中国平安 PING AN
保险·银行·投资

Docker技术分享

平安产险科技中心
技术架构组-赵延斌
2018年1月

中国平安 PING AN
保险·银行·投资

目录
Content

一、Docker基础

二、K8S容器编排

三、容器化落地介绍

四、K8S的持续集成

1.1 从一个Hello World开始

运行命令启动容器：

docker run **hub.yun.paic.com.cn/zhaoyanbin442/oraclelinux:7** echo 'Hello World'

```
[root@SZD-L0077282 ~]# docker run hub.yun.paic.com.cn/zhaoyanbin442/oraclelinux:7 echo 'Hello World'
Hello World
[root@SZD-L0077282 ~]#
```

运行命令启动容器并进入容器内部：

docker run –it hub.yun.paic.com.cn/zhaoyanbin442/oraclelinux:7 bash

-t 选项让Docker分配一个伪终端（pseudo-tty）并绑定到容器的标准输入上
-i 则让容器的标准输入保持打开

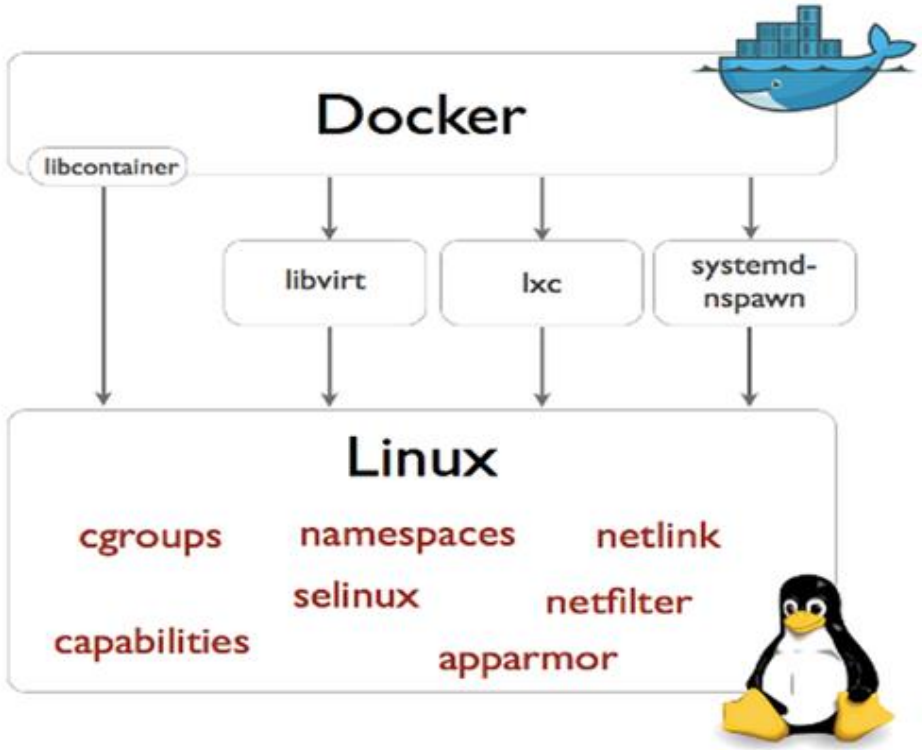
```
[root@SZD-L0077282 ~]# docker run -it hub.yun.paic.com.cn/zhaoyanbin442/oraclelinux:7 bash
[root@d7a498fb0c88 /]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@d7a498fb0c88 /]# ps
  PID TTY          TIME CMD
    1 ?        00:00:00 bash
   14 ?        00:00:00 ps
[root@d7a498fb0c88 /]# id
uid=0(root) gid=0(root) groups=0(root)
```

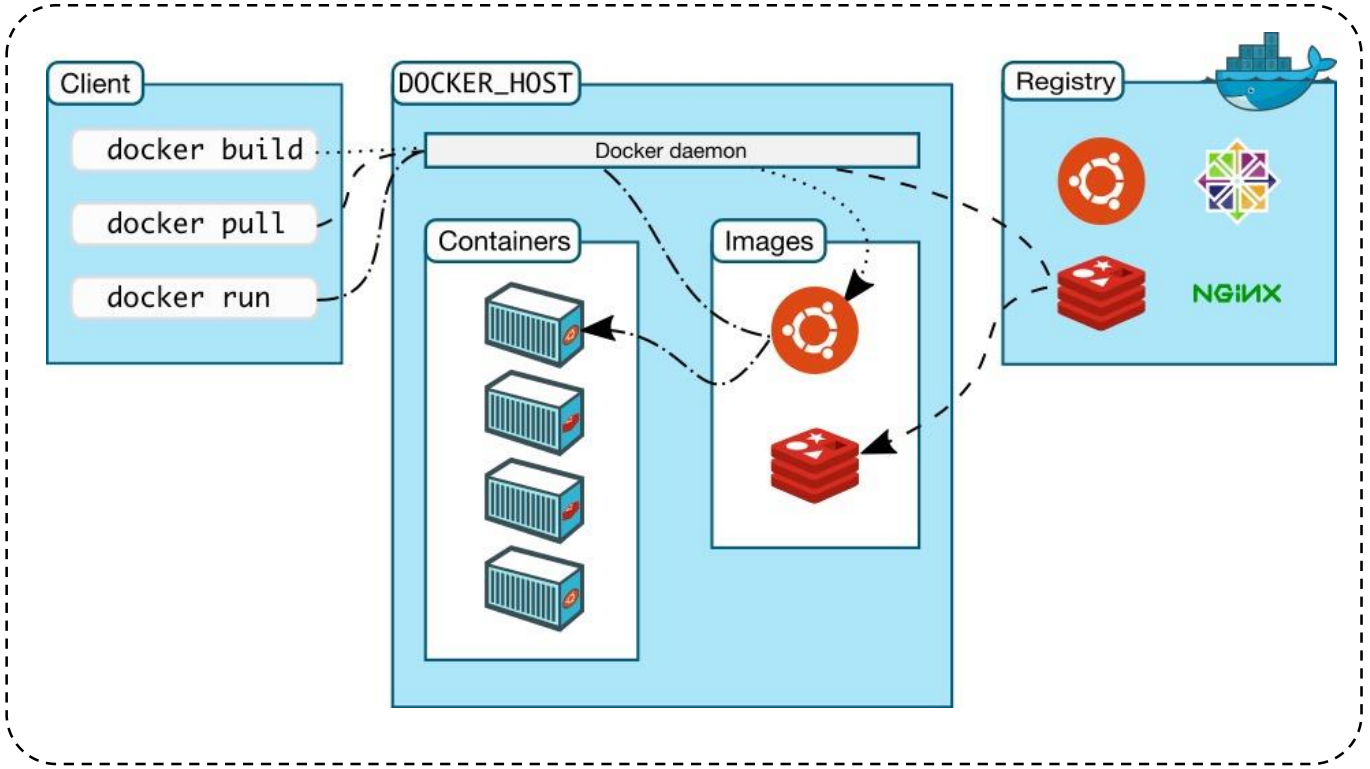
查看容器实例：

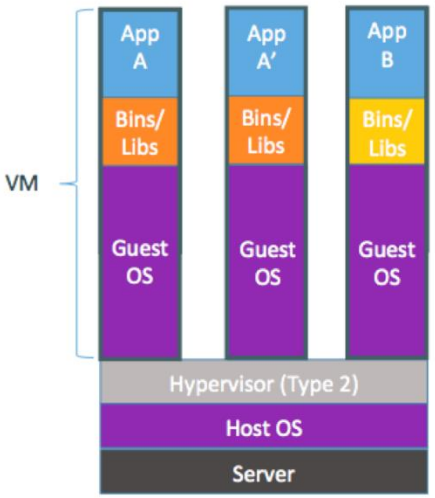
Docker ps

```
[root@SZD-L0077282 ~]# docker ps
CONTAINER ID        IMAGE                                                 COMMAND
             CREATED              STATUS              PORTS               NAMES
d7a498fb0c88        hub.yun.paic.com.cn/zhaoyanbin442/oraclelinux:7       "bash"
            About a minute ago   Up About a minute                       blissful_ardinghelli
```

