# 微服务

## 发布微服务

Docker create image

Build：docker build -t ndong211/my:v1 .（注意这里有点）

K8s创建示例

Pod：kubectl run my-test --image=ndong211/my:v1 --port=8080

Expose service：kubectl expose deployment my-test --type=NodePort

查看

kubectl get pods --selector="run=<my-test> " --output=wide

## 集群管理

### k8s集群管理

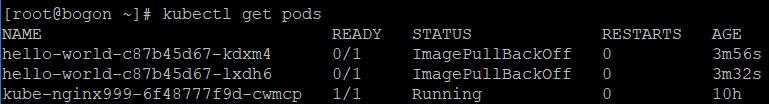
get

-o/output=json/yaml/wide|

#### 管理pods

查看pod

kubectl get pods [--all-namespaces]



删除：kubectl delete pods [pod-name]

Describe：kubectl describe pod <pod-name>

创建pod

kubectl run <pod-nane> --image=IMAGE [--port=port] [--replicas=NUM]

--replicas 副本数量

示例：

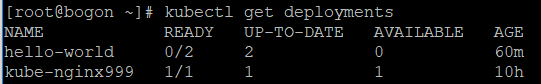
kubectl run nan-my1 --image=nan/my:v1 --port=8080

创建过程：kubectl http--> kube-apiserver --> deployment 持久化到etcd🡪 创建ReplicaSet和Pod，每个 Pod 都被调度到合适的节点 🡪 Kubelet按PodSpec定义，创建并启动容器

#### 管理Deployment

查看

kubectl get deploy/deployments [name]



* 创建：kubectl create deployment NAME --image=image [--dry-run]
* 删除：kubectl delete deployments <deploy-name>
* 扩容：kubectl scale deployment nan-my1 --replicas 2
* 更新镜像：kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
* 回滚：kubectl rollout undo deployment/nginx-deployment
* 显示 ReplicaSet 的信息

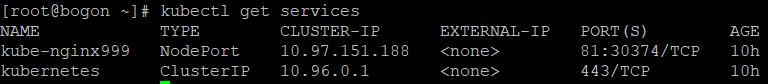
kubectl get replicasets/rs

kubectl describe replicasets

#### 管理Service

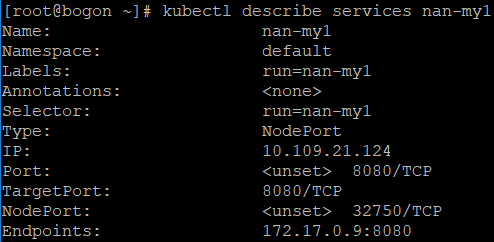
查看service

kubectl get svc/services



删除：kubectl delete services <name>

descripbe：kubectl describe services [label]



发布service

expose <-f fileName | type name> [--port=port] [--target-port=number-or-name] [--type=type]

target-port 容器端口

将资源暴露为新的Kubernetes Service，资源包括：

pod（po），service（svc），replication controller（rc），deployment（deploy），replica set（rs）

示例：

1. 将pod暴露给外网

kubectl expose deployment nan-my1 --type=NodePort

2.为rc创建service，并通过Service的80端口转发至容器的8000端口上。

kubectl expose rc nginx --port=80 --target-port=8000

3.由“nginx-controller.yaml”中指定的type和name标识的RC创建Service

kubectl expose -f nginx-controller.yaml --port=80 --target-port=8000

minikube查看service

minikube service kube-nginx999 [–url] [-n NAMESPACE]



#### 查看logs

查看指定pod的日志

kubectl logs <pod\_name>

kubectl logs -f <pod\_name> #类似tail -f

指定容器

kubectl logs <pod\_name> -c <container\_name>

Docker日志：docker logs <container\_id>

minikube查看日志：minikube logs

#### kubectl create

create -f fileName

通过配置文件名或stdin创建一个集群资源对象。支持json和yaml格式的文件。

示例：

kubectl create -f ./pod.json

#### 查看集群

kubectl cluster-info

/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

查看组件状态

kubectl get cs

kubectl get ep/endpoints

查看未授权的 CSR 请求

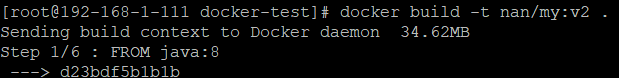
kubectl get csr (Certificate Signing Request)

