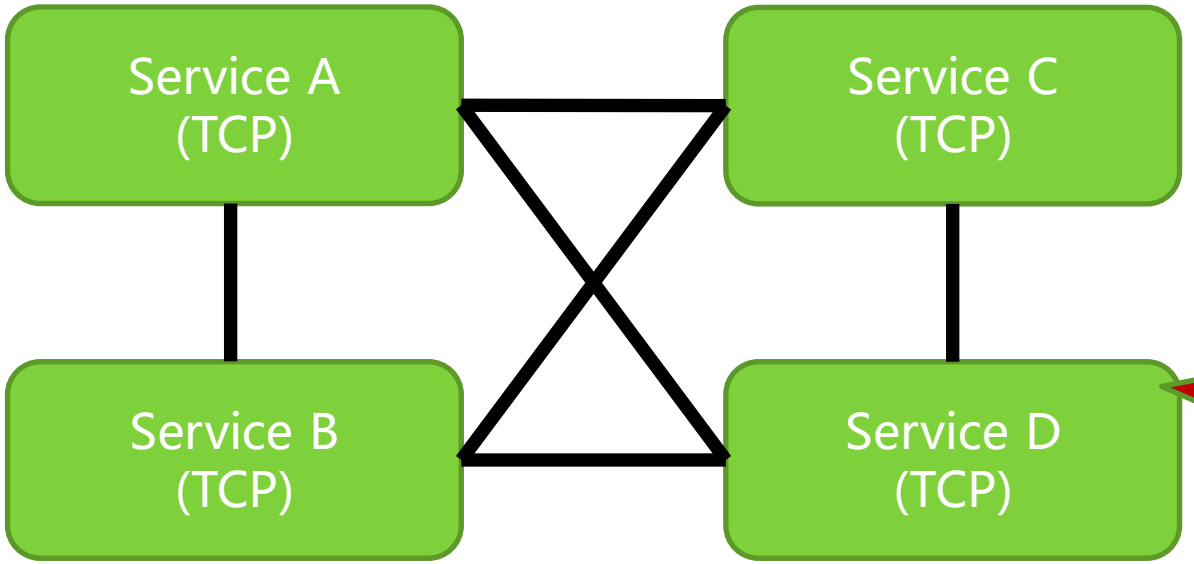
Kubernetes分布式集群架构



传统的架构

- 程序架构跟运行时态是相对分离的，无法从根本上保证分布式架构的最初设计
- 分布式规模部署的难题从未真正彻底解决

Kubernetes分布式集群架构



No Registy & Other Compent

Only Service

No Deployment Problem

No Maintainance Problem

Kubernetes分布式集群架构

服务注册和服务发现问题怎么解决的？

每个服务分配一个不变的虚拟IP+端口

系统env环境变量里有每个服务的服务名称到IP的映射

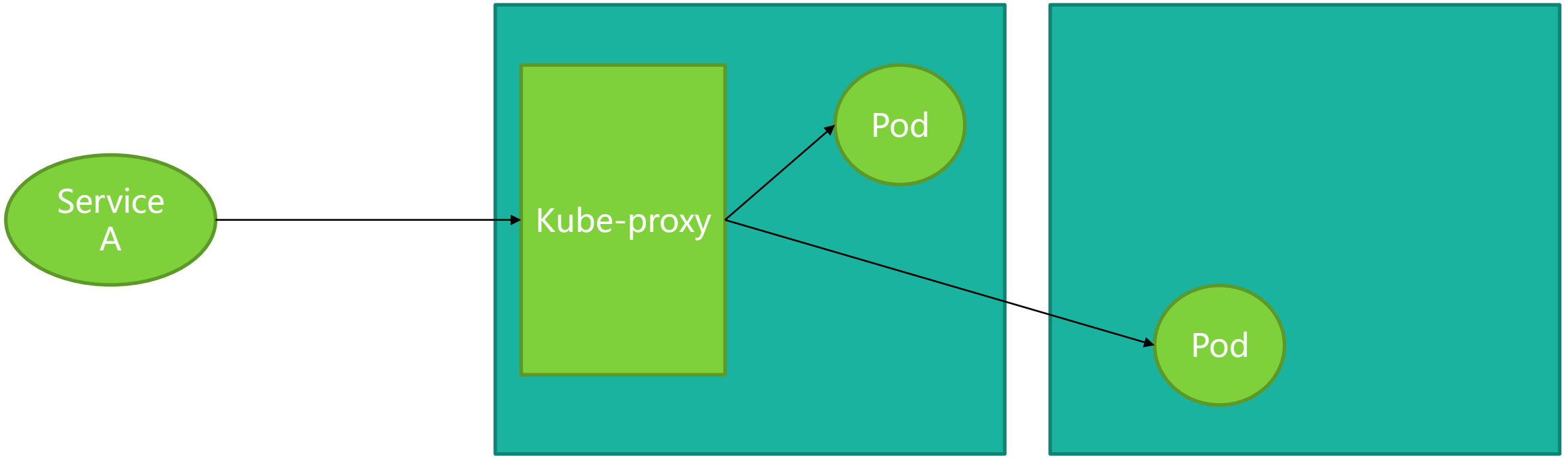
```
REDIS_MASTER_SERVICE_HOST=10.254.144.74

REDIS_MASTER_SERVICE_PORT=6379
```

```
client = new Predis\Client([
      'scheme' => 'tcp',
      'host'   => getenv('REDIS_MASTER_SERVICE_HOST') ,
      'port'   => $read_port,
    ]);
```