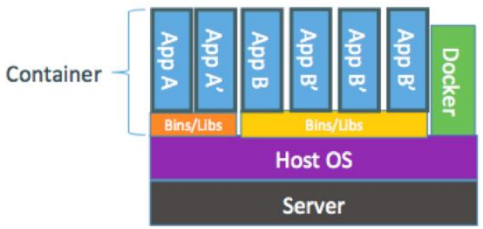
1.2 Docker 的基本原理及相关术语简单介绍







Containers are isolated, but share OS and, where appropriate, bins/libraries



- Docker　平台工具、守护进程
- Image　镜像，包括应用及系统的只读数据包
- Container　容器，虚拟机的运行实例
- Registry　仓库登记，管理镜像文件
- Docker hub　Docker官方公有仓库

1.3 容器的基本操作

1. 容器的停止启动：

docker stop d7a498fb0c88

docker start d7a498fb0c88

```
[root@SZD-L0077282 ~]# docker stop d7a498fb0c88
d7a498fb0c88
[root@SZD-L0077282 ~]#
[root@SZD-L0077282 ~]# docker start d7a498fb0c88
d7a498fb0c88
[root@SZD-L0077282 ~]#
[root@SZD-L0077282 ~]# docker ps
CONTAINER ID        IMAGE
            CREATED                  STATUS                  PORTS
d7a498fb0c88        hub.yun.paic.com.cn/zhaoyanbin442/oraclelinux:7
```

2. 查看容器日志：

docker logs d7a498fb0c88

```
[root@SZD-L0077282 ~]# docker logs d7a498fb0c88
[root@d7a498fb0c88 /]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt
[root@d7a498fb0c88 /]# ps
  PID TTY          TIME CMD
    1 ?        00:00:00 bash
   14 ?        00:00:00 ps
[root@d7a498fb0c88 /]# id
uid=0(root) gid=0(root) groups=0(root)
[root@d7a498fb0c88 /]#
[root@d7a498fb0c88 /]#
[root@d7a498fb0c88 /]#
```

3. 进入容器内部：

docker exec -it d7a498fb0c88 bash

docker attach f21963204ab3（退出会导致容器停止）

```
[root@SZD-L0077282 ~]# docker logs d7a498fb0c88
[root@d7a498fb0c88 /]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt
[root@d7a498fb0c88 /]# ps
  PID TTY          TIME CMD
    1 ?        00:00:00 bash
   14 ?        00:00:00 ps
[root@d7a498fb0c88 /]# id
uid=0(root) gid=0(root) groups=0(root)
[root@d7a498fb0c88 /]#
[root@d7a498fb0c88 /]#
[root@d7a498fb0c88 /]#
```

5. 容器隔离：

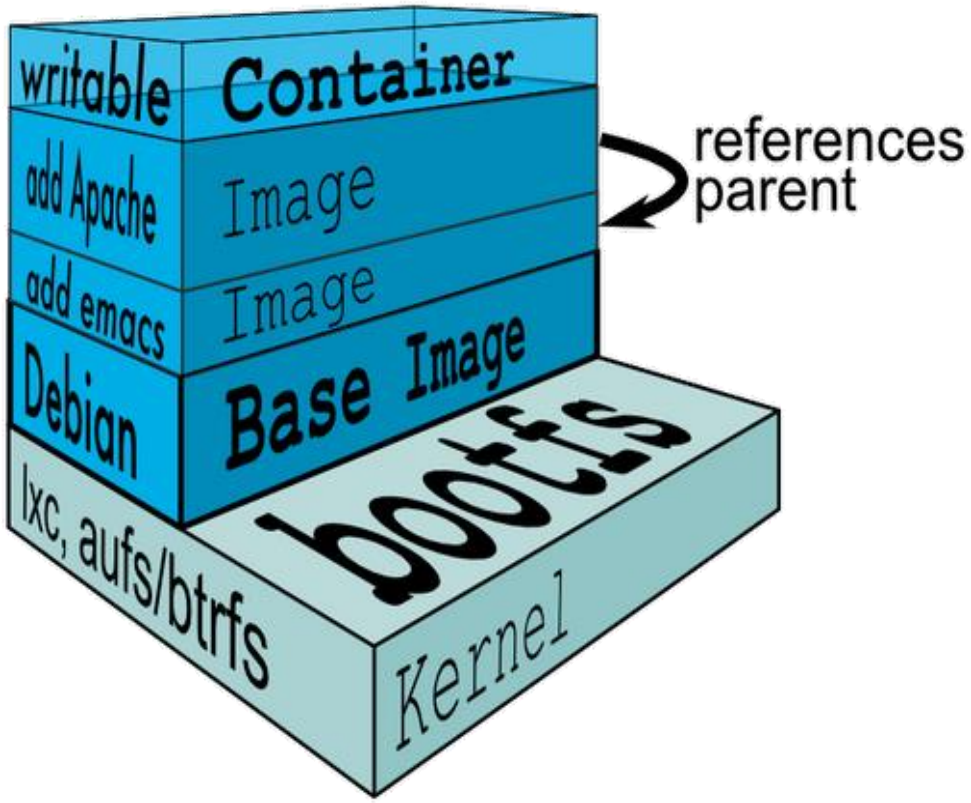
容器内执行简单shell：sh docker_shell.sh

```
[root@d7a498fb0c88 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Jan06 ?        00:00:00 bash
root        13     0  0 Jan06 ?        00:00:00 bash
root        52     0  0 11:38 ?        00:00:00 bash
root        86    52  0 11:40 ?        00:00:00 sh docker_shell.sh
root        89     0  0 11:40 ?        00:00:00 bash
root       101    86  0 11:41 ?        00:00:00 sleep 5
root       102    89  0 11:41 ?        00:00:00 ps -ef
[root@d7a498fb0c88 /]#
```

1.4 Docker 镜像原理简单介绍

Docker AUFS特性

- Docker镜像位于bootfs之上
- 每一层镜像的下面一层称为其父镜像(父子关系)
- 第一层镜像为Base Image
- 容器在最顶层
- 其下的所有层都为readonly
- Docker将readonly的FS层称作"image"



查看虚拟机镜像列表：
docker images

```
[root@SZD-L0077282 ~]# docker images
REPOSITORY                                                      TAG
        SIZE
hub.yun.paic.com.cn/cicd/wizard-k8s-cicdpoc1.0.0/javametrics    wizard-k8s-cicdpoc1.0.0-d4b73f7cee99eb1
ur ago    450 MB
hub.yun.paic.com.cn/cicd/wizard-k8s-cicdpoc1.0.0/javametrics    wizard-k8s-cicdpoc1.0.0-0fc5187a81c7d9e
ur ago    450 MB
hub.yun.paic.com.cn/cicd/wizard-k8s-cicdpoc1.0.0/javametrics    wizard-k8s-cicdpoc1.0.0-c032609de73b923
ur ago    450 MB
```

