Certified Kubernetes Administrator

Notes: Concepts & Lab notes

**Course Plan 4**

**Core Concepts 4**

Kubernetes Architecture 8

**Cluster Setup 10**

Master Node initialisation 10

Worker Nodes initialization - Worker1 & 2 10

Verify Cluster setup 11

Deep-dive into Master setup 11

ETCD - Backup 11

ETCD - Restore 12

**Workloads 12**

Pod 12

One-container-per-Pod 12

Multi-Container-Pod 12

Init-Container 13

Resource Limits 14

Deployment 15

Update Deployment 16

Rolling Back to a Previous Revision 16

Scaling deployment 16

Daemon Set 17

Jobs 18

CronJobs 19

Configuration basics 19

Env 20

ConfigMaps 21

Secrets 22

Labels & Selectors 23

Set-based selectors 23

Annotations 24

Namespaces 24

**Helm 24**

Creating application deployment using Helm 25

**Scheduling 25**

Static-Pod

NodeName 25

NodeSelector 26

Node Affinity 26

Pod Affinity 28

Taints and Tolerations 30

**Networking 32**

Service 33

Service Types 36

Ingress 37

Load Balancer 38

**Storage 38**

Non-Persistent/Ephemeral Volume 38

emptyDir 38

Host Directory 39

Volume Config 39

ConfigMap 39

Secret 40

Persistent Volume 40

Persistent Volume Claim 40

Access Modes 41

ReadWriteOnce 41

ReadWriteMany 41

**Troubleshooting 43**

Kubernetes cluster 43

Troubleshooting the control plane 43

Listing the nodes in a cluster 43

List the control plane pods 43

Worker Nodes 44

Application Troubleshooting 44

Basics 44

2-Tier App 48

**Azure Kubernetes Service 49**

Create AKS 49

Workloads 49

Rolling update and rollback 49

Service 49

Creating Load Balancer - Service 49

Storage - PV and PVC 49

Creating PVC with dynamic PV

# Course Plan

Day 1 - Containers, Core Concepts, & Cluster setup

Day 2 - Workloads

Day 3 - Helm & Scheduling

Day 4 - Services & Networking

Day 5 - Storage

Day 6 - AKS & Load Balancer

Day 7 - Troubleshooting

Day 8 - CKA Q&A, RBAC, Resource Quota, CI/CD