Kubernetes分布式集群架构

服务的负载均衡问题怎么解决的？

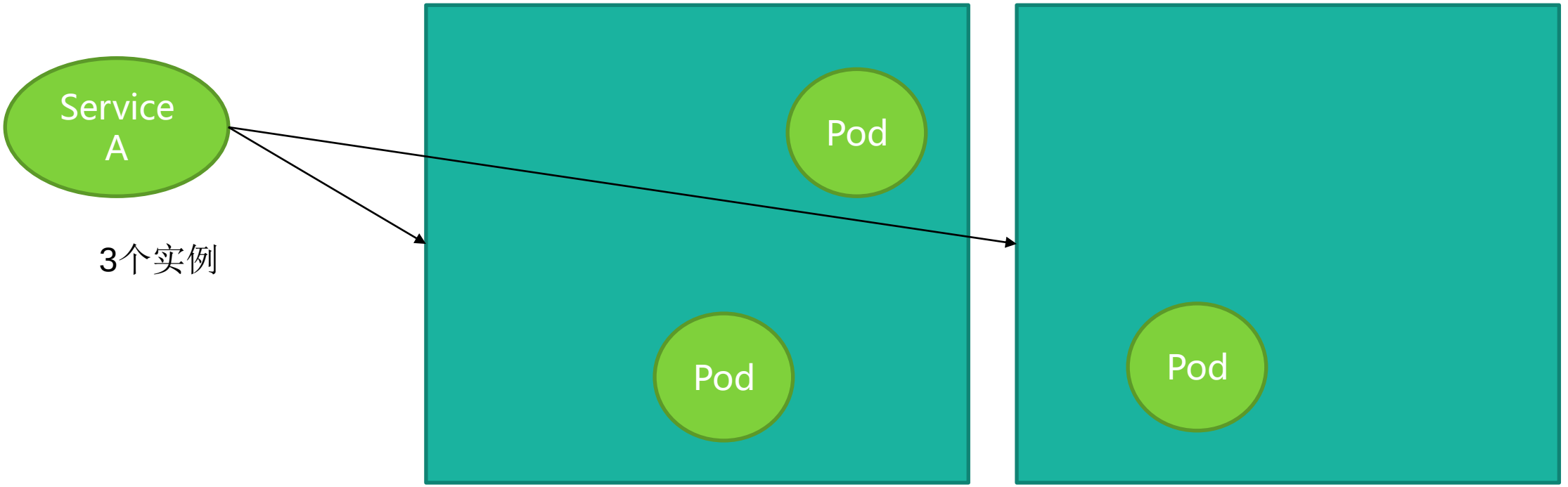
每个节点上都有一个软件实现的服务代理来实现负载均衡



Kubernetes分布式集群架构

服务的规模部署问题怎么解决的？

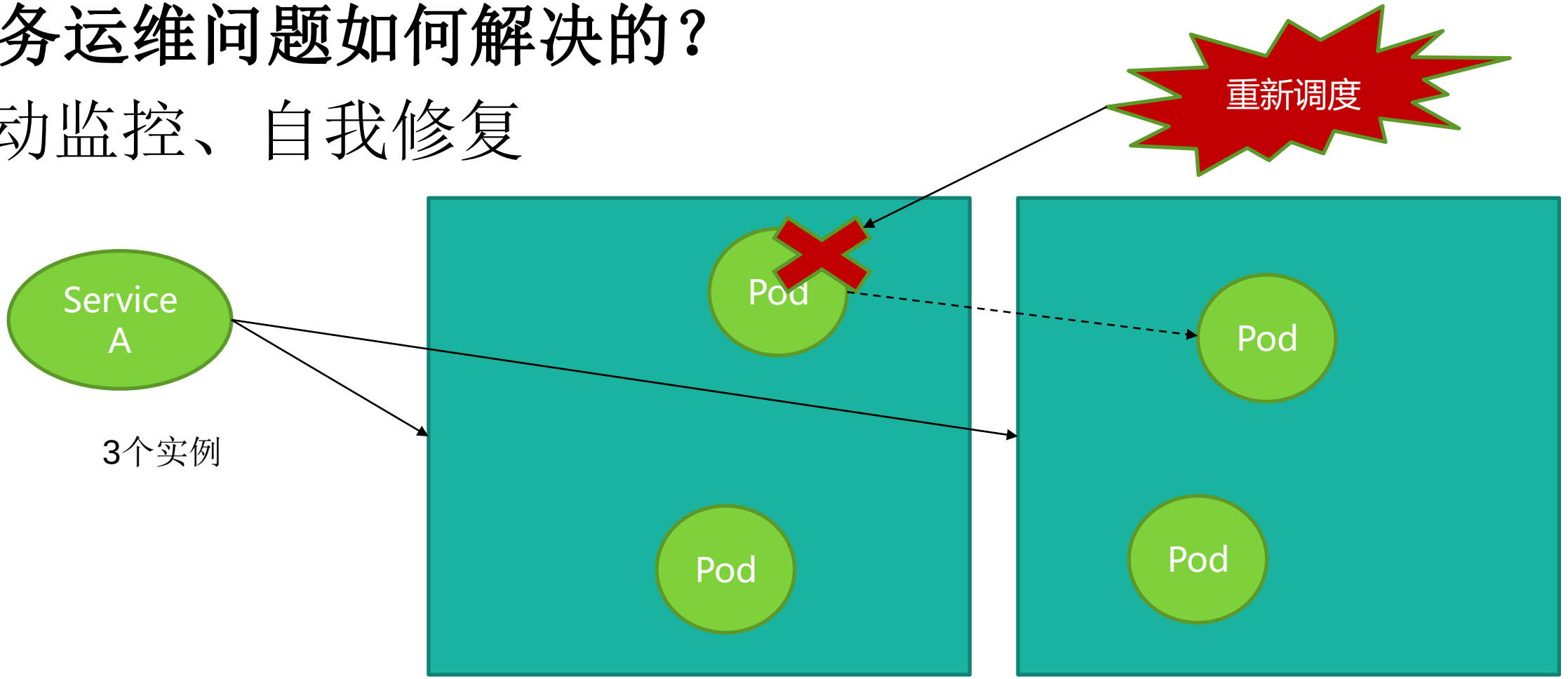
目标导向的做法：确定部署实例数，系统自动调度



Kubernetes分布式集群架构

服务运维问题如何解决的？

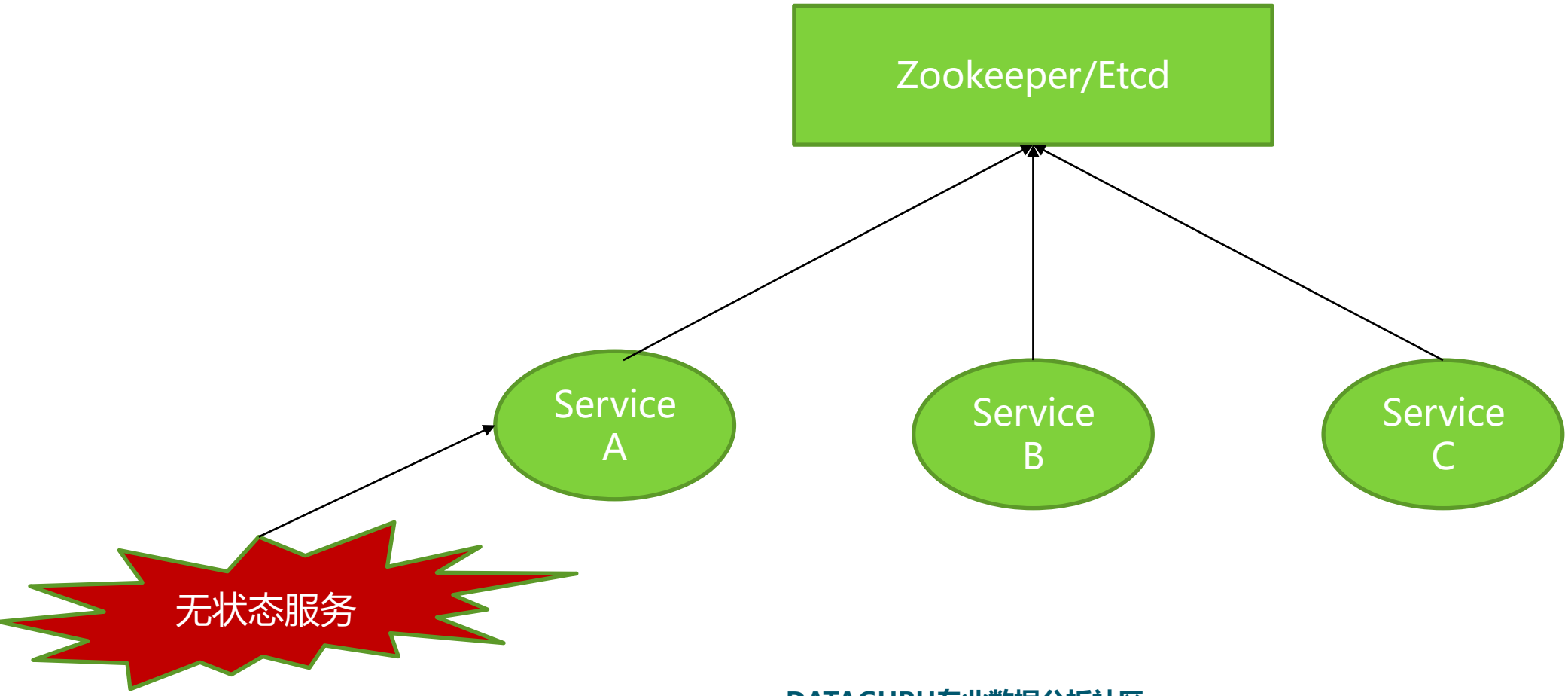
自动监控、自我修复



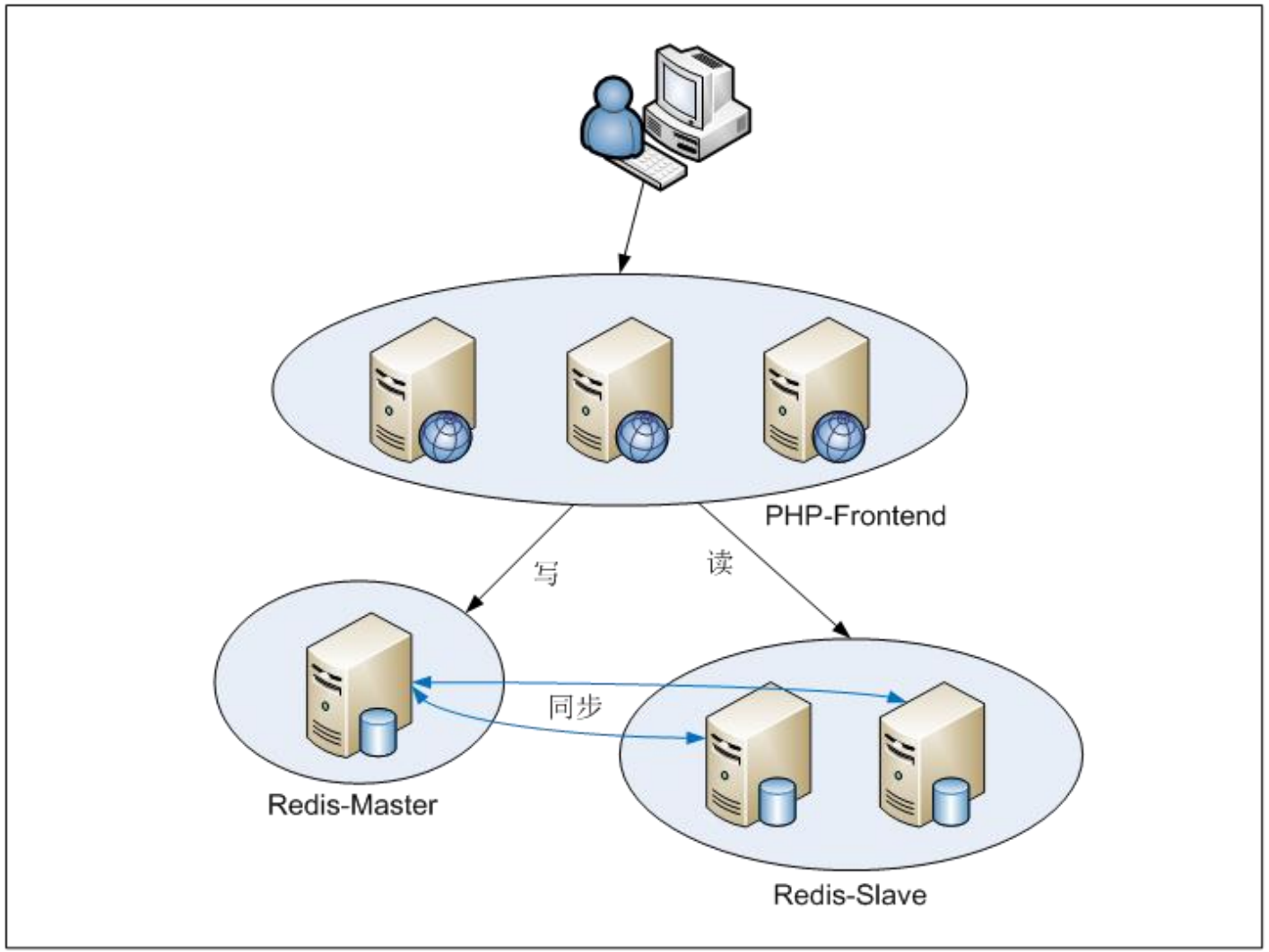
Kubernetes分布式集群架构

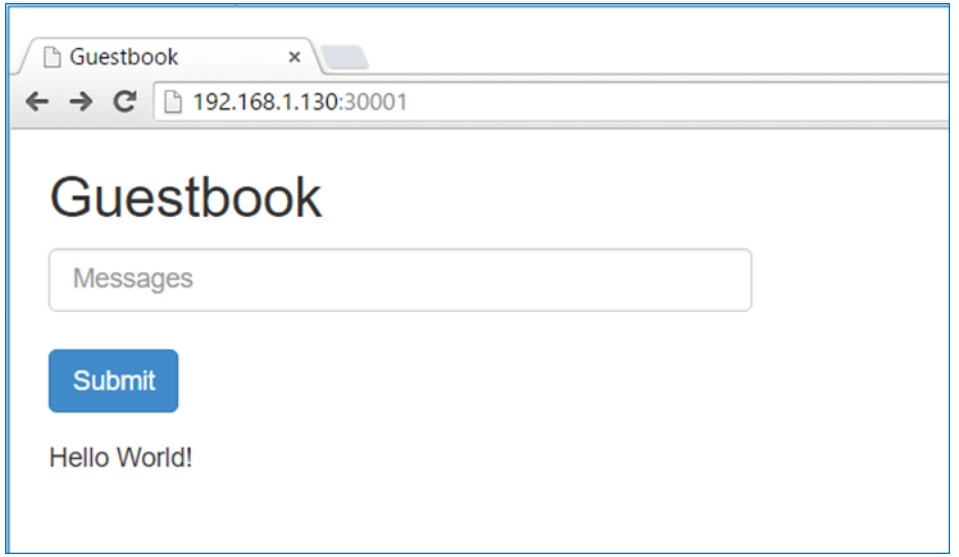
架构建议

集中配置，并且实时配置实施生效

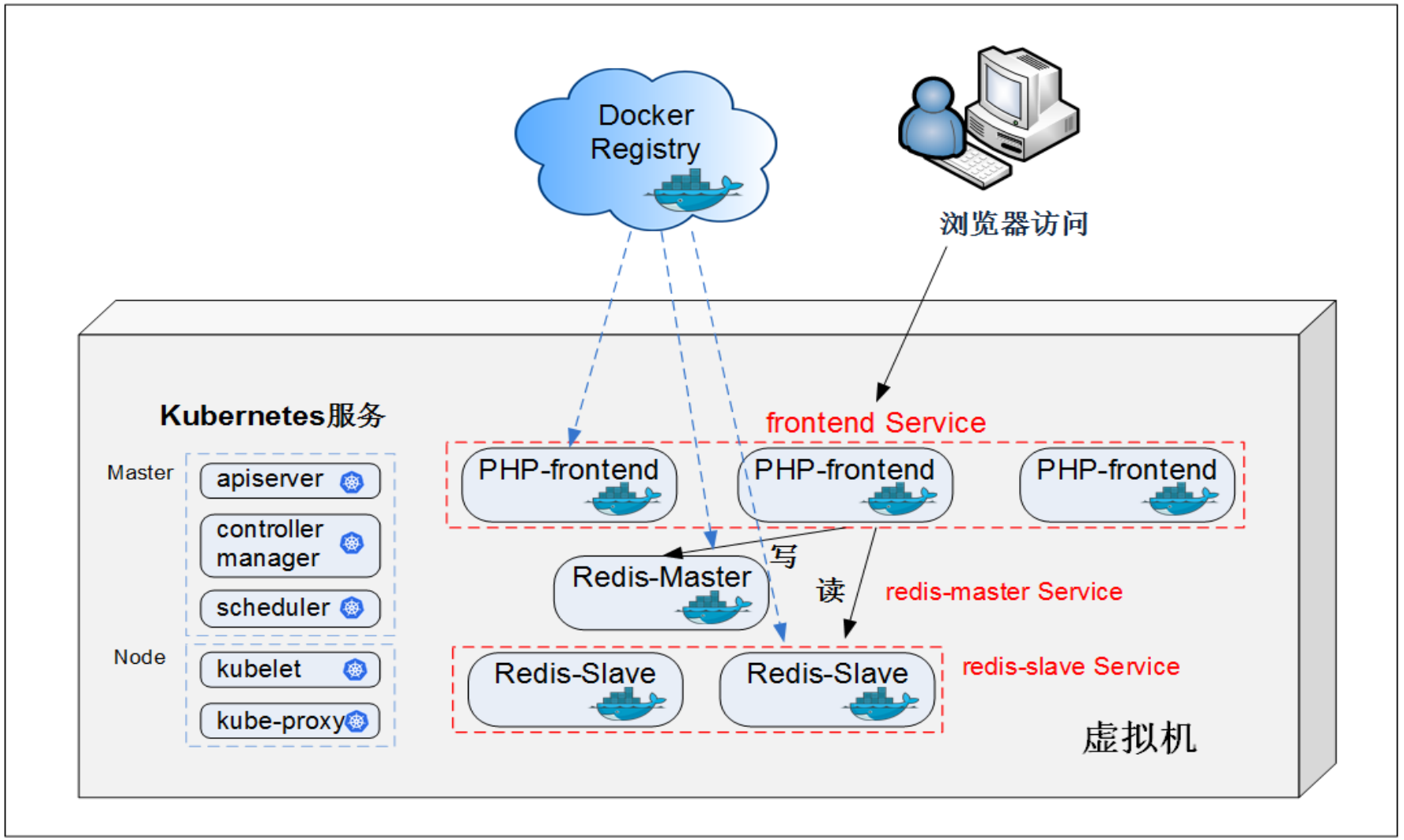


Kubernetes 集群架构例子





Kubernetes 集群架构例子