查看镜像详细信息：
Dockerinspect images:tag


```
"RootFS": {
        "Type": "layers",
        "Layers": [

"sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",

"sha256:dc44cf6f74b6770b59e77c1146184b637338afcf8f236ed37d2055cb00dbdbcc",

"sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",

"sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",

"sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",

"sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef",

"sha256:087379b882c665789af81f3fac2580260cc0a3eca6c4e818df446bbc6f71e11e",

"sha256:49d806cbdf2047811693ed0d0bb810bfee25ab8cd99d1239ad454d6d82537584",

"sha256:2ef2f0477f3a025295fa680cde401eeaea9b8ad8bbb178308c129eb782c43930"
        ]
```

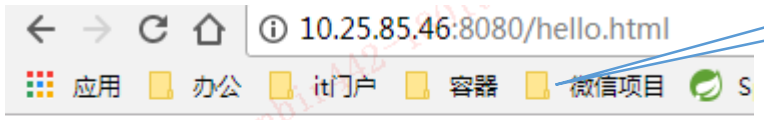
1.5 怎样构建docker镜像及访问容器内服务

```
FROM hub.yun.paic.com.cn/official/nginx:1.8.1
MAINTAINER zhaoyanbin <zhaoyanbin442>
COPY hello.html /usr/share/nginx/html
```

编写镜像构建描述文件：dockerfile

```
[root@SZD-L0077282 nginx]# docker build -t hello_nginx:1.0.0 .
Sending build context to Docker daemon 3.072 kB
Step 1/3 : FROM hub.yun.paic.com.cn/official/nginx:1.8.1
1.8.1: Pulling from official/nginx
4f4fb700ef54: Pull complete
fd61901c6550: Pull complete
05fe01a0826f: Pull complete
c6c25f27c788: Pull complete
b40e4900c014: Pull complete
57689335ceb2: Pull complete
8aaab78bf236: Pull complete
5a81470f2f53: Pull complete
Digest: sha256:17dd4d3125e2bf2e150f5f243d4c14d11dc9e5d5d779c1d533bcae2caffc25f6
Status: Downloaded newer image for hub.yun.paic.com.cn/official/nginx:1.8.1
 ---> c2e41b65940a
Step 2/3 : MAINTAINER zhaoyanbin <zhaoyanbin442>
 ---> Running in e16329eced8e
 ---> 1b7abb8690a9
Removing intermediate container e16329eced8e
Step 3/3 : COPY hello.html /usr/share/nginx/html
 ---> 590bd7b4ae63
Removing intermediate container 7a39bdec97e5
Successfully built 590bd7b4ae63
[root@SZD-L0077282 nginx]#
```

执行docker build



hello word

docker run --name hello-nginx -i -p 8080:80 hello_nginx:1.0.0

提示：
除了dockerfile的方式外，还可以通过将容器启动起来，安装配置完成后通过docker commit containerid 的方式由容器创建镜像

1.6 Docker 网络简要介绍

1. 查看docker网络：
docker network ls

```
[root@SZD-L0077282 ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8c9e150f13fc        bridge              bridge              local
4c9372376214        host                host                local
052a60294915        none                null                local
```

none 网络

故名思议，none 网络就是什么都没有的网络。挂在这个网络下的容器除了 lo，没有其他任何网卡。容器创建时，可以通过 --network=none 指定使用 none 网络

host 网络

连接到 host 网络的容器共享 Docker host 的网络栈，容器的网络配置与 host 完全一样。可以通过 --network=host 指定使用 host 网络

Bridge 网络

Docker 安装时会创建一个 命名为 docker0 的 linux bridge。如果不指定--network，创建的容器默认都会挂到 docker0 上。

自定义 网络

Docker 提供三种 user-defined 网络驱动：bridge, overlay 和 macvlan。overlay 和 macvlan 用于创建跨主机的网络。

2. 查看docker bridge 网络详情：
docker network inspect bridge

```
[root@SZD-L0077282 ~]# docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "8c9e150f13fc47c4254d79bd1a5d157becef633335533a99676d6975dc94e827",
        "Created": "2017-12-18T10:49:57.562689444+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.1.137.1/24",
                    "Gateway": "172.1.137.1"
                }
            ]
        },
        "f182aab6c47fe26e4569a08f9914b858a9546ccb68fc7d504ea308f27a9eec01": {
            "Name": "hello-nginx",
            "EndpointID": "9e079269c3d598becd76dda73b15be25b19f236d650fac381fbaca22220df085",
            "MacAddress": "02:42:ac:01:89:07",
            "IPv4Address": "172.1.137.7/24",
            "IPv6Address": ""
        }
```

3. 通过ifconfig可见容器网络的网关是docker0

```
[root@SZD-L0077282 ~]# ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 172.1.137.1  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::42:9eff:fe98:5951  prefixlen 64  scopeid 0x20<link>
        ether 02:42:9e:98:59:51  txqueuelen 0  (Ethernet)
        RX packets 39681777  bytes 12142157445 (11.3 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 40866072  bytes 82145923307 (76.5 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

1.7 Docker 存储简单介绍

1、Docker 为容器提供了两种存放数据的资源：

- 由 storage driver 管理的镜像层和容器层：

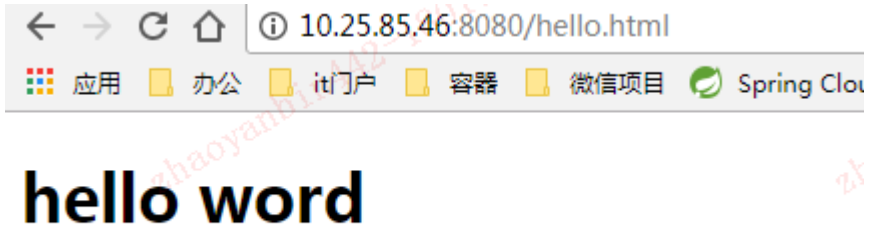
  Docker 支持多种 storage driver，有 AUFS、Device Mapper、Btrfs、OverlayFS、VFS 和 ZFS

- Data Volume：

  Data Volume 本质上是 Docker Host 文件系统中的目录或文件，能够直接被 mount 到容器的文件系统中

2. 相同示例介绍虚拟机存储挂载到容器内部：