# 

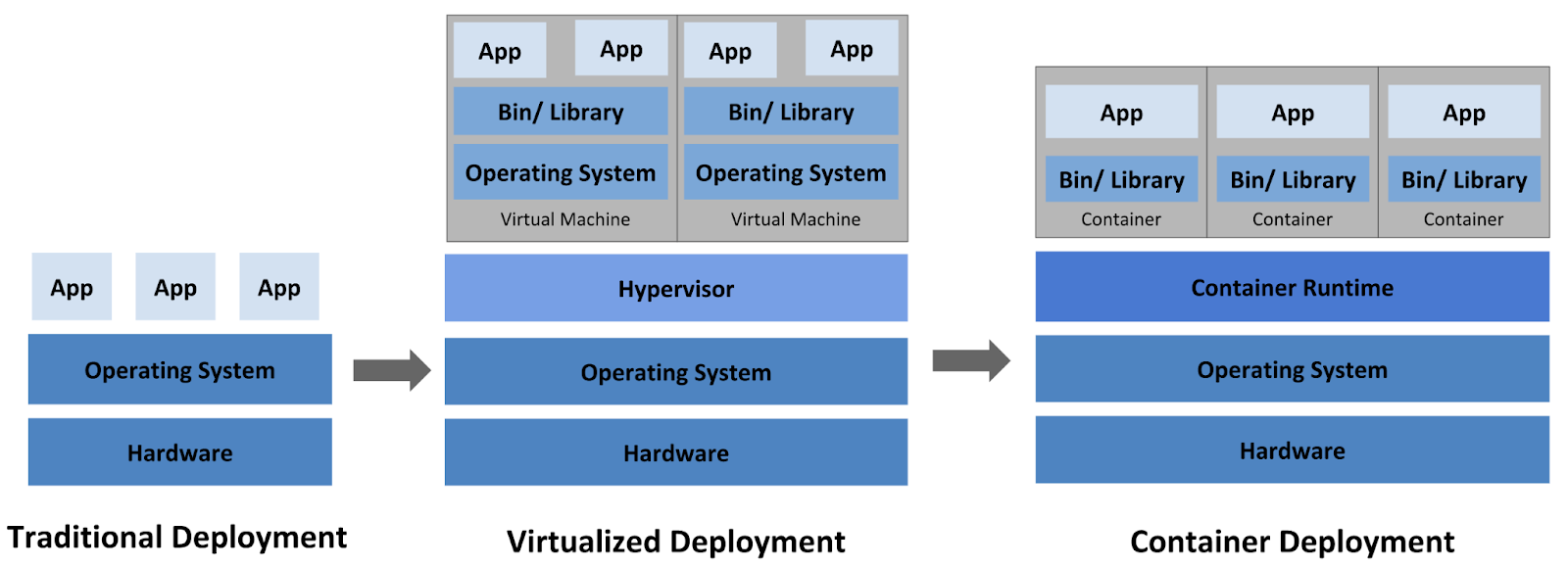
# 

# 

# 

# Core Concepts

Container Evolution



Containerized App -

What’s inside? - App + Library + Bin

Image - App bundled with Dependencies

Container - Image deployed as Container

Deployment differs based on env - Dev, Staging, Prod

Envs, Configurations(Port, DB), Secrets, Resource limits, etc.

**Docker Lab:**

#Images and containers

docker pull nginx

=> Pulls latest version, see all versions -<https://hub.docker.com/_/nginx?tab=tags>

docker run --name nginx -dt -p 80:80 nginx

docker ps

docker stop nginx

docker ps

docker ps -a

docker start nginx

docker ps

docker exec -it nginx /bin/sh

docker rm nginx

docker ps -a

docker images

docker rmi nginx

docker images

# Cleanup

## Containers

docker rm $(docker ps -qa)

## Images

docker image prune -a

app.js

const http = require('http');

const hostname = '0.0.0.0';

const port = 80;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello Docker Chief\n');

});

server.listen(port, hostname, () => {

console.log('Server running at http://%s:%s/', hostname, port);

});

process.on('SIGINT', function() {

console.log('Caught interrupt signal and will exit');

process.exit();

});

**Dockerfile**

# Use an official Node runtime as the parent image

FROM node:6

# Set the working directory in the container to /app

WORKDIR /app

# Copy the current directory contents into the container at /app

ADD . /app

# Make the container's port 80 available to the outside world

EXPOSE 80

# Run app.js using node when the container launches

CMD ["node", "app.js"]

**Docker Commands**

docker build -t node-app:0.1 .

docker images

docker run -p 4000:80 --name my-app -dt node-app:0.1

curl [http://localhost:4000](http://localhost:4000/)

docker logs my-app

docker stop my-app && docker rm my-app

docker run -p 4000:80 --name my-app02 -d node-app:0.1

docker ps

docker logs [container\_id]