### Docker

#### 3.1.1 镜像

创建镜像

docker build -t ndong211/my:v1 .（注意这里有点）



* --t/tag: 镜像的名字及标签
* 最后的 . 代表在当前目录下寻找 dockerfile,也可-f指定位置。

Dockerfile

FROM java:8

VOLUME /tmp

COPY ./my-1.0-SNAPSHOT.jar my.jar

RUN bash -c "touch /my.jar"

EXPOSE 8080

ENTRYPOINT ["java","-jar","my.jar"]

~

查看镜像

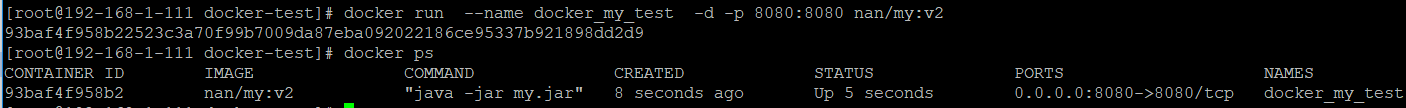
docker images [name]



#### 3.1.2 容器

创建容器

docker run --name names -d -p 80:80 repository:tag



* -d 后台运行
* -p 端口映射 -P容器端口随机

Docker run 用镜像创建容器 + docker start。

启动容器

启动：docker start/restart container\_id

停止：docker stop container\_id或container\_name

删除：docker rm container\_id

管理容器

logs

查看内部标准输出 docker logs -f bf08b7f2cd89

管理容器 docker exec -it docker\_nginx\_v1 bash

ps

docker ps [-a]

container id image command port names

d11a0358e403 nginx:v1 "nginx -g 'daemon ..." 0.0.0.0:88->80/tcp docker\_nginx\_v2

查看容器信息

docker inspect container\_id

#### 3.1.3 docker hub

[Reference](https://yeasy.gitbooks.io/docker_practice/repository/dockerhub.html)

docker pull/push <image-name>

push示例

docker push ndong211/my:v2

docker search +name

# 第二章DevOps

## 2.1 kubernetes

[Reference](https://jimmysong.io/kubernetes-handbook/concepts/deployment.html)

### 管理对象

Objects

|  |  |
| --- | --- |
| 类别 | 名称 |
| 资源对象 | Pod、ReplicaSet、ReplicationController、Deployment、StatefulSet、DaemonSet、Job、CronJob、HorizontalPodAutoscaling、Node、Namespace、Service、Ingress、Label、CustomResourceDefinition |
| 存储对象 | Volume、PersistentVolume、Secret、ConfigMap |
| 策略对象 | SecurityContext、ResourceQuota、LimitRange |
| 身份对象 | ServiceAccount、Role、ClusterRole |

* Pod：一组关联的容器集合
* Namespace：对一组资源和对象的抽象集合
* Node：Pod真正运行的主机。K8s为了管理Pod，
* service：对一组提供相同功能的Pods的抽象
* DaemonSet 守护进程集。DaemonSet保证在特定或所有Node节点上都运行一个Pod实例，常用来部署一些集群的日志采集、监控或者其他系统管理应用。典型的应用包括:

日志收集，比如fluentd，logstash等

系统监控，比如Prometheus Node Exporter，collectd等

系统程序，比如kube-proxy, kube-dns, glusterd, ceph，ingress-controller等

* Ingress：Kubernetes中的负载均衡我们主要用到了以下两种机制：

Service：使用Service提供集群内部的负载均衡，Kube-proxy负责将service请求负载均衡到后端的Pod中

Ingress Controller：使用Ingress提供集群外部的负载均衡

#### Node

Node是kubernetes集群的工作节点，可以是物理机也可以是虚拟机

Node的状态

Node包括如下状态信息：

* Address

HostName：可以被kubelet中的--hostname-override参数替代。

ExternalIP：可以被集群外部路由到的IP地址。

InternalIP：集群内部使用的IP，集群外部无法访问。

* Condition

OutOfDisk：磁盘空间不足时为True

Ready：Node controller 40秒内没有收到node的状态报告为Unknown，健康为True，否则为False。

MemoryPressure：当node有内存压力时为True，否则为False。

DiskPressure：当node有磁盘压力时为True，否则为False。

* Capacity

CPU

内存

可运行的最大Pod个数

* Info：节点的一些版本信息，如OS、kubernetes、docker等

#### 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，让一个namespace中的service可以访问到其他的namespace中的服务

kubectl get namespaces



对象创建

创建K8s的Object可用yaml文件配置。

示例

kubectl create -f nginx-deployment.yaml --record

1. apiVersion: apps/v1beta1
2. kind: Deployment
3. metadata:
4. name: nginx-deployment
5. spec:
6. replicas: 3
7. template:
8. metadata:
9. labels:
10. app: nginx
11. spec:
12. containers:
13. - name: nginx
14. image: nginx:1.7.9
15. ports:
16. - containerPort: 80

* apiVersion - 创建该对象所使用的 Kubernetes API 的版本
* kind - 对象的类型
* metadata - 帮助识别对象唯一性的数据，包括一个 name 字符串、UID 和可选的 namespace
* spec.replicas字段代表了受此RC管理的Pod，需要运行的副本数。
* template模块用于定义Pod，包括Pod的名字，Pod拥有的label以及Pod中运行的应用。

#### Pod概览

Pod封装应用容器(一个或多个)，存储、独立的网络IP，Pod代表着部署的一个单位：k8s中应用的一个实例，可能由一个或者多个容器组合在一起共享资源。

K8s管理的是Pod而不是直接管理容器，集群中Pod有如下两种使用方式：

* 一个Pod中运行一个容器(常用)。
* 在一个Pod中同时运行多个容器。

Pause容器

又叫Infra容器。主要为每个业务容器提供以下功能：

* 在pod中担任Linux命名空间共享的基础；
* 启用pid命名空间，开启init进程。
* 解析：pause容器将内部的80端口映射到宿主机的8880端口

### K8s架构

k8s架构



Master架构



Node架构



每个Node节点上至少需要运行container runtime（Docker）、kubelet和kube-proxy服务。

Kubelet

集群每个 Node 节点都启动一个Kubelet服务进程。处理 Scheduler下发到本节点任务/管理Pod生命周期。

这意味着它将处理 Pod 与 Container Runtime之间所有的转换逻辑，包括挂载卷、容器日志、垃圾回收以及其他重要事件。

可以把 Kubelet当成一种特殊的 Controller，它每隔20秒向kube-apiserver查询 Pod，过滤 NodeName 与自身所在节点匹配的 Pod 列表。

Kubelet启动容器过程

拉取容器的镜像

通过 CRI 创建容器

启动容器

### 控制器

Kubernetes中内建了很多controller（控制器），这些相当于一个状态机，用来控制Pod的具体状态和行为。

#### Deployment

Deployment 为 Pod 和 ReplicaSet 提供了一个声明式定义(declarative)方法，用来替代以前的ReplicationController 来方便的管理应用。典型的应用场景包括：

定义Deployment来创建Pod和ReplicaSet

* 滚动升级和回滚应用
* 扩容和缩容
* 暂停和继续Deployment

ReplicaSet

#### Horizontal Pod Autoscaling

[Reference](https://jimmysong.io/kubernetes-handbook/concepts/horizontal-pod-autoscaling.html)

应用的资源使用率通常都有高峰和低谷的时候，如何削峰填谷，提高集群的整体资源利用率，让service中的Pod个数自动调整呢？这就有赖于Horizontal Pod Autoscaling了，顾名思义，使Pod水平自动缩放。这个Object（跟Pod、Deployment一样都是API resource）也是最能体现kubernetes之于传统运维价值的地方，不再需要手动扩容了，终于实现自动化了，还可以自定义指标，没准未来还可以通过人工智能自动进化呢！

### 服务发现

Kubernetes中为了实现服务实例间的负载均衡和不同服务间的服务发现，创造了Serivce对象，同时又为从集群外部访问集群创建了Ingress对象。

#### Service

#### Ingress

### 存储

### 集群安全

用户

k8s 中有2种用户，一般用户和serviceAccount。

* 一般用户: 给集群外部用户使用，例如集群管理员通过 kubectl这个客户端来操作集群，使用的就是一般用户，这个用户通过 k8s 的 RBAC 权限系统获得相应权限。
* serviceAccount: 给集群内的资源使用，例如pod访问apiserver。每个pod都有一个 default serviceAccount。

