一、Storage10%

Understand storage classes, persistent volumes

1. storage classes:

系統管理員根據需求與架構，創建對應的 StorageClas 物件，該物件也需要描述後端的儲存空間，但是不需要描述相對應的資源大小

使用者接下來根據需求創建 PVC, 而這些 PVC內描述我要使用特定的 StorageClass

kubernetes 觀察到有 PVC 想要使用 StorageClass 且發現該 StorageClass 物件存在，就會透過 StorageClass 的屬性以及 PVC 的需求創造一個完全符合該 PVC 需求的 PV

最後該 PVC 與該 PV 就可以綁定並且供 Pod 使用

apiVersion: storage.k8s.io/v1

kind: StorageClass  
metadata:  
 name: gluster-vol-default  
provisioner: kubernetes.io/glusterfs  
parameters:  
 resturl: "http://192.168.10.100:8080"  
 restuser: ""  
 secretNamespace: ""  
 secretName: ""  
allowVolumeExpansion: true

1. persistent volumes

**apiVersion**: v1

**kind**: PersistentVolume

**metadata**:

**name**: pv0003

**spec**:

**capacity**:

**storage**: 5Gi

**volumeMode**: Filesystem

**accessModes**:

- ReadWriteOnce

**persistentVolumeReclaimPolicy**: Recycle

**storageClassName**: slow

**hostpath:**

**path**: /tmp/data

Understand volume mode, access modes and reclaim policies for volumes

1. volume mode: FileSystem & Block
2. access mode:

ReadWriteOnce

請求到的該塊空間只能同時給一個節點使用，節點上的各種容器使用可以同時

進行讀取或是寫入的動作。

ReadOnlyMany

請求到的該塊空間可以同時給多個節點使用，但是節點上的各種容器使用都只能基於讀取這種沒有寫入的操作。

ReadWrtieMany

請求到的該塊空間可以同時給多個節點使用，且大家要進行讀取或是寫入等動作都是沒有問題的。

1. reclaim policies:

3.1. Retained: 回收策略 Retain 使得用户可以手动回收资源。当 PersistentVolumeClaim 对象 被删除时，PersistentVolume 卷仍然存在，对应的数据卷被视为"已释放（released）"。 由于卷上仍然存在这前一申领人的数据，该卷还不能用于其他申领。 管理员可以通过下面的步骤来手动回收该卷：

删除 PersistentVolume 对象。与之相关的、位于外部基础设施中的存储资产 （例如 AWS EBS、GCE PD、Azure Disk 或 Cinder 卷）在 PV 删除之后仍然存在。

根据情况，手动清除所关联的存储资产上的数据。

手动删除所关联的存储资产；如果你希望重用该存储资产，可以基于存储资产的 定义创建新的 PersistentVolume 卷对象

3.2. Recycled: 如果下层的卷插件支持，回收策略 Recycle 会在卷上执行一些基本的 擦除（rm -rf /thevolume/\*）操作，之后允许该卷用于新的 PVC 申领。警告： 回收策略 Recycle 已被废弃。取而代之的建议方案是使用动态供应。

3.3.Deleted:

对于支持 Delete 回收策略的卷插件，删除动作会将 PersistentVolume 对象从 Kubernetes 中移除。

Understand persistent volume claims primitive

**apiVersion**: v1

**kind**: PersistentVolumeClaim

**metadata**:

**name**: myclaim

**spec**:

**accessModes**:

- ReadWriteOnce

**volumeMode**: Filesystem

**resources**:

**requests**:

**storage**: 8Gi

Know how to configure applications with persistent storage

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: mypod

**spec**:

**containers**:

- **name**: myfrontend

**image**: nginx

**volumeMounts**:

- **mountPath**: "/var/www/html"

**name**: mypd

**volumes**:

- **name**: mypd

**persistentVolumeClaim**:

**claimName**: myclaim

二、Troubleshooting30%

Evaluate cluster and node logging

kubectl get events

Understand how to monitor applications

1. 查看Pod log

Kubectl logs –f <pod name> -c <container name>

Manage container stdout & stderr logs

docker logs 显示当前运行的容器的日志信息，内容包含 STOUT(标准输出) 和 STDERR(标准错误输出)。日志都会以 json-file 的格式存储于 /var/lib/docker/containers/<容器id>/<容器id>-json.log

Kubectl logs –f <pod name> -c <container name>

Troubleshoot application failure

1. can not resolve: 找dns 或者 service name 错误
2. connection refused: 找service 端口与pod端口配置不匹配，label不匹配
3. Access denied: 找env中账号密码不匹配问题(deploy & 相应pod)