follow the log's output as the container is running, use the -f option,

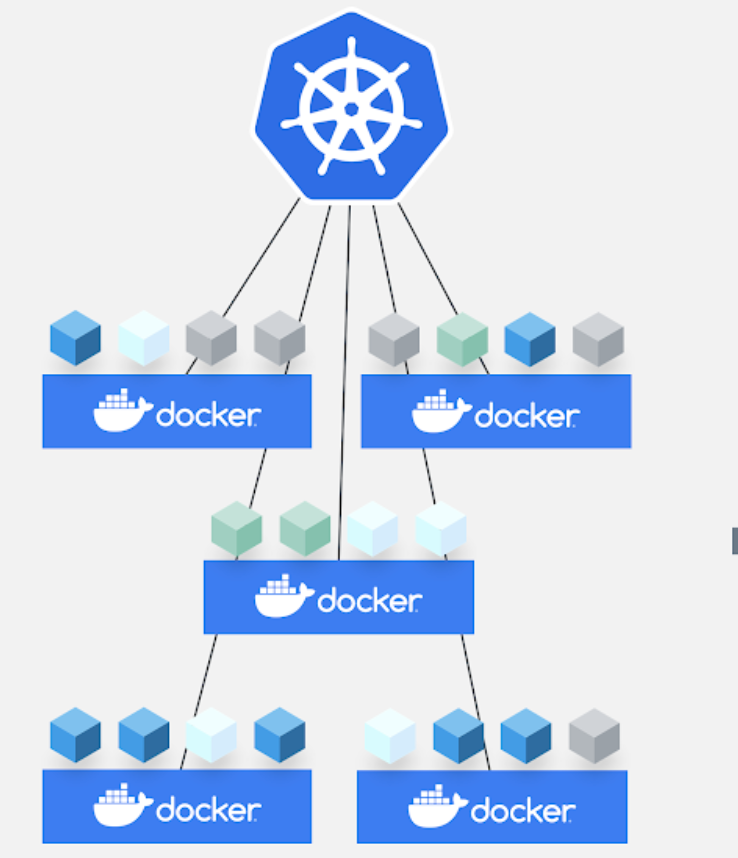
docker logs -f [container\_id]

examine a container's metadata in Docker by using Docker inspect:

docker inspect [container\_id]

docker exec -it my-app02 '/bin/bash'

How to manage containers in different nodes?



## Kubernetes Architecture

Master

API Server

Scheduler

Controller Manager

Cloud Controller Manager(CCM)