Kublet的认证授权

* 匿名访问，调用 kubelet API 的能力应受到限制
* 启用 X509 客户端证书身份验证：
* 启用 API bearer token（包括 service account token）。

kubelet bootstrap

kubelet 使用低权限的 bootstrap token 跟 api server 建立连接后，要能够自动向 api server 申请自己的证书，并且 api server 要能够自动审批证书。

kubectl create clusterrolebinding kubelet-bootstrap --user=kubelet-bootstrap --clusterrole=system:node-bootstrapper

给bootstrap token代表的用户 system:bootstrap:abcdef 赋予 clusterole certificatesigningrequests.certificates.k8s.io/nodeclient 和 system:node-bootstrapper，让该用户可以访问 csr API 以及自动审批其创建的 csr

给新的 work node 代表的用户 system:node:test-node 赋予 clusterrole system:certificates.k8s.io:certificatesigningrequests:selfnodeclient，让它发送的证书 renew 的请求能被自动审批

kubectl 审批

# 通过 CSR 请求

kubectl certificate approve <name>

kubectl certificate deny <name>

命令扩展：

kubectl delete csr 节点名称 #删除单个节点的请求

kubectl delete csr --all #删除所有节点请求

kubectl delete nodes node名称 #删除加入的节点

kubectl delete nodes --all #删除所有节点

RoleBinding

如果想限制user用户行为，需要使用 RBAC创建角色绑定以将该用户的行为限制在某个或某几个 namespace 空间范围内，例如

赋予用户kubelet-bootstrap角色

kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-bootstrapper --user=kubelet-bootstrap

kubectl get clusterrolebindings

否则报异常node exception：

server.go:271] failed to run Kubelet: cannot create certificate signing request: User "kubelet-bootstrap" cannot create certificatesigningrequests.certificates.k8s.io at the cluster scope. (post certificatesigningrequests.certificates.k8s.io)



roles

kubectl get clusterroles



RBAC-基于角色的访问权限控制

-authorization-mode=RBAC启动API Server

## 2.2 Docker

### Docker Remote API

远程管理docker,允许通过unix socket通信操作Docker daemon，也可以通过HTTP调用其Rest API，完成容器的查询，创建、销毁等等操作。

# 第三章 软件安装

### 3.1 Docker

#### 安装

[Reference](https://www.runoob.com/docker/centos-docker-install.html)

yum install docker-ce docker-ce-cli containerd.io

启动：systemctl start docker

vi /lib/systemd/system/docker.service

仓库地址配置/etc/yum.repos.d/docker-ce.repo

Uninstall

yum remove -y docker docker-latest docker-ce docker-common \

docker-selinux container-selinux \

docker-engine \

docker-ce-cli \

docker-client docker-client-latest \

docker-logrotate docker-latest-logrotate

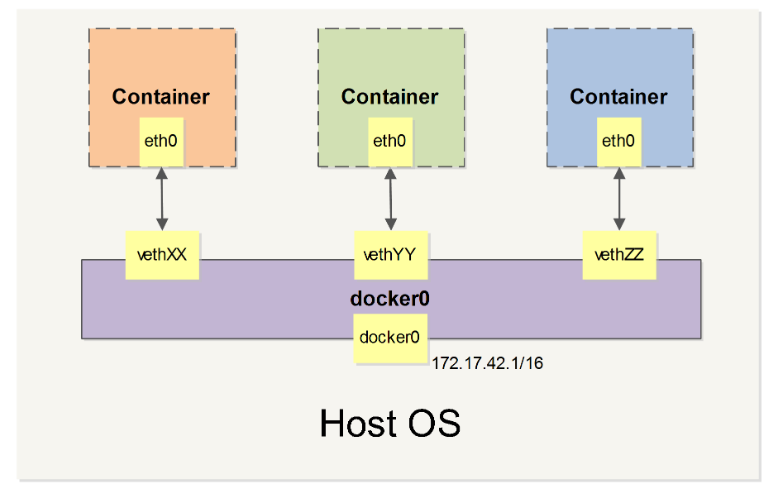
containerd.io

delete conf

/var/lib/docker

/etc/docker

#### 3.1.3 网络



### 6.3.2 k8s

安装

[Reference](https://kubernetes.io/docs/tasks/tools/install-kubectl/)