```
docker run --name hello-nginx3 -i -p 8080:80 -v
/opt/zhaoyanbin442/nginx/html:/usr/share/nginx/html hub.yun.paic.com.cn/official/nginx:1.8.1
```



二、K8S容器编排

2.1 容器编排框架的必要性

1、容器数量很少的情况下没有必要，如果我们有十几个、上百容器服务，我们怎么管理？
2、当我们需要跨节点自由部署容器的时候怎么办？
3、我们管理大量的容器跨主机通信的情况下怎么办？
4、当我们需要动态伸缩容器服务的时候我们怎么办？
5、当我们需要从源码做devops构建的时候我们怎么办？

1、我们需要一个框架帮我们统一管理底层计算、网络以及存储资源
2、我们需要一个框架帮我们跨多个节点、甚至跨不同AZ调度容器
3、我们需要一个框架帮我们管理服务版本并能进行滚动升级、回滚以及弹性伸缩
4、我们需要一个框架能支撑从源码构建服务的devops流程

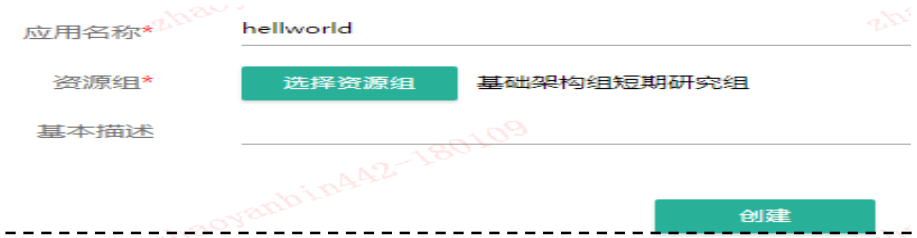
二、K8S容器编排

2.2 基于K8S的容器服务Hell World

1. 推送镜像到仓库：

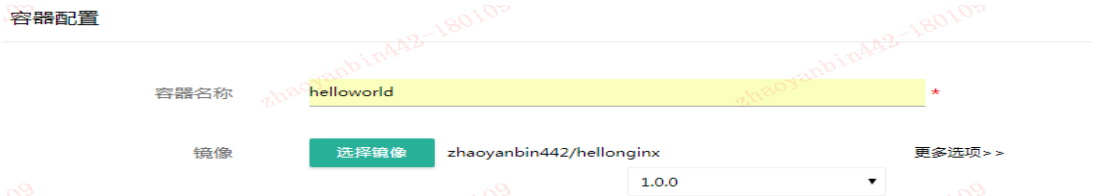
```
[root@SZD-L0077282 ~]# docker push hub.yun.paic.com.cn/zhaoyanbin442/hellonginx:1.0.0
The push refers to a repository [hub.yun.paic.com.cn/zhaoyanbin442/hellonginx]
de3b7db04039: Pushed
bb15a1293ba9: Pushed
5f70bf18a086: Mounted from zhaoyanbin442/node-exporter
65091cbd8fdd: Pushed
ef442477bbe2: Pushed
d3ffa230b8c2: Pushed
805e29ee5b61: Pushed
4f992dea5bc1: Pushed
f4577addd63e: Pushed
1.0.0: digest: sha256:c214c363274f0787db04afdb6c4027751be019c87888c7c0b7e023f99d8e3356 size: 4051
[root@SZD-L0077282 ~]#
```

2. Caas 创建应用：

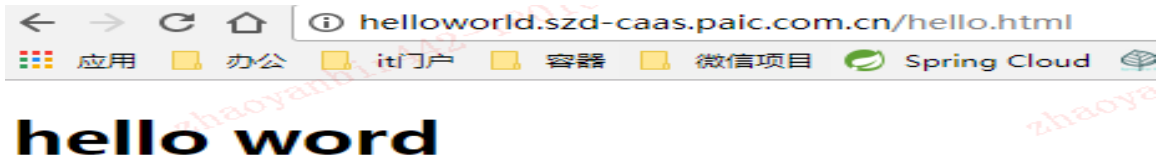


3. 通过Caas 创建容器服务：

4. 选择镜像并配置容器：



5. 通过Caas 配置容器访问：



6. 配置容器服务外部访问负载均衡入口：

二、K8S容器编排

2.3 结合实例介绍K8S的基本概念及主要操作

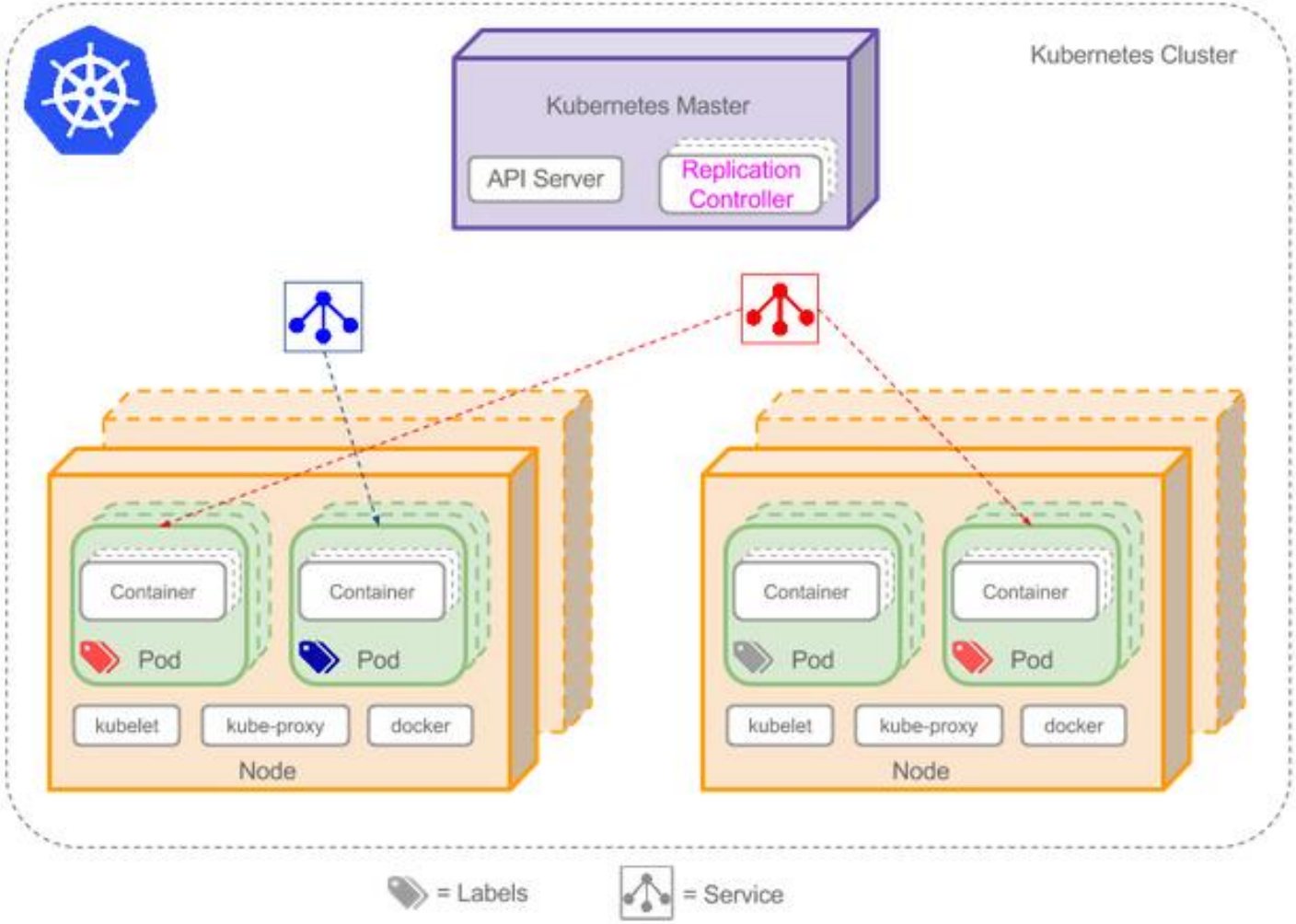
pod：
Replicasets、statefulsets 、daemonsets

获取pod列表：kubectl get pod （**podid** –o yaml）
查看pod控制台日志：kubectl logs **podid**
查看pod详情：kubectl describe pod **podid**
进入pod内的容器：kubectl exec –it **podid** bash

service：

获取service列表：kubectl get svc （**svcname** –o yaml）
查看service详情：kubectl describe svc **svcname**
编辑service：kubectl edit svc **svcname**

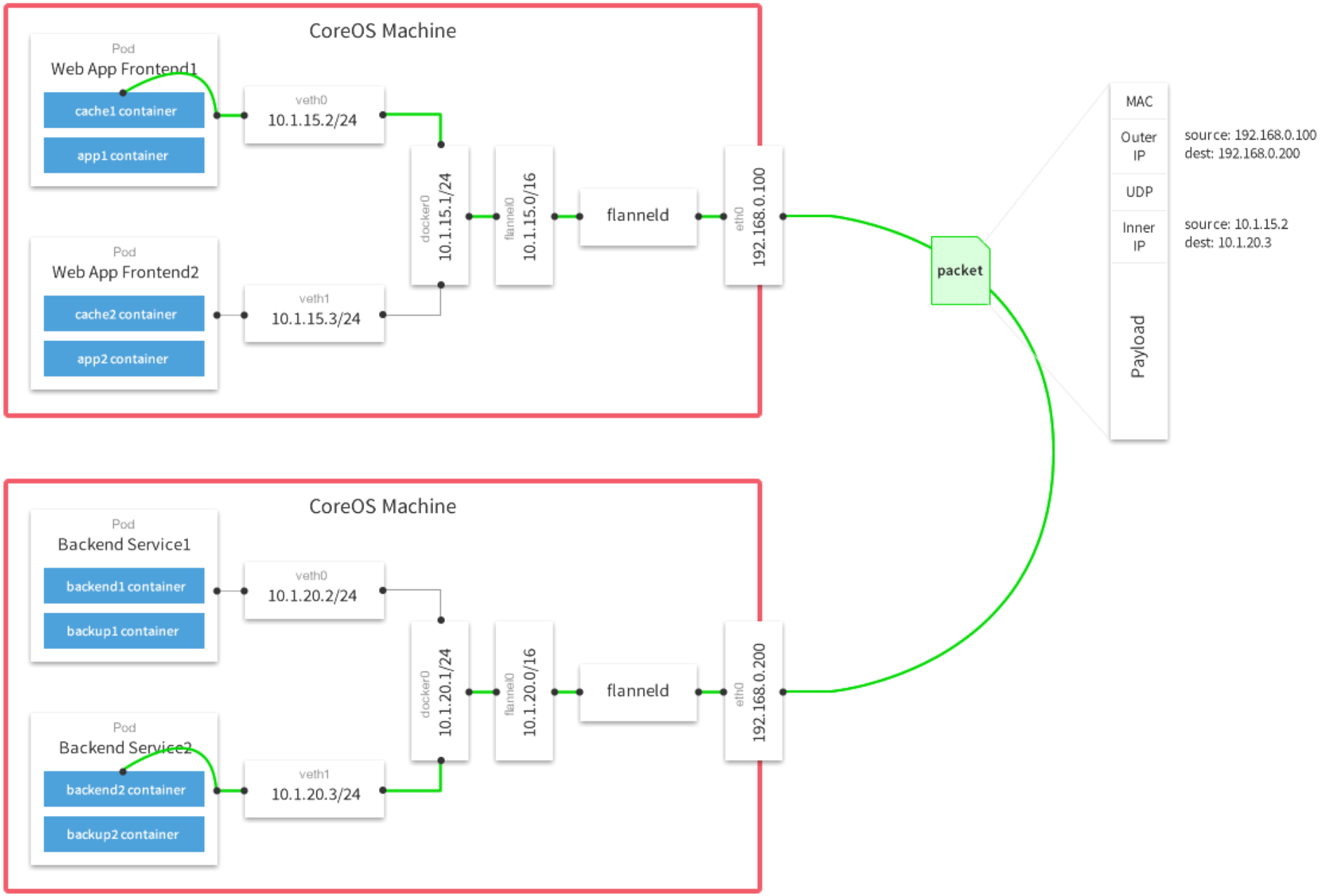
deployment：
根据yaml创建: kubectl create(apply/delete) –f xxx.yaml
获取deployment列表：kubectl get deployment
（**deployment** –o yaml）
查看deployment详情：kubectl describe deployment **deploymentname**
编辑deployment ：kubectl edit deployment **deploymentname**



Ingress ：

获取ingress列表：kubectl get ingress （**ingressname** –o yaml）
查看ingress详情：kubectl describe ingress **ingressname**
编辑ingress ：kubectl edit ingress **ingressname**

2.4 以flanneld为例介绍K8S的网络



2.4 K8S的存储介绍

1、EmptyDir：

- EmptyDir的用处是，可以在同一 Pod 内的不同容器之间共享工作过程中产生的文件
- 缺省情况下，EmptyDir 是使用主机磁盘进行存储的，也可以设置emptyDir.medium 字段的值为Memory

2、HostPath：

- 这种会把宿主机上的指定卷加载到容器之中

3、NFS/GlusterFS/CephFS/AWS/GCE 等等：

- Kubernetes 支持为数众多的云提供商和网络存储方案

4、ConfigMap 和 Secret：

- Kubernetes 支持将配置信息以环境变量或者文件的形式注入到容器中提供不同环境差异化运行

5、 PV & PVC ：

- PersistentVolume 和 PersistentVolumeClaim 提供了对存储支持的抽象，也提供了基础设施和应用之间的分界，管理员创建一系列的 PV 提供存储，然后为应用提供 PVC，应用程序仅需要加载一个 PVC，就可以进行访问

2.5 资源管理及调度

1、Namespace：

- 在一个Kubernetes集群中，可以使用Namespace创建多个“虚拟集群”，这些Namespace之间可以完全隔离，也可以通过某种方式，使一个Namespace中的Service可以访问到其他Namespace中的Service

2、node 及 lable：

- Node是Kubernetes集群的工作节点，它可以是物理机，也可以是虚拟机。我们创建的Pod，都运行在Kubernetes集群中的每个Node节点上。而且，在每个Node上还会存在一个运行容器的daemon进程，比如Docker daemon进程，它负责管理Docker容器的运行。
- 我们可以通过给Node打不同的标签（比如ssd节点，物理机节点），通过标签调度Pod在不同的Node上运行

3、资源调度：

- kubernetes通过一组规则，为每一个未调度的Pod选择一个主机，如调度流程中介绍，kubernetes的调度算法主要包括两个方面，过滤主机和主机打分

4、节点亲和性：

- Kubernetes 可以通过配置必要条件和可选条件将Pod调度到最合适的主机节点运行

5、Pod亲和性：

- Kubernetes 支持Pod服务的亲和和反亲和，即我们的哪些服务不能被调度到同一节点，哪些服务需要调度到同一节点，以及自身反亲和（同一节点不能运行两个相同的pod服务）

2.6 实例讲解K8S的扩容缩容

1、扩容：kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f scale --replicas=2 deploy/helloworld