使用到的命令：

kubectl get svc -n <namespace>

kubectl get ep <service-name>

Troubleshoot cluster component failure

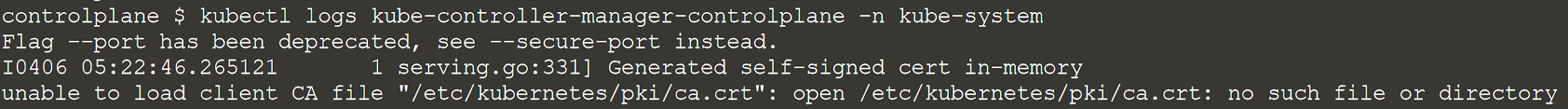
**Control plan failure**

Kubectl get pod –n kube-system 查看异常的POD

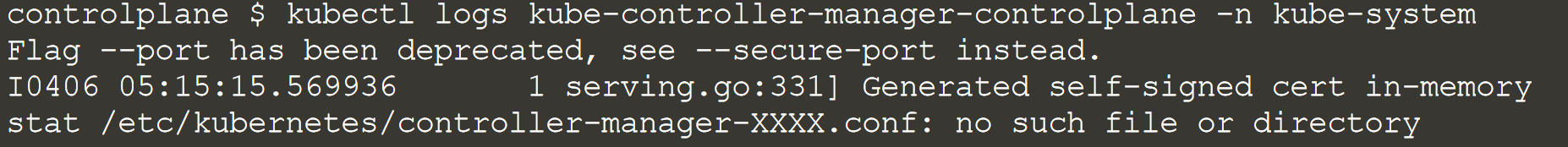
kubectl logs <pod-name> -n kube-system

1. 证书问题

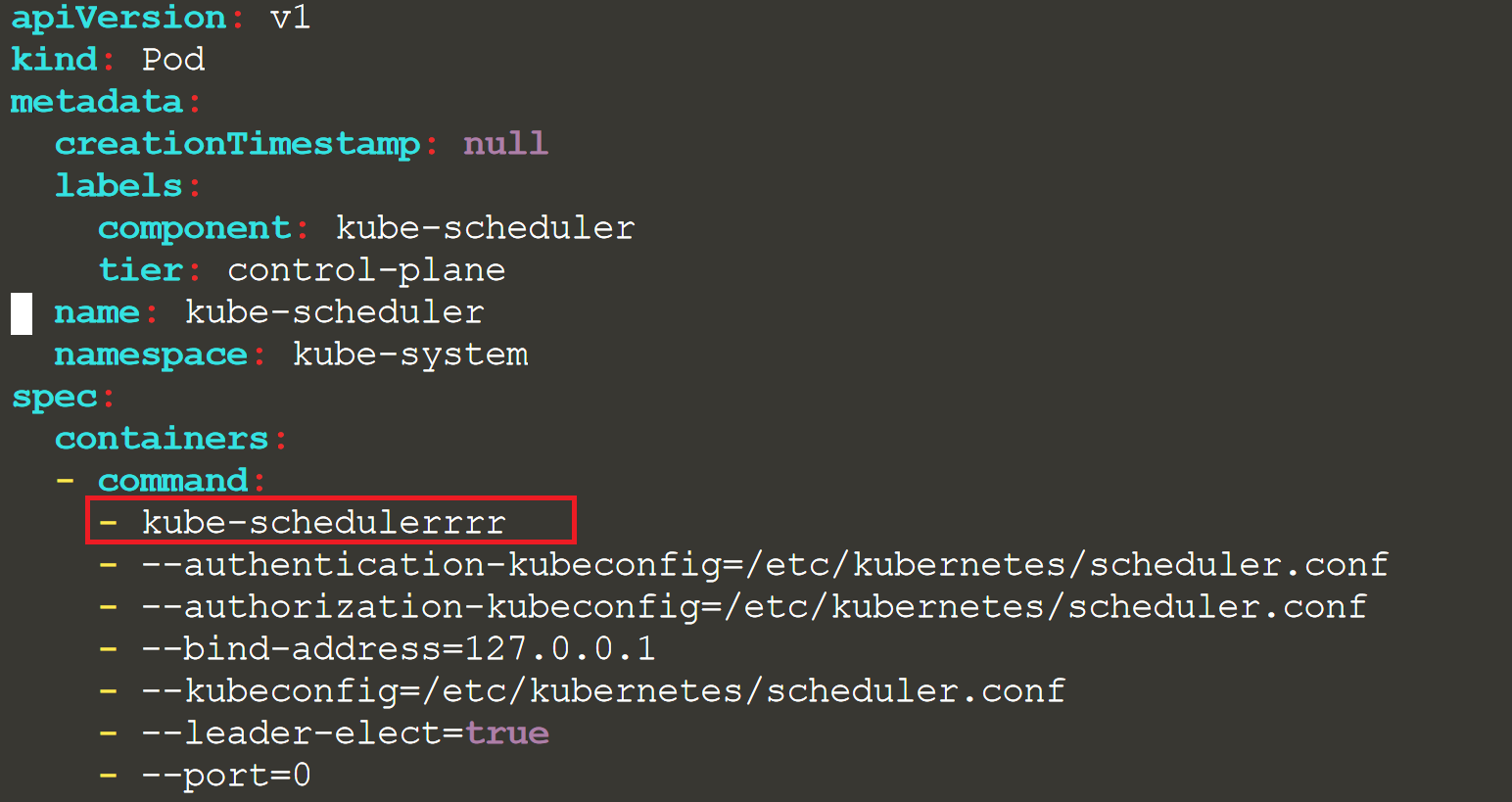
证书路径错误，volume挂载错误



1. 配置项名错误

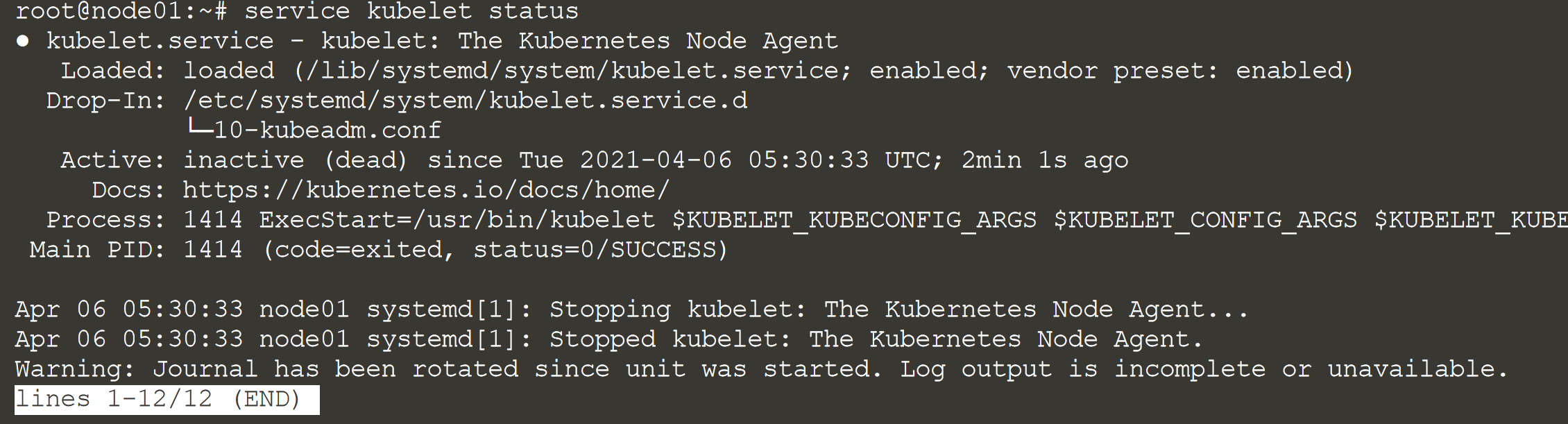


1. 参数异常



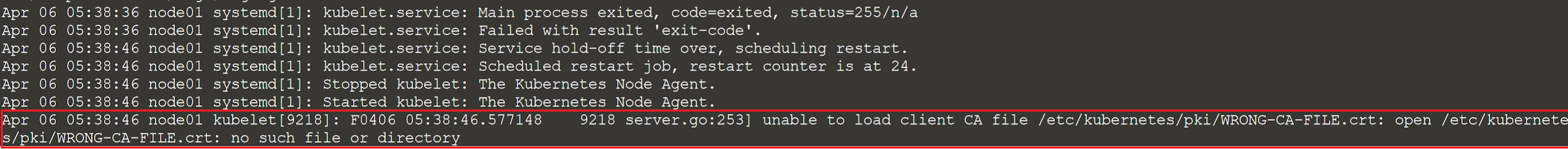
**Worker node failure**

1.node port 中kubelet 服务关闭

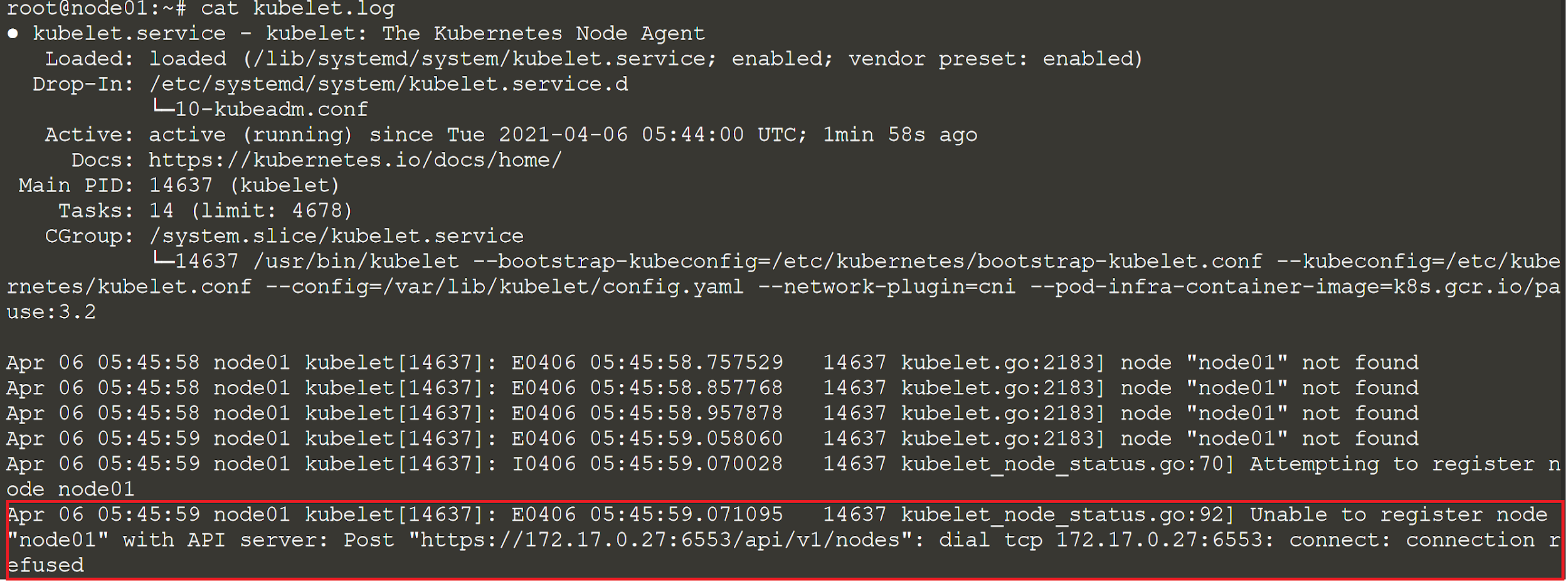


2.kubelet 证书异常导致服务关闭

journalctl -u kubelet.service

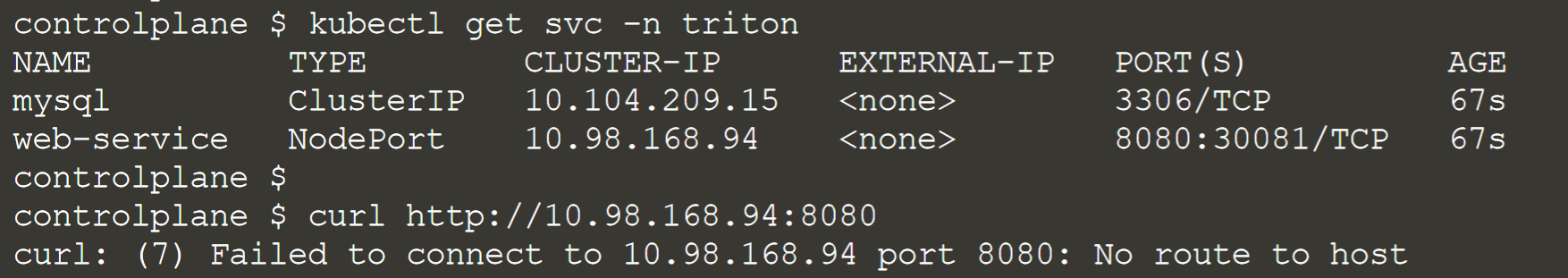


3. kubelet 配置api-server 错误导致服务异常



Troubleshoot networking

1. no cni: 特点 Failed to connect to xxxx No route to host



1. no proxy: Failed to connect to 10.98.131.136 port 8080: Connection refused
2. dns 问题: Readiness probe failed: Get "http://:8181/ready": dial tcp :8181: connect: connection refused

Workloads & Scheduling15%

Understand deployments and how to perform rolling update and rollbacks

1. kubectl scale deployment <deployments-name> --replicas==n –record
2. kubectl --record deployment.apps/nginx-deployment set image deployment.v1.apps/nginx-deployment nginx=nginx:1.16.1
3. kubectl rollout status deployment <deployment-name>
4. kubectl rollout undo deployment <deployment-name>