配置对多集群的访问:[Reference](https://kubernetes.io/zh/docs/tasks/access-application-cluster/configure-access-multiple-clusters/)

kubectl version

#### kubeadm

镜像问题

docker pull mirrorgooglecontainers/kube-proxy:v1.14.0

docker tag /mirrorgooglecontainers/kube-proxy:v1.14.0 k8s.gcr.io/kube-proxy:v1.14.0

docker rmi mirrorgooglecontainers/kube-proxy:v1.14.0

#### Minikube

[Reference](https://yq.aliyun.com/articles/221687)

安装

**1.下载minikube**

curl -Lo minikube http://kubernetes.oss-cn-hangzhou.aliyuncs.com/minikube/releases/v1.4.0/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

启动

minikube start --vm-driver=none

查看状态：minikube status

minikube dashboard

kubectl proxy --port=8001 --address='192.168.0.5' --accept-hosts='^.\*' &

更新

如需更新minikube，需要更新 minikube 安装包

minikube delete 删除现有虚机，删除 ~/.minikube 目录缓存的文件

vm driver

~/.minikube/machines/minikube/config.json

#### 3.2.2 kubectl

kubectl version：Print the client and server version information

Client Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.0", GoVersion:"go1.7.5", Platform:"linux/amd64"}

Server Version: version.Info{Major:"1", Minor:"5", GitVersion:"v1.5.2", GitTreeState:"clean", GoVersion:"go1.7.4", Compiler:"gc", }

##### kubeconfig

kubeconfig文件记录k8s集群的各种信息，对集群构建非常重要。

* kubectl从~/.kube/config，即kubectl的kubeconfig文件中获取访问kube-apiserver的地址，证书和用户名等信息
* kubelet/kube-proxy等在Node上的程序进程同样通过bootstrap.kubeconfig和kube-proxy.kubeconfig上提供的认证与授权信息与Master进行通讯

查询集群地址和凭证

kubectl config view

1. apiVersion: v1
2. clusters:
3. - cluster:
4. certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSU....
5. server: https://192.168.0.110:6443
6. name: kubernetes
7. contexts:
8. - context:
9. cluster: kubernetes
10. user: kubelet-bootstrap
11. name: default
12. current-context: default
13. kind: Config
14. preferences: {}
15. users:
16. - name: kubelet-bootstrap
17. user:
18. token: 5123398c0116eb8617398e4c38c3be

certificate-authority-data 是 ca.pem 的base64加密的内容

client-key-data:

kubectl config get-contexts

##### Volume存储数据卷

exception

Orphaned pod "cdb27b68-9e8d-4708-8179-b5579dab861f" found, but volume paths are still present on disk.

rm -rf /var/lib/kubelet/pods/\*

#### 手动安装

##### flannel

master和node上都要安装Flannel网络

配置

vi /usr/lib/systemd/system/flanneld.service

vi /etc/sysconfig/flanneld

etcdctl --endpoints=${ETCD\_ENDPOINTS} put /kubernetes/network/config '{"Network":"'${CLUSTER\_CIDR}'", "SubnetLen": 24, "Backend": {"Type": "vxlan"}}'

etcdctl --endpoints=${ETCD\_ENDPOINTS} get /kubernetes/network/config

--cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/etcd/ssl/etcd-key.pem --cert=/etc/etcd/ssl/etcd.pem

通过http向etcd增加key

curl -X PUT http://192.168.0.110:2379/v2/keys/kubernetes/network/config -d value='{"Network":"172.30.0.0/16", "SubnetLen": 24, "Backend": {"Type": "vxlan"}}'

##### master节点

安装kubernetes： yum -y install kubernetes

kube-apiserver

vi /usr/lib/systemd/system/kube-apiserver.service

vi /etc/kubernetes/apiserver

systemctl daemon-reload

systemctl restart kube-apiserver.service

kube-controller-manager

vi /usr/lib/systemd/system/kube-controller-manager.service

systemctl daemon-reload

systemctl restart kube-controller-manager.service

查看节点

curl http://192.168.0.110:8080/api/v1/nodes

kube-scheduler

vi /usr/lib/systemd/system/kube-scheduler.service

systemctl enable kube-scheduler

##### Node节点

1.安装kubernetes

yum -y install kubernetes&vi /etc/kubernetes/config

* KUBE\_LOGTOSTDERR="--logtostderr=true"
* KUBE\_LOG\_LEVEL="--v=0"
* KUBE\_ALLOW\_PRIV="--allow-privileged=false"
* KUBE\_MASTER="--master=http://k8s-master:8080"