Kubernetes 集群架构例子

1.创建redis-master Pod和服务

redis-master-controller.yaml，下面给出了该文件的完整内容：

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  replicas: 1
  selector:
    name: redis-master
  template:
    metadata:
      labels:
        name: redis-master
    spec:
      containers:
      - name: master
        image: kubeguide/redis-master
        ports:
        - containerPort: 6379
```

我们可以先定义Service，然后定义一个RC来创建和控制相关联的Pods，或者先定义RC来创建Pods，然后定义与之关联的Service，这两种方式最终的结果都一样，这里我们采用后一种思路。

kubectl create -f redis-master-controller.yaml

kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
redis-master-b03io	1/1	Running	0	1h

Kubernetes 集群架构例子

1.创建redis-master Pod和服务

创建一个与之关联的Service（服务）redis-master的定义文件（文件名为**redis-master-service.yaml**），完整内容如下：

```
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    name: redis-master
```

spec.selector确定了哪些Pod对应到本服务，这里的定义表明拥有redis-master 标签的Pod属于redis-master 服务，另外，ports 部分中的targetPort属性用来确定提供该服务的容器所暴露（EXPOSE）的端口号，即具体的服务进程在容器内的targetPort上提供服务，而port属性则定义了Service的虚端口。

kubectl create -f redis-master-service.yaml

kubectl get services
NAME LABELS SELECTOR
IP(S) PORT(S)
redis-master name=redis-master
name=redis-master 10.254.208.57 6379/TCP

Kubernetes 集群架构例子

2. 创建redis-slave Pod和服务

redis-slave-controller.yaml，下面给出了该文件的完整内容：

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-slave
  labels:
    name: redis-slave
spec:
  replicas: 2
  selector:
    name: redis-slave
  template:
    metadata:
      labels:
        name: redis-slave
    spec:
      containers:
      - name: slave
        image: kubeguide/guestbook-redis-slave
        env:
        - name: GET_HOSTS_FROM
          value: env
        ports:
        - containerPort: 6379
```

kubectl create -f redis-slave-controller.yaml

kubectl get rc

```
CONTROLLER    CONTAINER(S)   IMAGE(S)
SELECTOR          REPLICAS
redis-master   master        kubeguide/redis-master
name=redis-master   1
redis-slave    slave         kubeguide/guestbook-redis-slave
name=redis-slave    2
```