5. kubectl rollout history deployment <deployment-name>

Use ConfigMaps and Secrets to configure applications

1. Without ConfigMaps.

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: envar-demo

**labels**:

**purpose**: demonstrate-envars

**spec**:

**containers**:

- **name**: envar-demo-container

**image**: gcr.io/google-samples/node-hello:1.0

**env**:

- **name**: DEMO\_GREETING

**value**: "Hello from the environment"

1. ConfigMaps

Create

Kubectl create configmap <config-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>

**apiVersion**: v1

**kind**: ConfigMap

**metadata**:

**name**: game-demo

**data**:

*# property-like keys; each key maps to a simple value*

**player\_initial\_lives**: "3"

**ui\_properties\_file\_name**: "user-interface.properties"

Use configmaps

将configmap 放在 volum 中 挂载给pod

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: mypod

**spec**:

**containers**:

- **name**: mypod

**image**: redis

**volumeMounts**:

- **name**: foo

**mountPath**: "/etc/foo"

**readOnly**: **true**

**volumes**:

- **name**: foo

**configMap**:

**name**: myconfigmap

将configmap传入pod 环境变量

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: dapi-test-pod

**spec**:

**containers**:

- **name**: test-container

**image**: k8s.gcr.io/busybox

**command**: [ "/bin/sh", "-c", "env" ]

**envFrom**:

- **configMapRef**:

**name**: special-config

**restartPolicy**: Never

将configmap中的某值传入pod环境变量

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: dapi-test-pod

**spec**:

**containers**:

- **name**: test-container

**image**: k8s.gcr.io/busybox

**command**: [ "/bin/sh", "-c", "echo $(SPECIAL\_LEVEL\_KEY) $(SPECIAL\_TYPE\_KEY)" ]

**env**:

- **name**: SPECIAL\_LEVEL\_KEY

**valueFrom**:

**configMapKeyRef**:

**name**: special-config

**key**: SPECIAL\_LEVEL

1. Secrets

对于一些需要加密的配置，使用secrets替代

创建：

Kubctl create secret mysecret <secret-name> --from-literal=<key>=<value>

Value 需进行base64 encode

Echo –n “root” | base64 <==> echo –n “sdada==”|base64 --decode

**apiVersion**: v1

**kind**: Secret

**metadata**:

**name**: secret-sa-sample

**annotations**:

**kubernetes.io/service-account.name**: "sa-name"

**type**: kubernetes.io/service-account-token

**data**:

**extra**: YmFyCg==

绑定：

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: mypod

**spec**:

**containers**:

- **name**: mypod

**image**: redis

**volumeMounts**:

- **name**: foo

**mountPath**: "/etc/foo"

**readOnly**: **true**

**volumes**:

- **name**: foo

**secret**:

**secretName**: mysecret

Know how to scale applications

kubectl scale deployment <deployments-name> --replicas==n –record

Understand the primitives used to create robust, self-healing, application deployments

?????

Understand how resource limits can affect Pod scheduling

Container can not use more CPU & Memory resources than its limitation

If a pod tries to consume more memory than its limit , the pod will be terminated

若创建container不指定limit & request, 的default为512Mi(0.5 vCPU) CPU & 256Mi(1Mi = 1024 \* 1024 bytes)

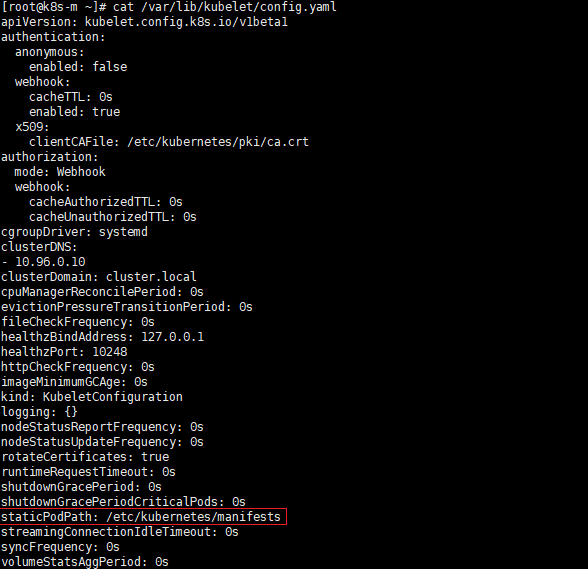
若只指定limit，则request = limit

若只指定request，则limit = default limit

\* Every container run in namespace must have its own memory limitation

\* Total amount memory used be be all containers in namespace must not exceed a specified limit

Awareness of manifest management and common templating tools



Cluster Architecture, Installation & Configuration25%

Manage role based access control (RBAC)

Cluster Role:

Service Account

apiVersion: v1

kind: ServiceAccount

metadata:

name: build-robot

Role: 需要指定该Role所属的命名空间

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: Role

**metadata**:

**namespace**: default

**name**: pod-reader

**rules**:

- **apiGroups**: [""] *# "" indicates the core API group*

**resources**: ["pods"]

**verbs**: ["get", "watch", "list"]

Cluster Role: 不受命名空间限制

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: ClusterRole

**metadata**:

*# "namespace" omitted since ClusterRoles are not namespaced*

**name**: secret-reader

**rules**:

- **apiGroups**: [""]

*#*

*# at the HTTP level, the name of the resource for accessing Secret*

*# objects is "secrets"*

**resources**: ["secrets"]

**verbs**: ["get", "watch", "list"]

Role-Binding: 将角色中定义的权限赋予一个或者一组用户，RoleBinding 在指定的名字空间中执行授权

**apiVersion**: rbac.authorization.k8s.io/v1

*# This role binding allows "jane" to read pods in the "default" namespace.*

*# You need to already have a Role named "pod-reader" in that namespace.*

**kind**: RoleBinding

**metadata**:

**name**: read-pods

**namespace**: default

**subjects**:

*# You can specify more than one "subject"*

- **kind**: User

**name**: jane *# "name" is case sensitive*

**apiGroup**: rbac.authorization.k8s.io

**roleRef**:

*# "roleRef" specifies the binding to a Role / ClusterRole*

**kind**: Role *#this must be Role or ClusterRole*

**name**: pod-reader *# this must match the name of the Role or ClusterRole you wish to bind to*

**apiGroup**: rbac.authorization.k8s.io

Cluster Role Binding: ClusterRoleBinding 在集群范围执行授权

一个 RoleBinding 可以引用某 ClusterRole 并将该 ClusterRole 绑定到 RoleBinding 所在的名字空间。 如果你希望将某 ClusterRole 绑定到集群中所有命名空间，你要使用 ClusterRoleBinding

RoleBinding对象也可以引用一个ClusterRole对象用于在RoleBinding所在的命名空间内授予用户对所引用的ClusterRole中 定义的命名空间资源的访问权限。

这一点允许管理员在整个集群范围内首先定义一组通用的角色，然后再在不同的命名空间中复用这些角色。

例如，尽管下面示例中的RoleBinding引用的是一个ClusterRole对象，但是用户”dave”（即角色绑定主体）还是只能读取”development” 命名空间中的secret（即RoleBinding所在的命名空间）。

参考：https://zzq23.blog.csdn.net/article/details/109680359

**apiVersion**: rbac.authorization.k8s.io/v1

*# 此集群角色绑定允许 “manager” 组中的任何人访问任何名字空间中的 secrets*

**kind**: ClusterRoleBinding

**metadata**:

**name**: read-secrets-global

**subjects**:

- **kind**: Group/ServiceAccount

**name**: manager *# 'name' 是不区分大小写的*

**apiGroup**: rbac.authorization.k8s.io/””

**roleRef**:

**kind**: ClusterRole

**name**: secret-reader

**apiGroup**: rbac.authorization.k8s.io

\*创建了绑定之后，你不能再修改绑定对象所引用的 Role 或 ClusterRole。 试图改变绑定对象的 roleRef 将导致合法性检查错误。 如果你想要改变现有绑定对象中 roleRef 字段的内容，必须删除重新创建绑定对象。

将service account 与 pod 绑定

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: my-pod

namespace: xxxx

**spec**:

**serviceAccountName**: build-robot

Normal User:

Create private key:

openssl genrsa -out john.key 2048

openssl req -new -key john.key -out john.csr

Create CertificateSigningRequest:

apiVersion: certificates.k8s.io/v1

kind: CertificateSigningRequest

metadata:

name: john

spec:

groups:

- system:authenticated

request: 

signerName: kubernetes.io/kube-apiserver-client

usages:

- client auth

request = cat john.csr | base64 | tr -d "\n"

Approve certificate signing request

kubectl get csr

kubectl certificate approve john

### Get the certificate

echo “<request>” | base64 --decode > john.crt

Create Role and RoleBinding

Role:

kubectl create role developer --verb=create --verb=get --verb=list --verb=update --verb=delete --resource=pods

RoleBinding:

kubectl create rolebinding developer-binding-john --role=developer --user=john

Add to kubeconfig

kubectl config set-credentials john --client-key=/home/vagrant/work/john.key --client-certificate=/home/vagrant/work/john.crt --embed-certs=true

kubectl config set-context john --cluster=kubernetes --user=john

For context, change the context to john

kubectl config use-context john

Approval or rejection

Approve:

kubectl certificate approve <certificate-signing-request-name>

Reject:

kubectl certificate deny <certificate-signing-request-name>

Use Kubeadm to install a basic cluster

## Letting iptables see bridged traffic:

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

br\_netfilter

EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

sudo sysctl --system

Shutdown swap:

sudo swapoff -a

vi /etc/fstab (sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab)

Shutdown SELINUX

sudo setenforce 0

vim /etc/selinux/config

SELINUX=disabled

Shutdown Firefall

sudo systemctl stop firewalld

sudo systemctl disable firewalld

sudo iptables -F && sudo iptables -X && sudo iptables -F -t nat && sudo iptables -X -t nat

sudo iptables -P FORWARD ACCEPT

Install Docker

<https://docs.docker.com/engine/install/#server>

## Installing kubeadm, kubelet and kubectl

sudo apt-get update

sudo apt-get install -y apt-transport-https ca-certificates curl

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

查看kubelet版本：

kubelet --version

Initializing

kubeadm init <args> --apiserver-advertise-address=<ip-address>

加入node的指令24小时候会过期，可以使用：

kubeadm token create --print-join-command 重新生成相关指令或参考

https://www.cnblogs.com/hongdada/p/9854696.html

CNI Plugin (Weave)

kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=**$(**kubectl version | base64 | tr -d '\n'**)**"

Manage a highly-available Kubernetes cluster

Etcd Quorum（必须有多少etcd node alive） = N/2 + 1 (N = 节点数)

Provision underlying infrastructure to deploy a Kubernetes cluster

Perform a version upgrade on a Kubernetes cluster using Kubeadm

版本控制

kube-apiserver = x

controller-manager & kube-scheduler = x-1

kubelet & kube-proxy = x-2

kubectl = x+1 <==> x-1

Determine which version to upgrade to

apt update

apt-cache madison kubeadm

1.Upgrading control plane nodes

apt-mark unhold kubeadm && \

apt-get update && apt-get install -y kubeadm=1.19.x-00 && \

apt-mark hold kubeadm

-

# since apt-get version 1.1 you can also use the following method

apt-get update && \

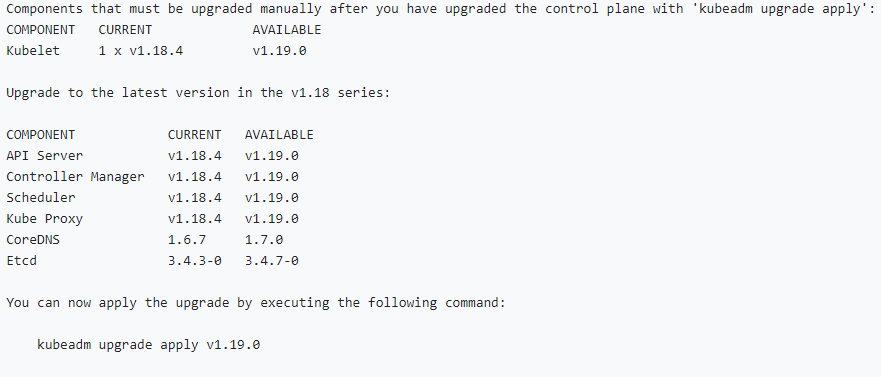
apt-get install -y --allow-change-held-packages kubeadm=1.19.x-00

Verify Version

kubeadm version

Check Upgrade Plan

sudo kubeadm upgrade plan



Upgrade:

kubeadm upgrade apply v1.19.x

For Other control plan:

kubeadm upgrade node

Drain the control plan node:

kubectl drain <cp-node-name> --ignore-daemonsets

Upgrade kubelet & kubectl

apt-mark unhold kubelet kubectl && \

apt-get update && apt-get install -y kubelet=1.19.x-00 kubectl=1.19.x-00 && \

apt-mark hold kubelet kubectl

-

# since apt-get version 1.1 you can also use the following method

apt-get update && \

apt-get install -y --allow-change-held-packages kubelet=1.19.x-00 kubectl=1.19.x-00

sudo systemctl daemon-reload

sudo systemctl restart kubelet

Uncordon control plan node

kubectl uncordon <cp-node-name>

2.Upgrade Worker Nodes

Upgrade kubeadm

apt-mark unhold kubeadm && \

apt-get update && apt-get install -y kubeadm=1.19.x-00 && \

apt-mark hold kubeadm

-

# since apt-get version 1.1 you can also use the following method

apt-get update && \

apt-get install -y --allow-change-held-packages kubeadm=1.19.x-00

Upgrade kubelet configuration

sudo kubeadm upgrade node

Drain the node

kubectl drain <node-to-drain> --ignore-daemonsets

Upgrade Kubelet & kubectl

# replace x in 1.18.x-00 with the latest patch version

apt-mark unhold kubelet kubectl && \

apt-get update && apt-get install -y kubelet=1.18.x-00 kubectl=1.18.x-00 && \

apt-mark hold kubelet kubectl

-

# since apt-get version 1.1 you can also use the following method

apt-get update && \

apt-get install -y --allow-change-held-packages kubelet=1.18.x-00 kubectl=1.18.x-00

Restart the kubelet

sudo systemctl daemon-reload

sudo systemctl restart kubelet

Uncordon the node

kubectl uncordon <node-to-drain>

Implement etcd backup and restore

搜索关键词etcd snapshot

\*Etcd port 2379 is the port of ETCD to which all control plane components connect to. 2380 is only for etcd peer-to-peer connectivity

Backup:

ETCDCTL\_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt \

--cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key \

snapshot save /tmp/snapshot-pre-boot.db

Restore:

ETCDCTL\_API=3 etcdctl --endpoints=https://[127.0.0.1]:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt \

--name=master \

--cert=/etc/kubernetes/pki/etcd/server.crt --key=/etc/kubernetes/pki/etcd/server.key \

--data-dir /var/lib/etcd-from-backup(备份恢复后需要更改路径) \

--initial-cluster=master=https://127.0.0.1:2380 \

--initial-cluster-token=etcd-cluster-1 \

--initial-advertise-peer-urls=https://127.0.0.1:2380 \

snapshot restore /tmp/snapshot-pre-boot.db

systemctl daemon-reload

service etcd restart

Services & Networking20%

Understand host networking configuration on the cluster nodes

Understand connectivity between Pods

CNI Plugins

Weave:

kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=**$(**kubectl version | base64 | tr -d '\n'**)**"

Flannel:

https://github.com/flannel-io/flannel/blob/master/Documentation/kube-flannel.yml

Understand ClusterIP, NodePort, LoadBalancer service types and endpoints

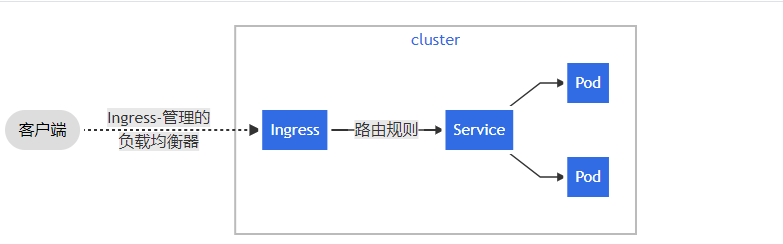
ClusterIP: 通过集群的内部 IP 暴露服务，选择该值时服务只能够在集群内部访问。 这也是默认的 ServiceType

NodePort: 通过每个节点上的 IP 和静态端口（NodePort）暴露服务。 NodePort 服务会路由到自动创建的 ClusterIP 服务。 通过请求 <节点 IP>:<节点端口>，你可以从集群的外部访问一个 NodePort 服务。

LoadBalancer: 使用云提供商的负载均衡器向外部暴露服务。 外部负载均衡器可以将流量路由到自动创建的 NodePort 服务和 ClusterIP 服务上。

Know how to use Ingress controllers and Ingress resources

Ingress 公开了从集群外部到集群内服务的 HTTP 和 HTTPS 路由。 流量路由由 Ingress 资源上定义的规则控制。



Ingress controller install

##### 1.kubectl create ns ingress-space

2.kubectl create configmap nginx-configuration --namespace=ingress-space

3.create role & rolebindings

4.

**apiVersion**: networking.k8s.io/v1

**kind**: Ingress

**metadata**:

**name**: ingress-resource-backend

**namespace**: critical-space

**annotations**:

nginx.ingress.kubernetes.io/rewrite-target:/

**spec**:

**defaultBackend**:

**resource**:

**apiGroup**: k8s.example.com

**kind**: StorageBucket

**name**: static-assets

**rules**:

- host: rewrite.bar.com

http:

paths:

- backend:

serviceName: http-svc

servicePort: 80

path: /something(/|$)(.\*)

**apiVersion**: networking.k8s.io/v1

**kind**: Ingress

**metadata**:

**name**: ingress-wildcard-host

**annotations**:

nginx.ingress.kubernetes.io/rewrite-target:/

**spec**:

**rules**:

- **host**: "foo.bar.com"

**http**:

**paths**:

- **pathType**: Prefix

**path**: "/bar"

**backend**:

**service**:

**name**: service1

**port**:

**number**: 80

Know how to configure and use CoreDNS

<service-name>.<namespace>.<type>.<root>

webb-service.apps.svc.cluster.local

10-244-2-5.apps.pod.cluster.lodal

\*要注意用pod ip 而不是用service ip!

配置文件

CoreDNS configmap

Choose an appropriate container network interface plugin

k8s document search :

# Creating Highly Available clusters with kubeadm

kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=**$(**kubectl version | base64 | tr -d '\n'**)**"

实用命令行

显示labels

kubectl get svc --show-labels

kubectl get pod --show-labels

创建带有label的pod

kubectl run --images=redis --labels=”tier=db,name=app”

查看符合某些label=xxx的pod

kubectl get pod -l tier=db

创建svc, expose pod

kubectl expose pod valid-pod --port=444 --target-port=8443 --name=frontend --type='': Type for this service: ClusterIP, NodePort, LoadBalancer, or ExternalName. Default is 'ClusterIP'.

创建deployments

kubectl create deployment webapp --image=kodekloud/webapp-color --replicas=3

创建Taint

kubectl taint nodes <node-name> key=value:<taint-effect>

taint-effect:

kubectl taint nodes <node-name> key=value:NoSchedule(Nodes can only accept pods that can tolerant the taint)

kubectl taint nodes <node-name> key=value:PreferNoSchedule(System will try to avoid placing a pod on the node but is not guaranteed)

kubectl taint nodes <node-name> key=value:NoExecute(Pod will not be scheduled on the node and existing pods on the node if any will be evicted if they do not tolerate the taint)

查看node taint

kubectl describe node <node-name> | grep -i taint

创建node label

kubectl label nodes <node-name> <label-key>:<label-value>

查看node label

kubectl get nodes <node-name> --show-labels

在无kube-scheduler情况下为pod选择nodes

apiVersion: v1

kind: Pod

metadata:

name: nginx

spec:

nodeName: node01

containers:

- image: nginx

name: nginx

节点亲和性：

Operator:

Exists: check if the label size exists on the nodes and do not need values for that

应用管理

kubectl set image deployment/myapp-deployment <container-name>=nginx:1.20

Applications Rolling Update

kubectl rollout status deployment <deployment-name>

kubectl rollout history deployment <deployment-name>

kubectl rollout undo deployment <deployment-name>

ConfigMap管理

kubectl create configmap <configmap-name> --from-literal=<key1>=<value1> \ --from-literal=<key2>=<value2>

kubectl create configmap <configmap-name> --from-file=<path>

POD 使用ConfigMap:

K8S Document Search : configure a pod use configmap

Secret管理

kubectl create secret generic <secret-name> --from-literal=<key1>=<value1>

kubectl create secret generic <secret-name> --from-file=<file-path>

\*value = echo -n ‘value1’ | base 64

POD设置变量：

kubectl run hazelcast --image=hazelcast --env="DNS\_DOMAIN=cluster" --env="POD\_NAMESPACE=default"