```
[root@SZD-L0077283 ~]# kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f get deployment
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
grafana-core        1         1         1            1           13d
helloworld          1         1         1            1           13h
```

```
[root@SZD-L0077283 ~]# kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f scale --replicas=2 deploy/helloworld
deployment "helloworld" scaled
[root@SZD-L0077283 ~]# kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f get deployment
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
grafana-core        1         1         1            1           13d
helloworld          2         2         2            2           13h
```

2、缩容：kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f scale --replicas=1 deploy/helloworld

```
[root@SZD-L0077283 ~]# kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f get deployment
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
grafana-core        1         1         1            1           13d
helloworld          2         2         2            2           13h
```

```
[root@SZD-L0077283 ~]# kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f scale --replicas=1 deploy/helloworld
deployment "helloworld" scaled
[root@SZD-L0077283 ~]# kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f get deployment
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
grafana-core        1         1         1            1           13d
helloworld          1         1         1            1           13h
```

提示：

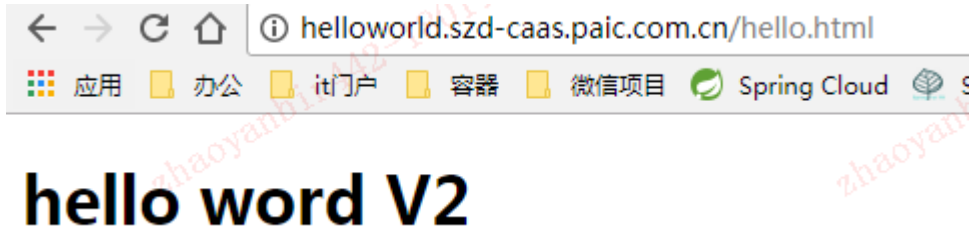
1. **除了kubectl 命令外直接curl k8s api也能实现同样功能**
2. **可以通过定制策略或者结合第三方监控实现自动伸缩**

2.6 K8S的滚动升级与回滚