Etcd

Worker

Kubelet <-> Dockerd

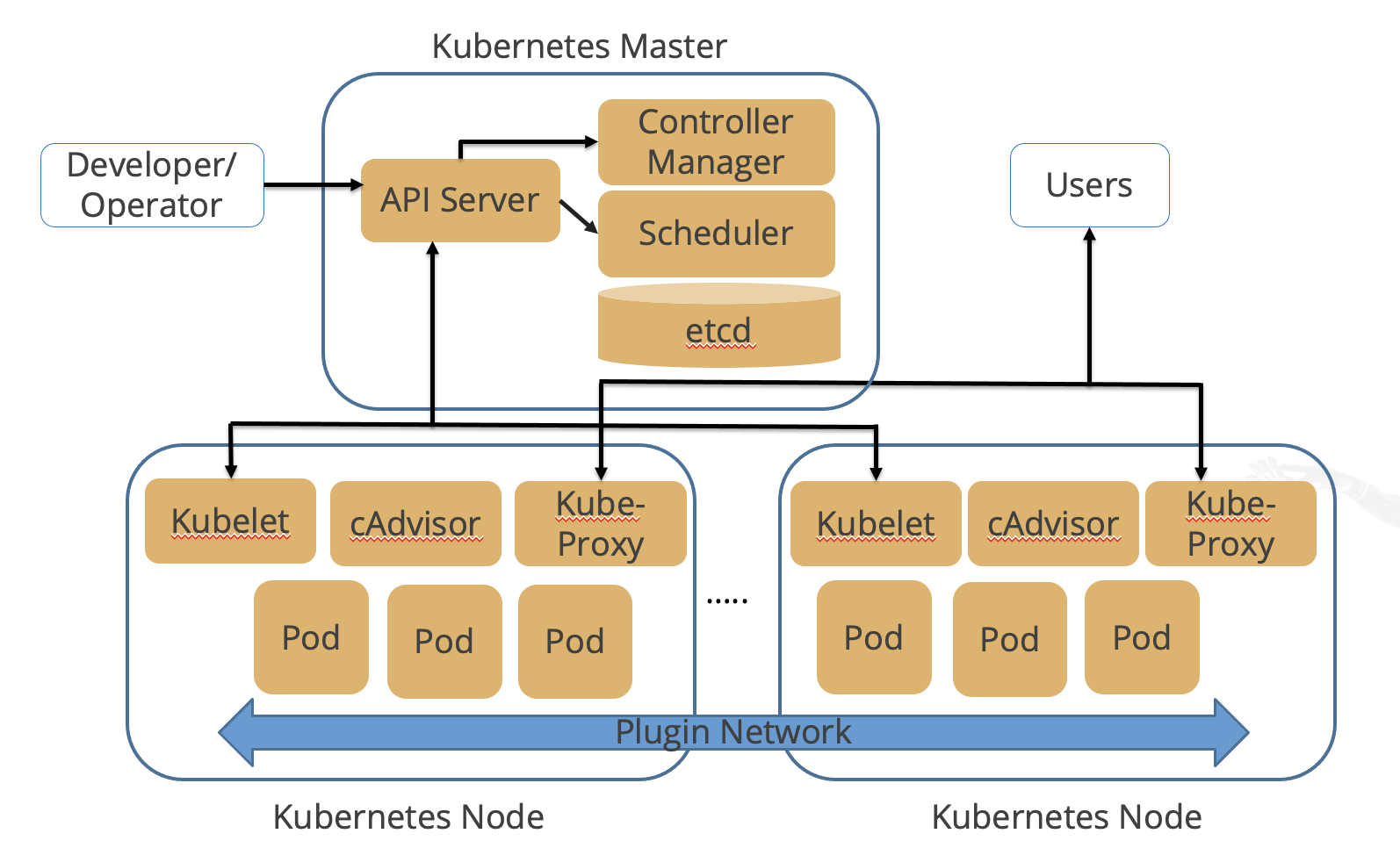
Kube-proxy

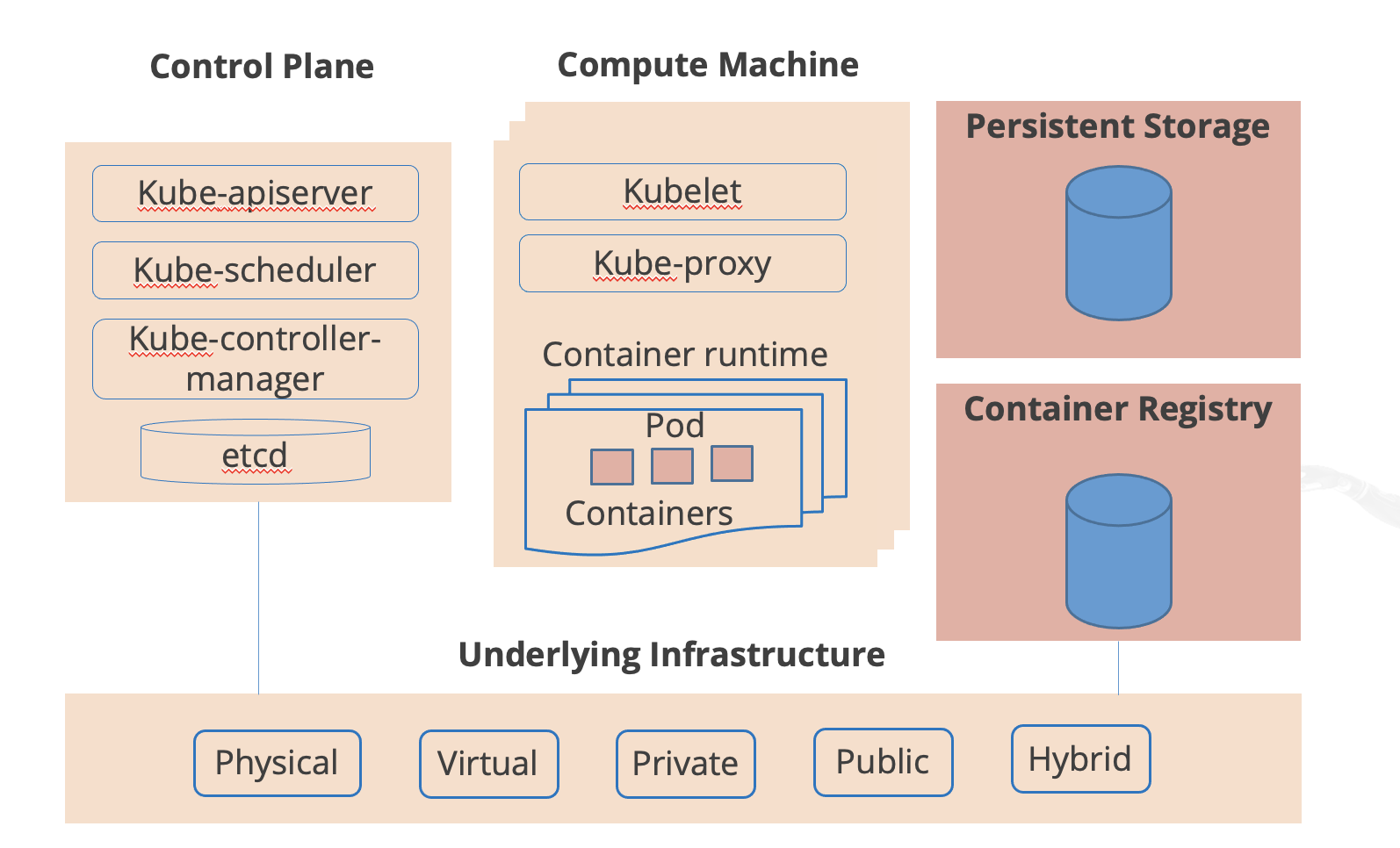
Pods - Containers - App running