kubelet

vi /usr/lib/systemd/system/kubelet.service

vi /etc/kubernetes/kubelet

* KUBELET\_ADDRESS="--address=0.0.0.0"
* KUBELET\_HOSTNAME="--hostname-override=k8s-node-1"
* KUBELET\_API\_SERVER="--api-servers=http://k8s-master:8080"
* KUBELET\_POD\_INFRA\_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/rhel7/pod-infrastructure:latest" #确认此镜像存在，否则pod无法启动
* KUBELET\_ARGS=""

/etc/kubernetes/kubelet.conf

–client-ca-file

3.启动服务

systemctl daemon-reload

systemctl restart kubelet

kube-proxy

/usr/lib/systemd/system/kube-proxy.service

Vi

#### 集群安全设置

[Reference](https://o-my-chenjian.com/2017/04/25/Security-Settings-Of-K8s/)

#### etcd

etcdctl version

ETCDCTL\_API=3

配置

vi /usr/lib/systemd/system/etcd.service

vi /etc/etcd/etcd.conf

#[Member]

ETCD\_DATA\_DIR：etcd数据保存目录

ETCD\_LISTEN\_CLIENT\_URLS：供外部**客户端**使用的url

ETCD\_ADVERTISE\_CLIENT\_URLS：暴露给外部**客户端**使用的url

ETCD\_NAME：etcd实例名称

#[Clustering]

ETCD\_LISTEN\_PEER\_URLS：集群内部通信使用的URL

ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS：广播给集群内其他成员访问的URL

ETCD\_INITIAL\_CLUSTER：初始集群成员列表

ETCD\_INITIAL\_CLUSTER\_TOKEN：集群的名称

ETCD\_INITIAL\_CLUSTER\_STATE：初始集群状态，new为新建集群

rm -rf /var/lib/etcd/default.etcd

systemctl daemon-reload

systemctl restart etcd

etcdctl cluster-health

配置TSL

etcdctl member list --endpoints=https://192.168.0.111:2379 --cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/etcd/ssl/etcd-key.pem --cert=/etc/etcd/ssl/etcd.pem

CMD

* -w, --write-out="simple" 输出格式(fields, json, protobuf, simple, table)
* --endpoints=[127.0.0.1:2379] ：gRPC endpoints

Restful api

curl http://192.168.0.110:2379/v2/keys/

查看集群

etcdctl endpoint status

etcdctl [--write-out=table] endpoint health

etcdctl member list

etcdctl endpoint health --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/ssl/ca.pem --key=/etc/kubernetes/ssl/etcd-key.pem --cert=/etc/kubernetes/ssl/etcd.pem

示例：

etcdctl --endpoints=http://192.168.0.110:2379 member list -w table

集群成员

跟集群成员相关的命令如下：

member add Adds a member into the cluster

member remove Removes a member from the cluster

member update Updates a member in the cluster

member list Lists all members in the cluster

示例

etcdctl member list -w table



CRUD

* 增：etcdctl --endpoints=$ENDPOINTS put foo "Hello World!"
* 查：etcdctl [--endpoints=$ENDPOINTS] [--write-out="json"] get foo
* 删：etcdctl --endpoints=$ENDPOINTS del key

Failed to connect to apiserver: the server has asked for the client to provide credentials