```
[root@SZD-L0077282 nginx]# cat hello.html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>hello word V2</title>
</head>
<body>
<h1>hello word V2</h1>
</body>
</html>
```

```
docker build -t hello_nginx:1.1.0 .
docker tag hello_nginx:1.1.0 hub.yun.paic.com.cn/zhaoyanbin442/hellonginx:1.1
docker push hub.yun.paic.com.cn/zhaoyanbin442/hellonginx:1.1.0
```

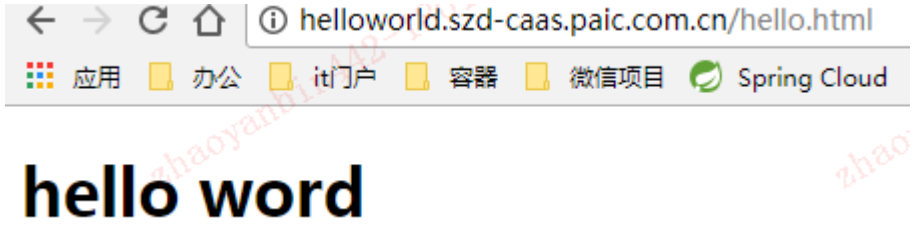
kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f set image deploy helloworld helloworld=hub.yun.paic.com.cn/zhaoyanbin442/hellonginx:1.1.0




```
kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f get deployment helloworld -o yaml
```

```
spec:
  containers:
  - image: hub.yun.paic.com.cn/zhaoyanbin442/hellonginx:1.1.0
    imagePullPolicy: Always
    name: helloworld
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
  dnsPolicy: ClusterFirst
```

kubectl -s 10.25.65.209:8080 --namespace=szd-0879950f rollout undo deploy helloworld