POD设置Command & Args

kubectl run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>

将node unschedule

kubectl cordon <node-name>

将node 中的pod 删除

kubectl drain <node-name>

查看k8s Cluster 版本信息

kubectl version --short

创建kubeconfig文件

*# 设置集群参数*

export KUBE\_APISERVER="https://172.20.0.113:6443"

kubectl config set-cluster kubernetes \

--certificate-authority=/etc/kubernetes/ssl/ca.pem \

--embed-certs=true \

--server=${KUBE\_APISERVER} \

--kubeconfig=devuser.kubeconfig

*# 设置客户端认证参数*

kubectl config set-credentials devuser \

--client-certificate=/etc/kubernetes/ssl/devuser.pem \

--client-key=/etc/kubernetes/ssl/devuser-key.pem \

--embed-certs=true \

--kubeconfig=devuser.kubeconfig

*# 设置上下文参数*

kubectl config set-context kubernetes \

--cluster=kubernetes \

--user=devuser \

--namespace=dev \

--kubeconfig=devuser.kubeconfig

*# 设置默认上下文*

kubectl config use-context kubernetes --kubeconfig=devuser.kubeconfig

指定config file

kubectl config --kubeconfig=my-kube-config use-context research

权限检查：

kubectl auth can-i create deployments --as devuser

查看证书内容：

openssl x509 -in <cert-path> -text -noout

获取权限所属资源(api groups)：

kubectl api-resources

指定私有仓库拉取镜像：(k8s文档中所搜关键字：private registry)

1.<仓库地址>/<镜像名称>

\*kubectl set image deploy web nginx=myprivateregistry.com:5000/nginx:alpine

1.创建secret:

kubectl create secret docker-registry regcred **\**

--docker-server=<你的镜像仓库服务器> **\**

--docker-username=<你的用户名> **\**

--docker-password=<你的密码> **\**

--docker-email=<你的邮箱地址>

2.pod使用

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: private-reg

**spec**:

**containers**:

- **name**: private-reg-container

**image**: <your-private-image>

**imagePullSecrets**:

- **name**: regcred

与pod进行交互

1.不进入pod

kubectl exec <pod-name> -- <command>

kubectl exec ubuntu-sleeper -- whoami

2.进入pod

kubectl exec -it <pod-name> -- <command>

Security context:

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: security-context-demo

**spec**:

**securityContext**:

**runAsUser**: 1000

**runAsGroup**: 3000

**fsGroup**: 2000

**containers**:

- **name**: sec-ctx-demo

**image**: busybox

**command**: [ "sh", "-c", "sleep 1h" ]

**securityContext**:

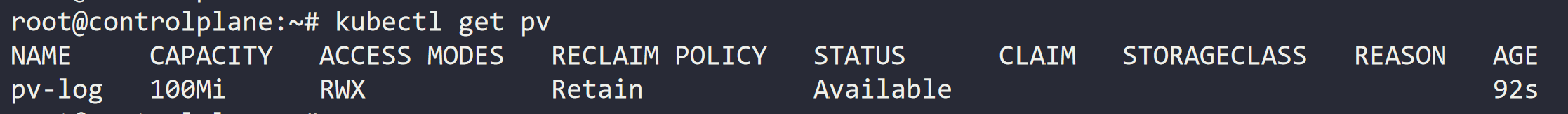
**capabilities**:

**add**: ["NET\_ADMIN", "SYS\_TIME"]

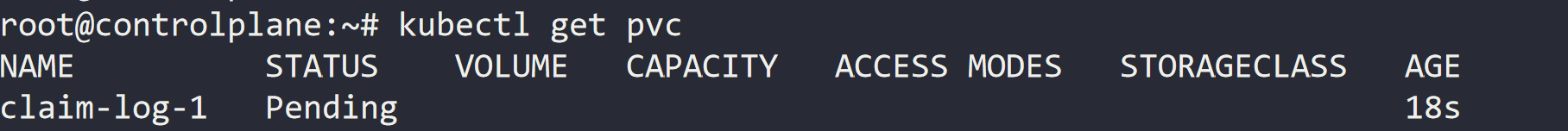
\***capabilities 只可以存在于container level而不能存在于spec level**

创建PV：关键字 Pod PV

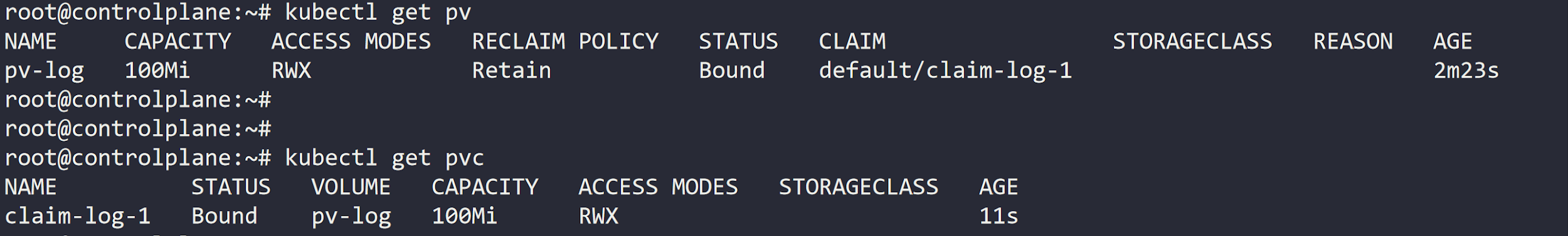
刚创建后PV状态：



刚创建PVC后状态：

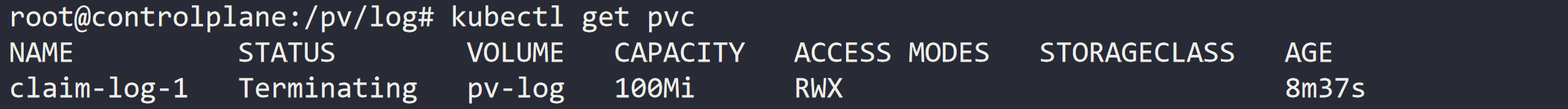


PVC申领PV后的状态：

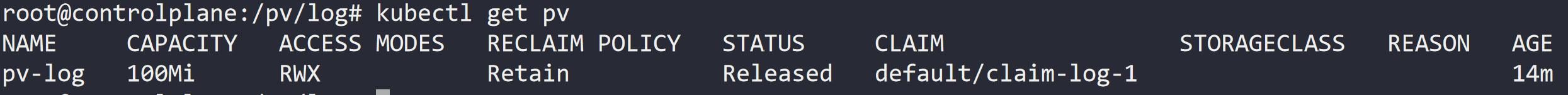


\*pvc 申领50Mi 但是给了100Mi

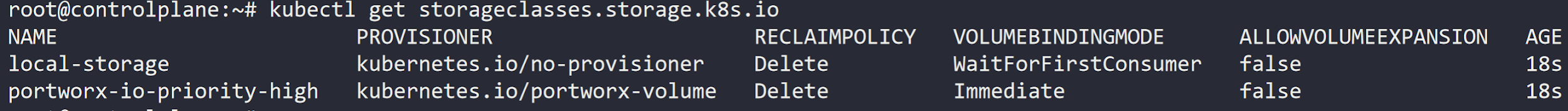
\*若pvc绑定了pod，此时删除pvc，状态为terminating，删除所绑定的pod后，pvc删除