Kubernetes 集群架构例子

2. 创建redis-slave Pod和服务

redis-slave-service.yaml的内容如下：

```
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
  labels:
    name: redis-slave
spec:
  ports:
  - port: 6379
  selector:
    name: redis-slave
```

kubectl create -f redis-slave-service.yaml

Kubernetes 集群架构例子

为了实现redis集群的主从数据同步，redis-slave需要知道redis-master的地址，所以在redis-slave镜像的启动命令/run.sh 中，我们可以如下内容：

```
redis-server --slaveof
${REDIS_MASTER_SERVICE_HOST} 6379
```

Kubernetes 集群架构例子

3. 创建frontend Pod和服务

类似地，定义frontend的RC配置文件——**frontend-controller.yaml**，内容如下：

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: frontend
  labels:
    name: frontend
spec:
  replicas: 3
  selector:
    name: frontend
  template:
    metadata:
      labels:
        name: frontend
    spec:
      containers:
      - name: frontend
        image: kubeguide/guestbook-php-frontend
        env:
        - name: GET_HOSTS_FROM
          value: env
        ports:
        - containerPort: 80
```

kubectl create -f frontend-controller.yaml

Kubernetes 集群架构例子

服务定义文件**frontend-service.yaml**的内容如下：

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    name: frontend
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30001
  selector:
    name: frontend
```

kubectl create -f frontend-service.yaml

Kubernetes 集群架构例子


```
if (isset($_GET['cmd']) === true) {
  $host = 'redis-master';
  if (getenv('GET_HOSTS_FROM') == 'env') {
    $host = getenv('REDIS_MASTER_SERVICE_HOST');
  }
  header('Content-Type: application/json');
  if ($_GET['cmd'] == 'set') {
    $client = new Predis\Client([
      'scheme' => 'tcp',
      'host'   => $host,
      'port'   => 6379,
    ]);

    $client->set($_GET['key'], $_GET['value']);
    print('{"message": "Updated"}');
  } else {
    $host = 'redis-slave';
    if (getenv('GET_HOSTS_FROM') == 'env') {
      $host = getenv('REDIS_SLAVE_SERVICE_HOST');
    }
    $client = new Predis\Client([
      'scheme' => 'tcp',
      'host'   => $host,
      'port'   => 6379,
    ]);
```

我们注意到Pod里提供的容器镜像为kubeguide/guestbook-php-frontend，该镜像中所包含的PHP的留言板源码（guestbook.php）如下：

Kubernetes 集群架构例子

经过上面的三个步骤，我们终于成功实现了留言板系统在Kubernetes上的部署工作，现在到了一起来见证成果的时刻了，在你的笔记本上打开浏览器，输入下面的URL：

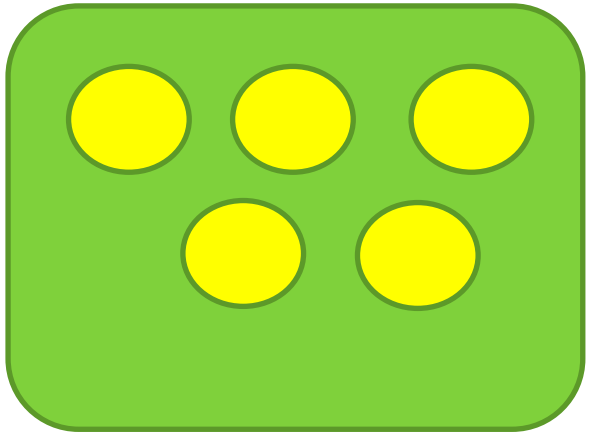
http://虚拟机IP:30001

如果看到如图1.4所示的网页，并且看到网页上有一条留言——“Hello World!”，那么恭喜你，之前的努力没有白费，如果看不到这个网页，可能有几个原因，比如防火墙的问题，无法访问30001端口，或者因为你是代理上网，浏览器错把虚机的IP地址当成远程地址了，如果这种情况无法解决，那么也可以在虚机上直接运行 curl localhost:30001 来验证此端口是否能被访问，如果还是不能访问，那么这肯定不是机器的问题…

