# 概念与原理

## 基本概念

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

## K8s架构

### k8s架构



### Master架构



### Node架构



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

# 资源管理对象

Objects

描述 Kubernetes 对象

创建Kubernetes对象对应的.yaml 文件中，需要的字段：

* 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

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

* 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。

kubectl create -f docs/user-guide/nginx-deployment.yaml --record

# Pod状态与生命周期管理

## 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。

# 集群资源管理

## 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



# 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

# 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

# 存储

# todo

在二类区的服务器上

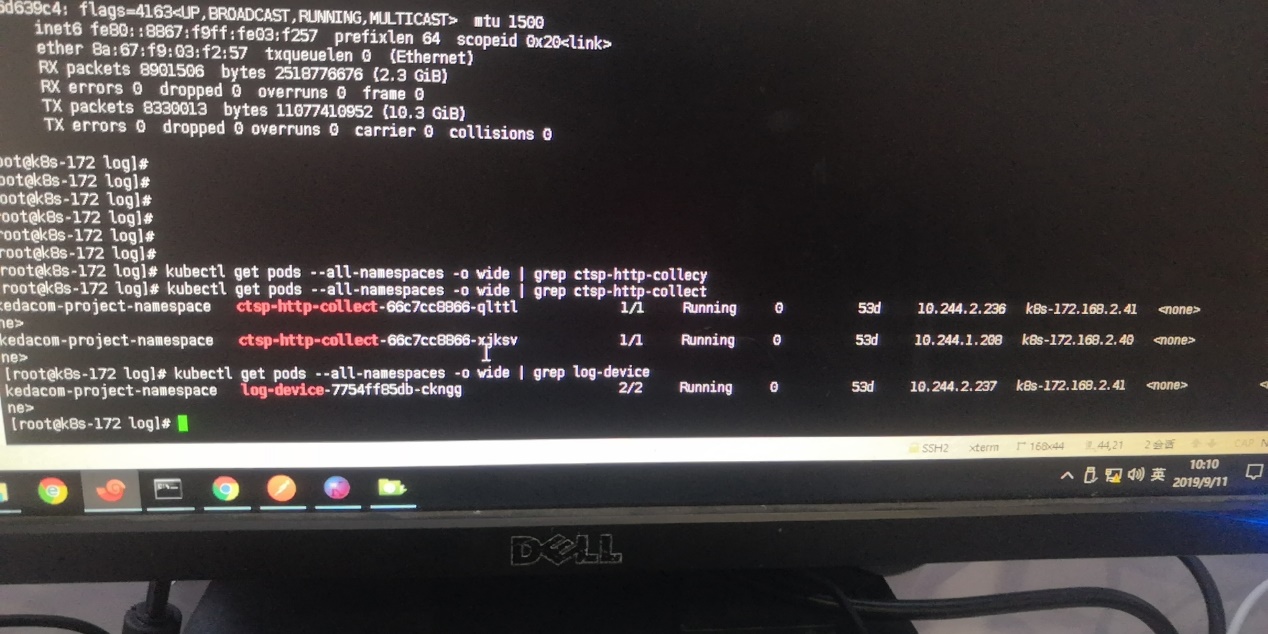
查设备日志的服务

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



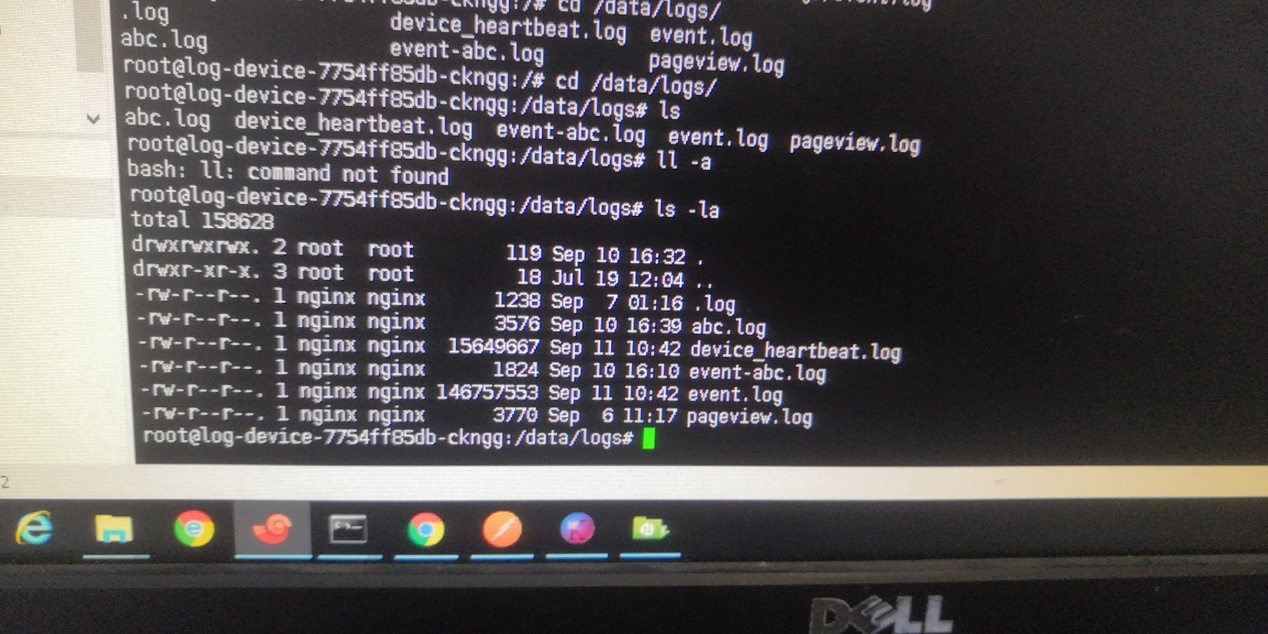
kubectl -n dolphin exec -it log-device-7885d65c9b-9l2w8（此处为下图的名字） bash

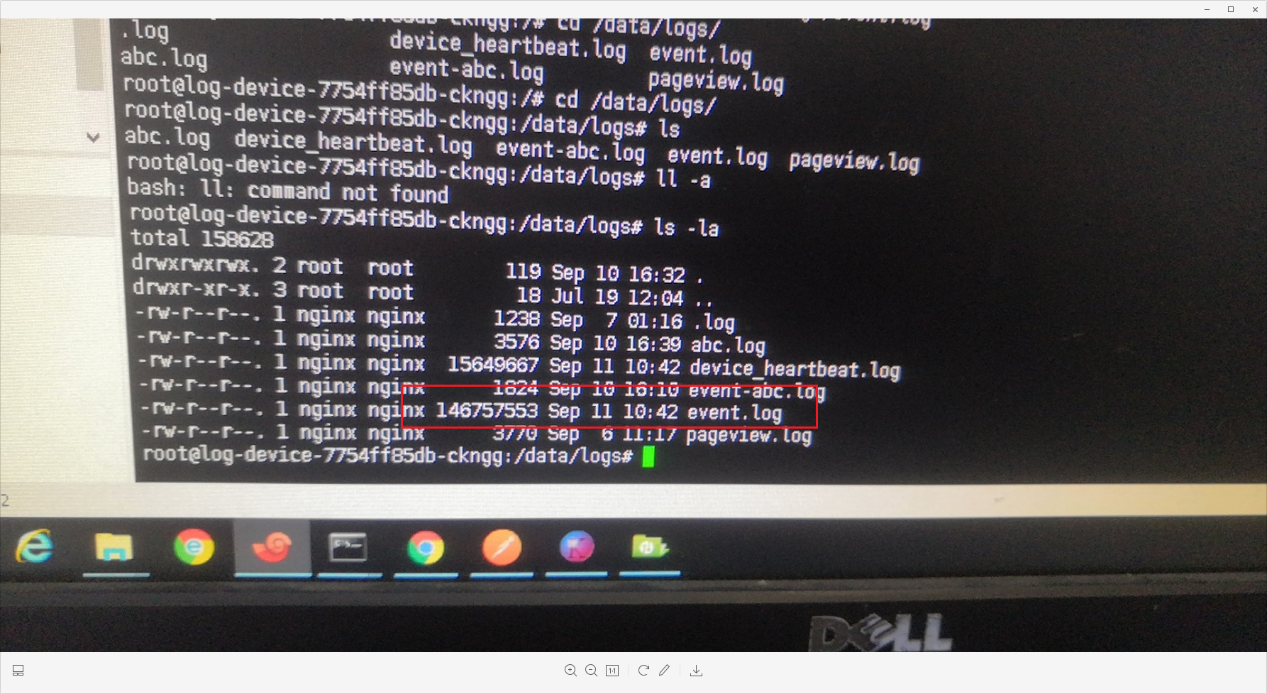
C:\Users\keda\AppData\Local\Temp\WeChat Files\10ce15cb149112aadf1e09a6d0c2bdb.png

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