Failed to connect to apiserver: Forbidden: "/healthz?timeout=1s

# todo

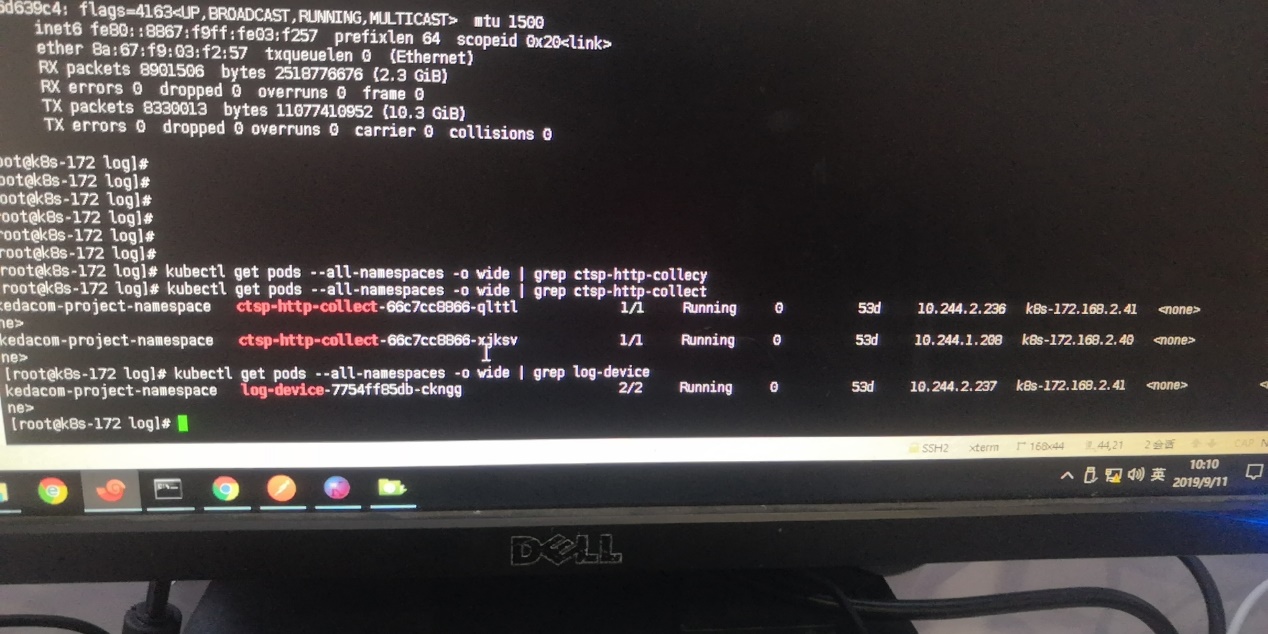
市局二类网服务器，查看设备日志的服务

Kubectl get pods --all-namespaces -o wide |grep ctsp-http-collect

kubectl –n kedacom-project-namespace logs ctsp-http-collect-66c7cc8866-qlttl > ctsp-http-collect-01.log

kubectl -n kedacom-project-namespace logs ctsp-http-collect-66c7cc8866-xjksv > ctsp-http-collect-02.log

kubectl get pods --all-namespaces |grep log-device



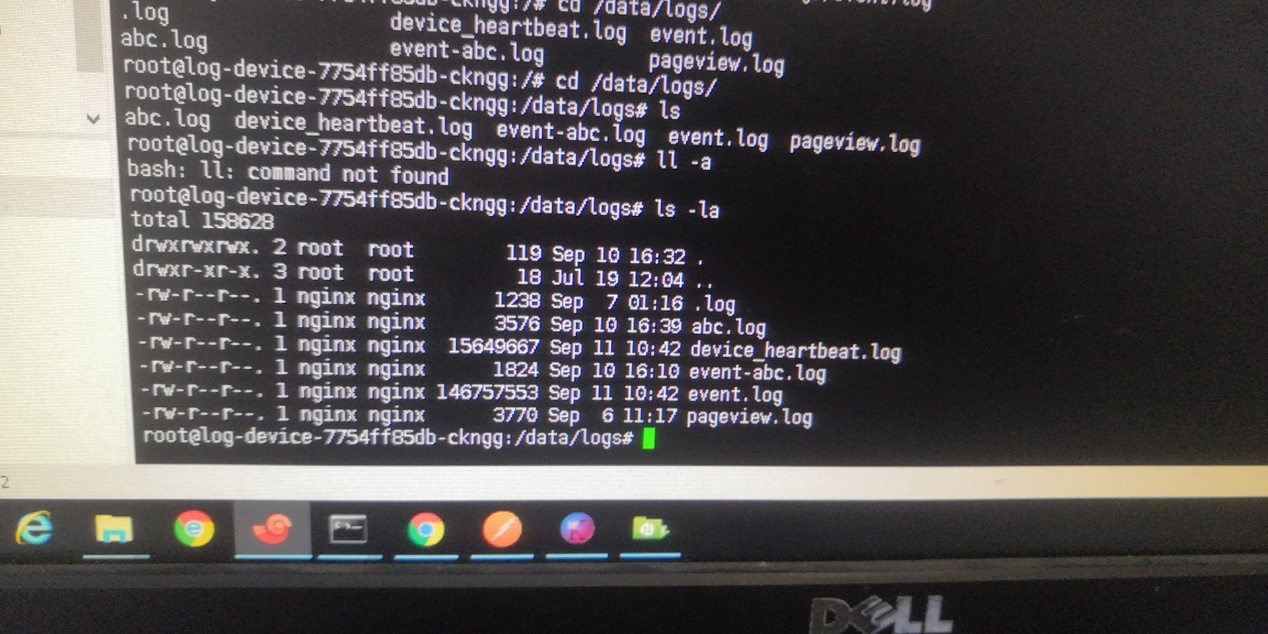
kubectl -n dolphin exec -it log-device-7885d65c9b-9l2w8（此处为下图的名字） bash