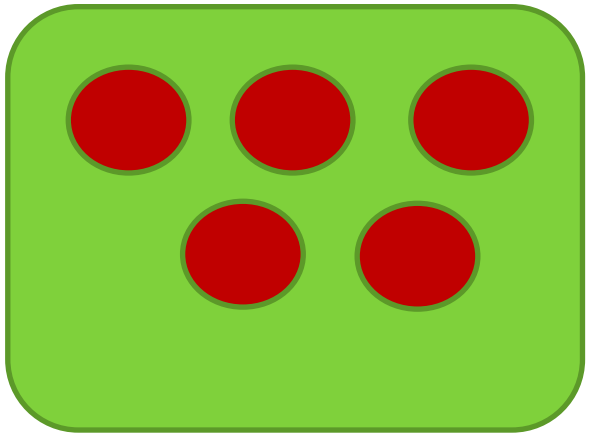
集群运维常见问题 资源隔离与调度问题

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  replicas: 1
  selector:
    name: redis-master
  template:
    metadata:
      labels:
        name: redis-master
    spec:
      containers:
      - name: master
        image: kubeguide/redis-master
        ports:
        - containerPort: 6379
      nodeSelector:
        zone: test
```

kubectl label nodes kubernetes-minion1 zone=test



测试环境



开发环境

kubectl label nodes kubernetes-minion1 zone=dev

集群运维常见问题　扩容与升级问题

RC　replicas

kubectl scale [--resource-version=version] [--current-replicas=count] --replicas=COUNT RESOURCE

--rollback

kubectl rolling-update frontend --image=image:v2

资源配额问题


```
apiVersion: v1
kind: ReplicationController
metadata:
  name: redis-master
  labels:
    name: redis-master
spec:
  replicas: 1
  selector:
    name: redis-master
  template:
    metadata:
      labels:
        name: redis-master
    spec:
      containers:
      - name: master
        image: kubeguide/redis-master
        ports:
        - containerPort: 6379
        resources:
          limits:
            cpu: 0.5
            memory: 128Mi
```

kube-apiserver ... --admission_control=LimitRanger,ResourceQuota

资源配额问题

当Kubernetes启动一个容器时，会将CPU配额值乘以1024并转为整数传递给docker run的--cpu-shares参数，之所以乘以1024是因为Docker的cpu-shares参数是以1024为基数计算CPU时间的。另外，Docker官方文档里解释说cpu-shares是一个相对权重值 (relative weight)，因此Kubernetes官方文档里解释cpu: 0.5表示该容器占用0.5个CPU计算时间的说法其实是不准确的。仅当该节点是单核心CPU而且只运行两个容器，每个容器的CPU配额设定为0.5时，上述说法才成立。假如一个节点上同时运行了3个容器A，B，C，其中A容器的CPU配额设置为1，B与C设置为0.5。那么，当系统的CPU利用率达到100%时，A容器只占用了1*100/(1+0.5+0.5)=50%的CPU时间，而B与C分别占用25%的CPU时间。如果此时我们加入一个新的容器D，它的CPU配额也设置为1，则通过计算我们得到A此时只占据33%的CPU时间。对于目前主流的多核CPU，容器的CPU配额会在多核心上进行承担。因此在多核CPU上，即使某个容器声明CPU<1，它也可能会占满多个CPU核。例如2个设定cpu=0.5的容器运行在4核的CPU上，每个容器可能会用光4*0.5/(0.5+0.5)=2个CPU核。

来自Kubernetes权威指南

集群运维常见问题　私有docker registry

docker run -d -e SETTINGS_FLAVOR=dev -e STORAGE_PATH=/tmp/registry -v /opt/data/registry:/tmp/registry -p 5000:5000 registry　可以从/opt/data/registry下找到私有仓库都存在哪些镜像

我本地是在Master节点192.168.131.134建立的私有仓库，地址是**192.168.131.134:5000**

在Minion节点上修改docker守护进程的配置文件，

增加**--insecure-registry 192.168.131.134:5000**（私有仓库的访问地址），并重启

在Minion节点上修改kubelet的配置文件，增加pod_infra_container_image 参数，修改pause image为私有仓库的image

```
[root@centos-minion ~]# cat /etc/kubernetes/kubelet
# The address for the info server to serve on (set to 0.0.0.0 or "" for all interfaces)
KUBELET_ADDRESS="--address=0.0.0.0  --pod_infra_container_image=192.168.131.134:5000/google_containers/pause"
```

上述操作需要在每一个Minion节点上执行通过。

在Master节点或者任何一个已经下载了google_containers/pause镜像的机器上，将此镜像重新打tag，标识成为私有仓库的镜像，并推送到私有仓库中：

打标签（**2c40b0526b63为google_containers/pause的Id**）

```
docker tag 2c40b0526b63    192.168.131.134:5000/google_containers/pause
```

推到私有镜像中

```
docker push 192.168.131.134:5000/google_containers/pause
```

查看私有镜像的内容

```
[root@centos-minion ~]# docker search 192.168.131.134:5000/google_containers
```

Thanks

FAQ时间