```
spec:
  containers:
  - image: hub.yun.paic.com.cn/zhaoyanbin442/hellonginx:1.0.0
    imagePullPolicy: Always
    name: helloworld
    resources: {}
    terminationMessagePath: /dev/termination-log
```

3.1 Weblogic服务容器化

1、镜像构建：

- 通过COPY的方式构建基础weblogic镜像
- 通过wlst脚本的方式离线配置weblogic资源（jdbc数据源）和应用部署
- 通过在线weblogic控制台的方式配置安全域等资源，然后再通过脚本进行环境差异化配置

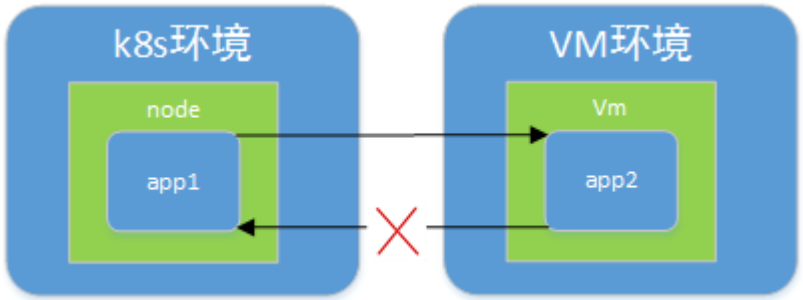
2、weblogic启动配置参数化：

- JVM内存参数：java应用启动都需要定制内存堆大小，gc策略等参数以获取更好的性能
- 自定义classpath：有些应用启动前需要JVM提前加载一些容器级别的自定义配置及jar包
- 自定义的-Dxxx java options：除了上述配置以外，有些应用还需要配置额外的环境参数，通过-Dxxx=yyy的方式加载。

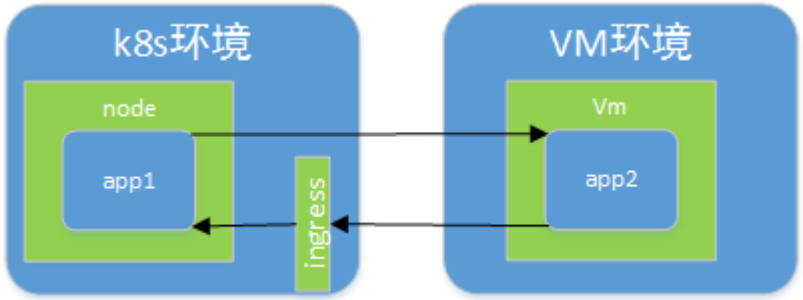
3、遇到的问题：

- 由于涉及绑定主机名，基于weblogic中间件的SSO不可用

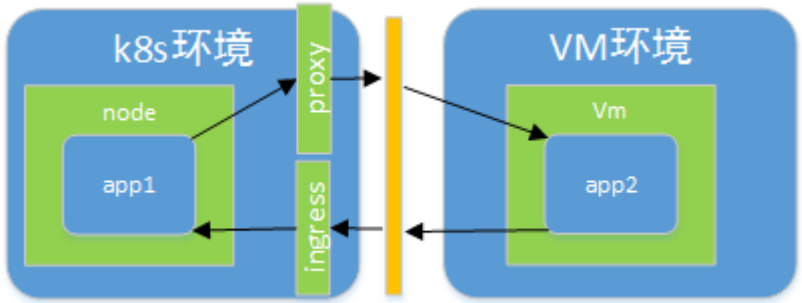
3.2 K8S对外提供服务及调用外部服务



- K8s中的pod服务有自己的ip并且外部虚拟机不能直接访问
- VM环境中的服务直接使用VM的ip提供服务，容器可以访问到



- K8s可以通过ingress的方式将外部请求引流到内部
- K8s可以通过nodeport（类似docker的端口映射，k8s开放30000-32767直接的端口到node）的方式对外提供服务



- 需要穿墙的时候就涉及到k8s环境node到外部服务VM间开墙
- 当集群节点数量很多、甚至多个app部署在同一个集群的情况下，开墙对应用的扩展造成很大的阻碍
- 我们引入nginx来对pod服务代理墙外服务，这样应用可以灵活扩展，开墙的影响降到最低

3.3 系统容器化后与ESG的集成

1、应用与esg注册中心等服务的通讯：

- 将ESG注册中心等服务注册到k8s内的nginx服务，供内部pod服务调用

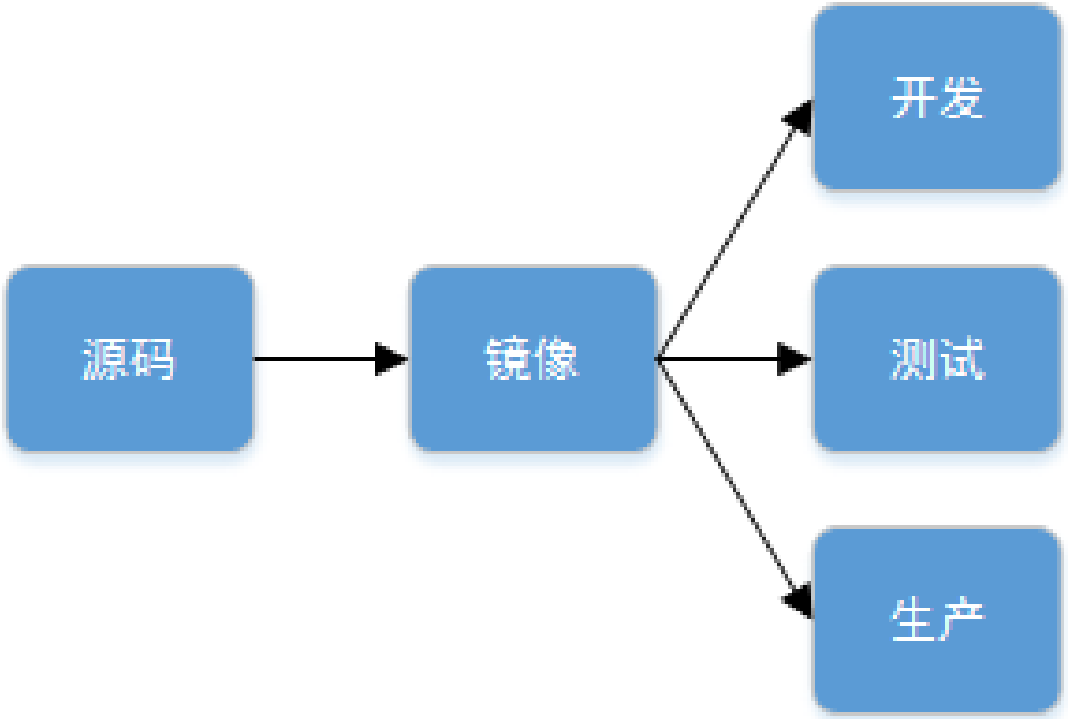
2、应用作为服务提供方怎么做：

- 因为pod内ip对k8s环境外的虚拟机不可达，需要将服务以负载均衡的方式注册到esg
- 注册前端的F5虚ip或者pod的ingress服务，避免外部服务通过pod ip访问k8s内服务

2、应用作为服务消费方怎么做：

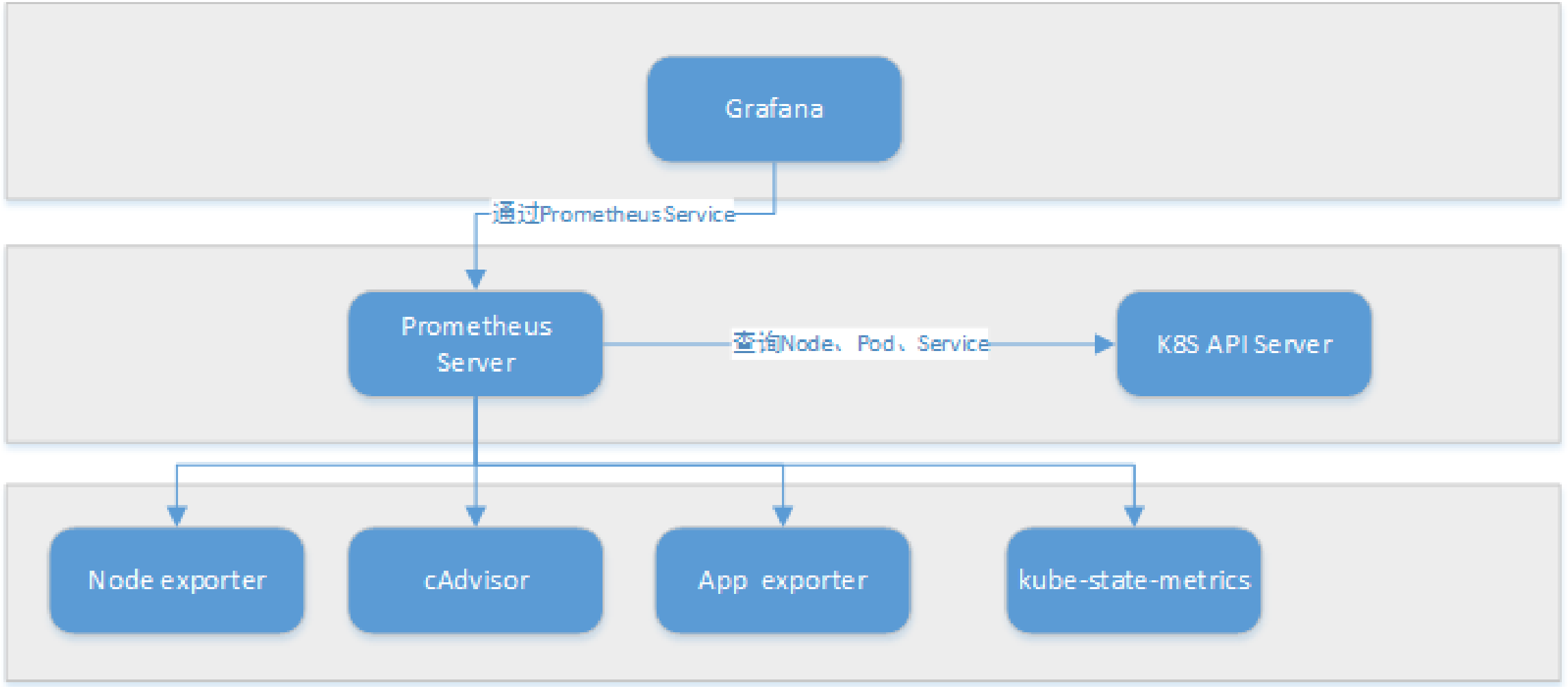
- 同一区域内不需要开墙的情况下，pod内服务可以直接通过服务提供方的虚拟机ip地址或者负载均衡地址调用外部服务进行服务消费
- 如果跨区等涉及到开墙的情况下，最好通过esg进行代理请求转发、或者应用需要做地址转换（首先将外部服务注册到endpoint，然后将外部地址转换为内部的endpoint服务，这样会涉及到定制开发）

3.4 系统配置在容器化后的处理方式



- 我们希望一套镜像能够同时被开发、测试以及生产环境使用
- 将不同环境的配置文件分不同目录打到镜像中，通过注入环境变量来加载（密码等敏感信息会暴露）
- 将环境相关的配置文件定义在k8s configmap中由运维进行创建，容器启动后只需加载configmap到容器指定目录即可（密码等敏感信息配置到cyberark）
- 由神兵管理不同环境configmap的配置值
- 通过建设分布式配置中心，系统启动后通过配置服务加载不同环境的配置

3.5 容器化后的系统监控



提示：
因为用到了基于K8S的自动发现节点、POD以及service；
需要事先确定命名空间是否有权限创建role以及cluster role；

3.6 容器化后的应用健康检查

```yaml
spec:
  containers:
  - name: liveness
    args:
    - /server
    image: gcr.io/google_containers/liveness
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
          - name: X-Custom-Header
            value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