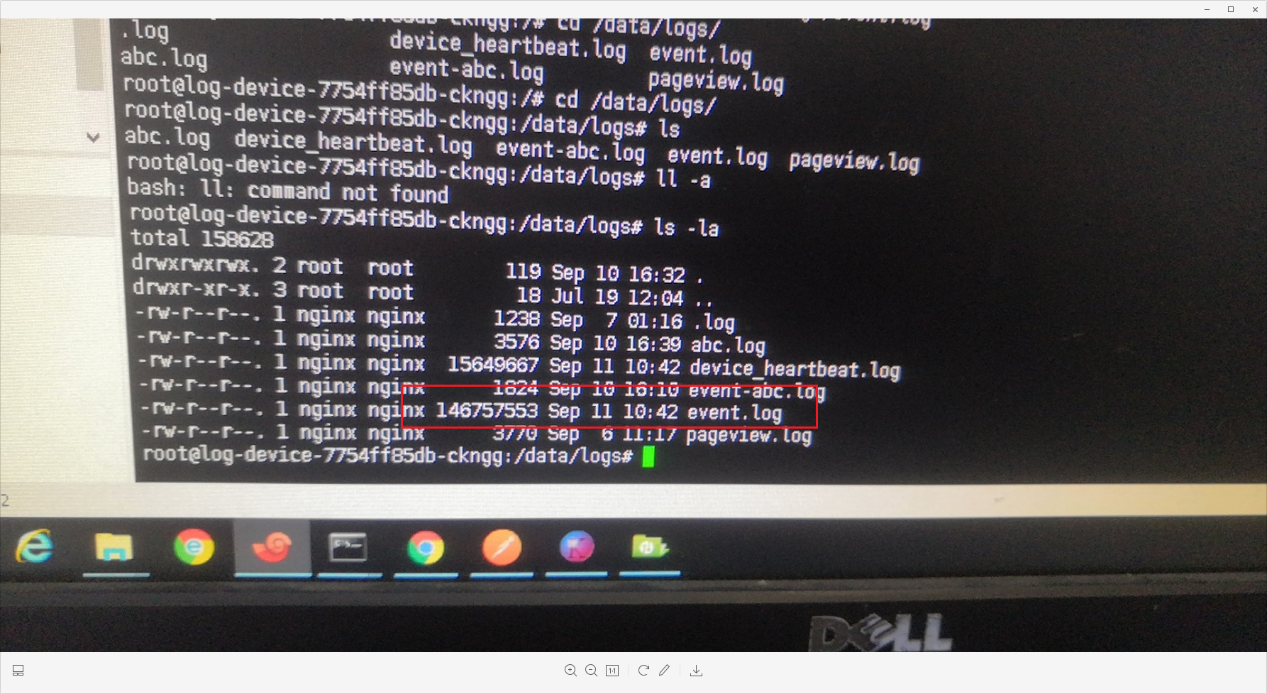
C:\Users\keda\AppData\Local\Temp\WeChat Files\10ce15cb149112aadf1e09a6d0c2bdb.png

进入容器之后

ls /data/logs/event.log -la

看下这个文件





kubectl -n dolphin exec -it log-device-7885d65c9b-9l2w8 bash -c flume-collect

用这个进flume服务 看下