删除pod后pv状态



Storage Class:

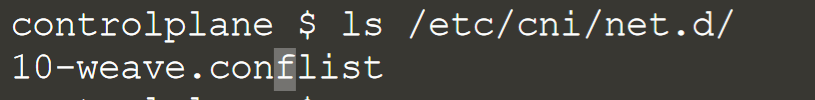


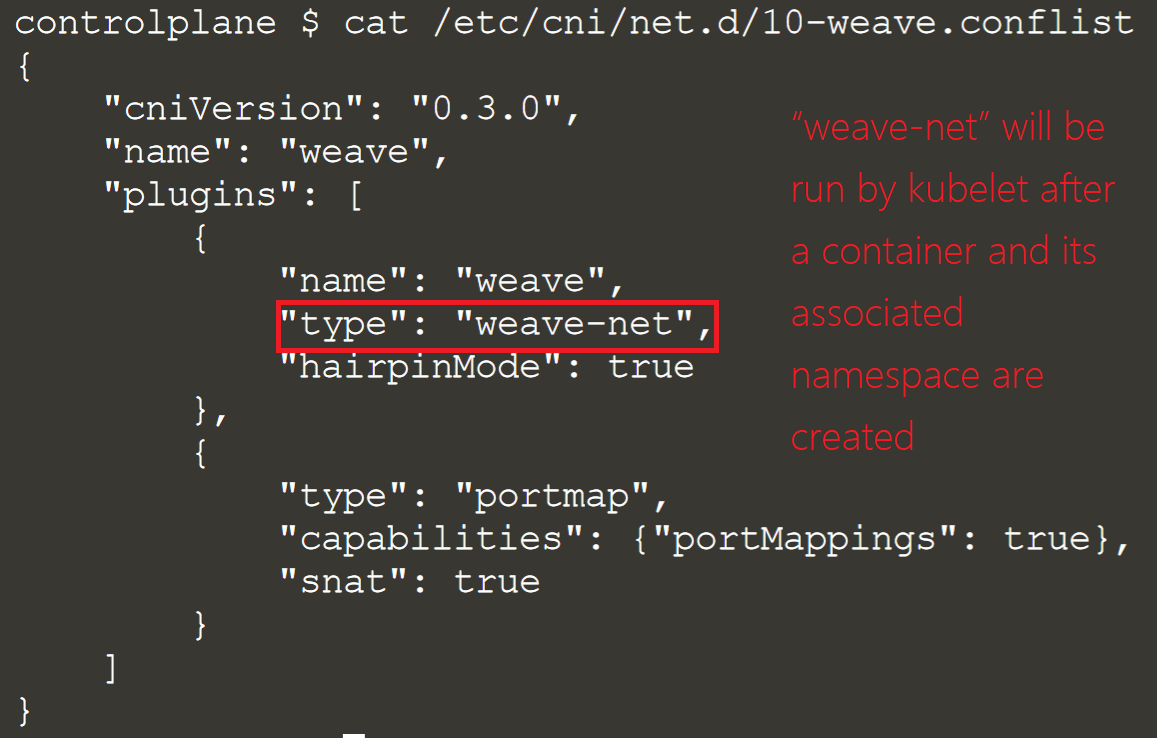
\*no-provisioner 不支持动态申领pv

\*VOLUMEBINDINGMODE = WaitForFirstConsumer

This will delay the binding and provisioning of a PersistentVolume until a Pod using the PersistentVolumeClaim is created.

查看k8s cluster采用的cni：ls /etc/cni/net.d/

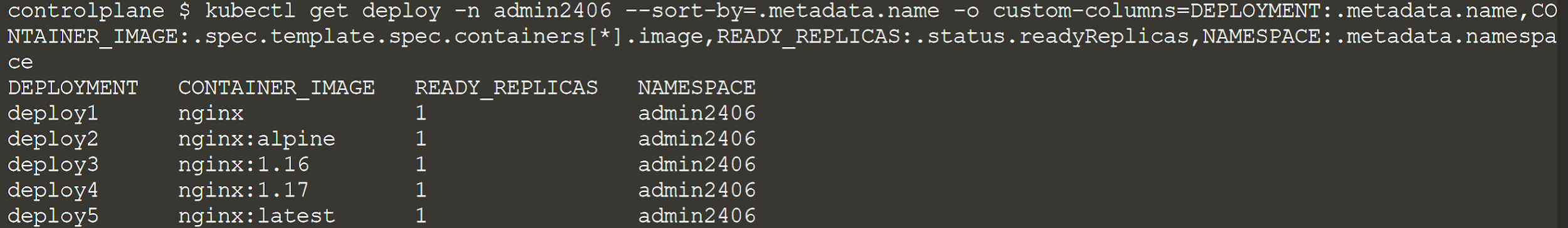




JSON解析

column & sort-by

kubectl get deploy -n admin2406 --sort-by=.metadata.name -o custom-columns=DEPLOYMENT:.metadata.name,CONTAINER\_IMAGE:.spec.template.spec.containers[\*].image,READY\_REPLICAS:.status.readyReplicas,NAMESPACE:.metadata.namespace



循环

‘{range .items[\*]}{.metadata.name}{“\t”}{.status.capacity.cpu}{“\n”}{end}’

过滤

$[?(@.xxx == 1 )] 支持等于（==）不等于（！=）包含（in） 不包含（nin）

K8S 网络ip range

Service Network:

cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep cluster-ip-range

Node Network:

ip addr

Pod Network:

ip addr

查看kube-proxy 模式

kubectl -n kube-system get configmaps kube-proxy -o yaml | grep mode

\*若为””，则默认为iptables

或用kubectl logs kube-proxy-ft6n7 -n kube-system 进行查看

查看资源使用率：

kubectl top node/pod

shell script 相关；

command: [“/bin/sh”,”-c”,”if[ -f /workdir/calm.txt ];then sleep 100000; else exit 1; fi”]

command:[“/bin/sh”,”-c”,”touch /workdir/calm”]

查看label 为name=name-cpu-loader的资源

kubectl top pods -l name=name-cpu-loader --sort-by=cpu

实用：

kubectl 备忘单

<https://kubernetes.io/zh/docs/reference/kubectl/cheatsheet/>

<https://blog.csdn.net/shenhonglei1234/category_10513313.html>