提示：
有时候我们系统失去响应，但是应用进行并没有结束，这时候人为的处理应该是重启，健康检查就是做这样的任务

http健康检查

健康检查请求路径

自定义请求头

初始化延时检查

健康检查间隔

四、K8S的持续集成

4.1 环境准备

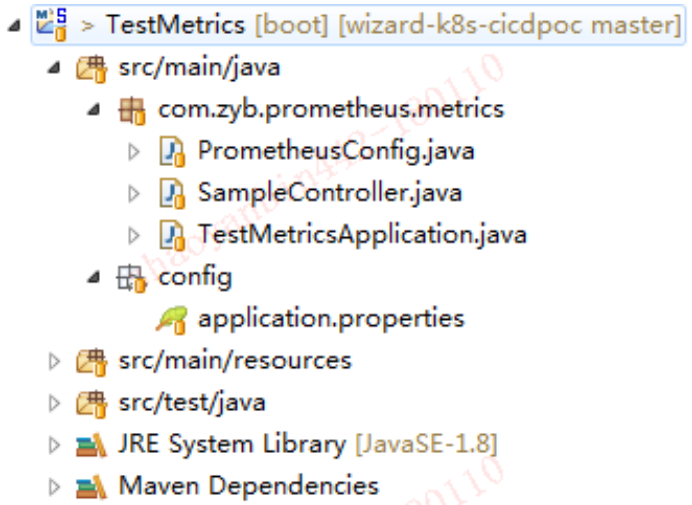
部署组件信息 vt test pipeline stg2 => metrics docker	
Name:	metrics_docker
中文名:	
组件类型:	common
使用部署用户:	deployop
部署平台目录:	/wls/deployop/systems/stg/vt_test_pipeline_stg2_metrics_docker
部署平台同步目录:	/wls/deployop/.systems_sync/stg/vt_test_pipeline_stg2_metrics_docker
服务器主机IP:	30.17.255.18
服务器目录:	
排除目录设置:	tmp,nfslink,deploybackup,resolve
部署命令配置[hotdeploy]:	curl -F "file" -F "zone=SZD" -F "version" http://30.16.232.66:8080/api/upload

- 服务器主机ip是镜像仓库的地址而不是实际运行服务的ip
- 部署命令中指定要部署的可用区例如：SZD为开发、SHB为测试

四、K8S的持续集成

4.2 准备镜像构建及应用编排配置

1. 创建演示项目



2. 简单的restful接口服务

```java
@RestController
public class SampleController {

    private static Random random = new Random();

    private static final Counter requestTotal = Counter.build()
        .name("my_sample_counter")
        .labelNames("status")
        .help("A simple Counter to illustrate custom Counters in

    @RequestMapping("/endpoint")
    public String endpoint() {
        if (random.nextInt(2) > 0) {
            requestTotal.labels("success").inc();
            return "success";
        } else {
            requestTotal.labels("error").inc();
            return "error";
        }

    }
```

3. 准备dockerfile构建镜像

```
# Pull base image
# ---------------
FROM hub.yun.paic.com.cn/official/jdk:8

# Maintainer
# ----------
MAINTAINER  zhaoyanbin<zhaoyanbin442>

RUN mkdir -p /opt/javametrics/

COPY TestMetrics-0.0.1-SNAPSHOT.jar launch.sh  /opt/javametrics/

# Install and configure Oracle JDK
# --------------------------------
WORKDIR /opt/javametrics/

CMD [ "bash" , "launch.sh" ]
```

4. 准备应用编排yaml

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  labels:
    caas_service: javametrics
  name: javametrics
  namespace: szd-0879950f
spec:
  replicas: 1
  selector:
    matchLabels:
      caas_service: javametrics
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        caas_service: javametrics
    spec:
      containers:
      - image: hub.yun.paic.com.cn/cicd/wizard-k8s-cicdpoc1.0.0/javametrics:IMAGE_VERSION
```

deployment

四、K8S的持续集成

4.3 配置神兵流水线

源码地址

地址#1

http://10.11.112.66/docker-container-scripts/wizard-k8s-cicdpoc.git

配置源码地址

自动编译间隔

编译类型 deployflow

deployflow文件 deployflow.properties

部署流水线模板 Default

应用版本号 wizard-k8s-cicdpoc1.0.0

编译前脚本

配置编译及部署参数

移交脚本

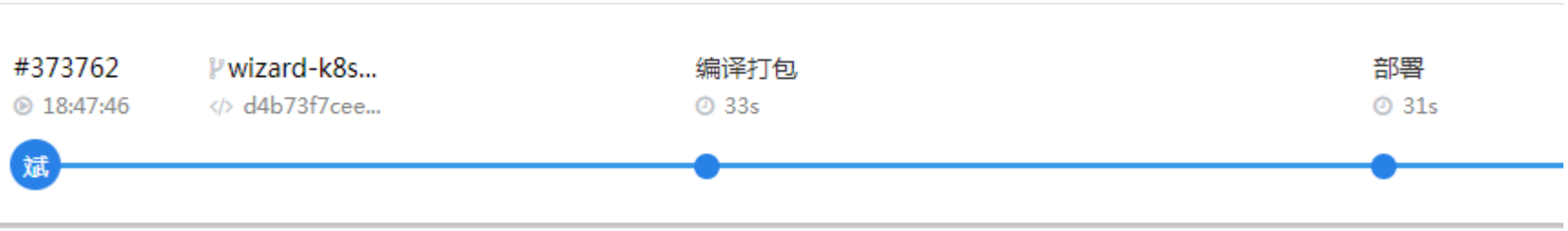
```
cp TestMetrics/target/TestMetrics*.jar ./dockerfile/
chmod +x ./dockerfile/*.sh
cd dockerfile
docker build -t javametrics .
cd ..
mkdir -p rel
cp deploy_desc/* rel/
```

配置移交脚本

4.4 验证持续集成

```
[root@SZD-L0077283 ~]# kubectl get ingress javametrics
Error from server (NotFound): ingresses.extensions "javametrics" not found
[root@SZD-L0077283 ~]# kubectl get svc javametrics
Error from server (NotFound): services "javametrics" not found
[root@SZD-L0077283 ~]# kubectl get pod javametrics
Error from server (NotFound): pods "javametrics" not found
[root@SZD-L0077283 ~]# kubectl get deployment javametrics
Error from server (NotFound): deployments.extensions "javametrics" not found
[root@SZD-L0077283 ~]#
```

确保没有已经运行的服务

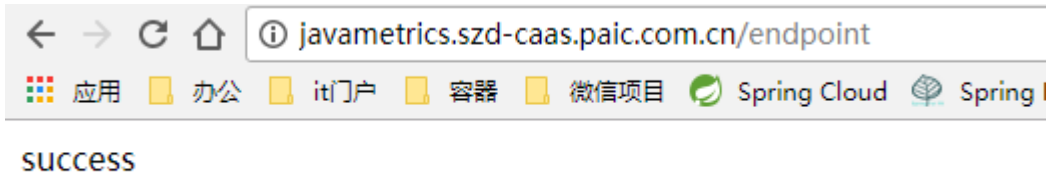


运行神兵构建流程

```
[root@SZD-L0077283 ~]# kubectl get deployment javametrics
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
javametrics   1         1         1            0           38s
[root@SZD-L0077283 ~]# kubectl get svc javametrics
NAME          CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
javametrics   172.254.32.6   <nodes>       8888:30888/TCP   42s
[root@SZD-L0077283 ~]# kubectl get ingress javametrics
NAME          HOSTS                                ADDRESS   PORTS     AGE
javametrics   javametrics.szd-caas.paic.com.cn               80        47s
[root@SZD-L0077283 ~]# kubectl get pod javametrics
```

后台查看部署的服务

提示：
Caas 平面界面目前还不能自动更新服务信息与状态



通过负载均衡入口验证服务

提示：
同时演示修改程序代码后通过神兵更新服务的场景

五、有用的参考资料

参考资料：

K8s官方文档：https://kubernetes.io/docs/home/

K8s中文社区：https://www.kubernetes.org.cn/

Docker one社区：http://dockone.io/

Docker info社区：http://www.dockerinfo.net/

配置Pod的liveness和readiness探针：https://www.kubernetes.org.cn/2362.html

持续集成示例项目git地址：http://10.11.112.66/docker-container-scripts/wizard-k8s-cicdpoc

Cloud Man博客：https://my.oschina.net/u/2397560/home

同呼吸、共命运、心连心

中国平安 PING AN

保险 · 银行 · 投资