Operator - Access API-Server for k8s calls - Create, Update, Read, Delete resources

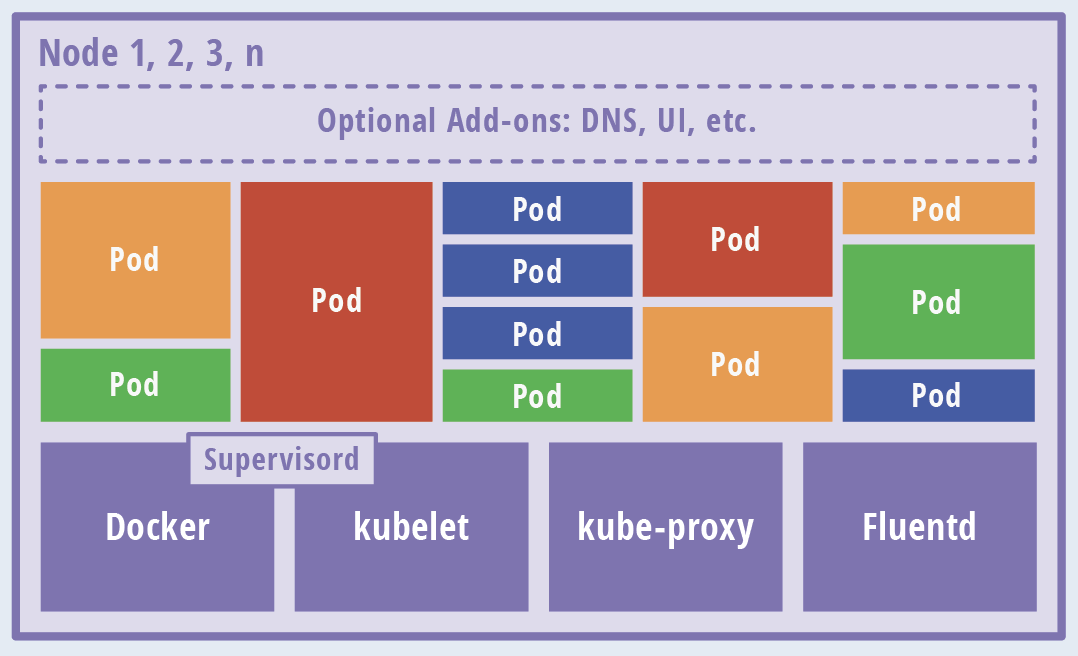
K8s client - UI, API, CLI - kubectl

User - App users - Access App deployed in k8s

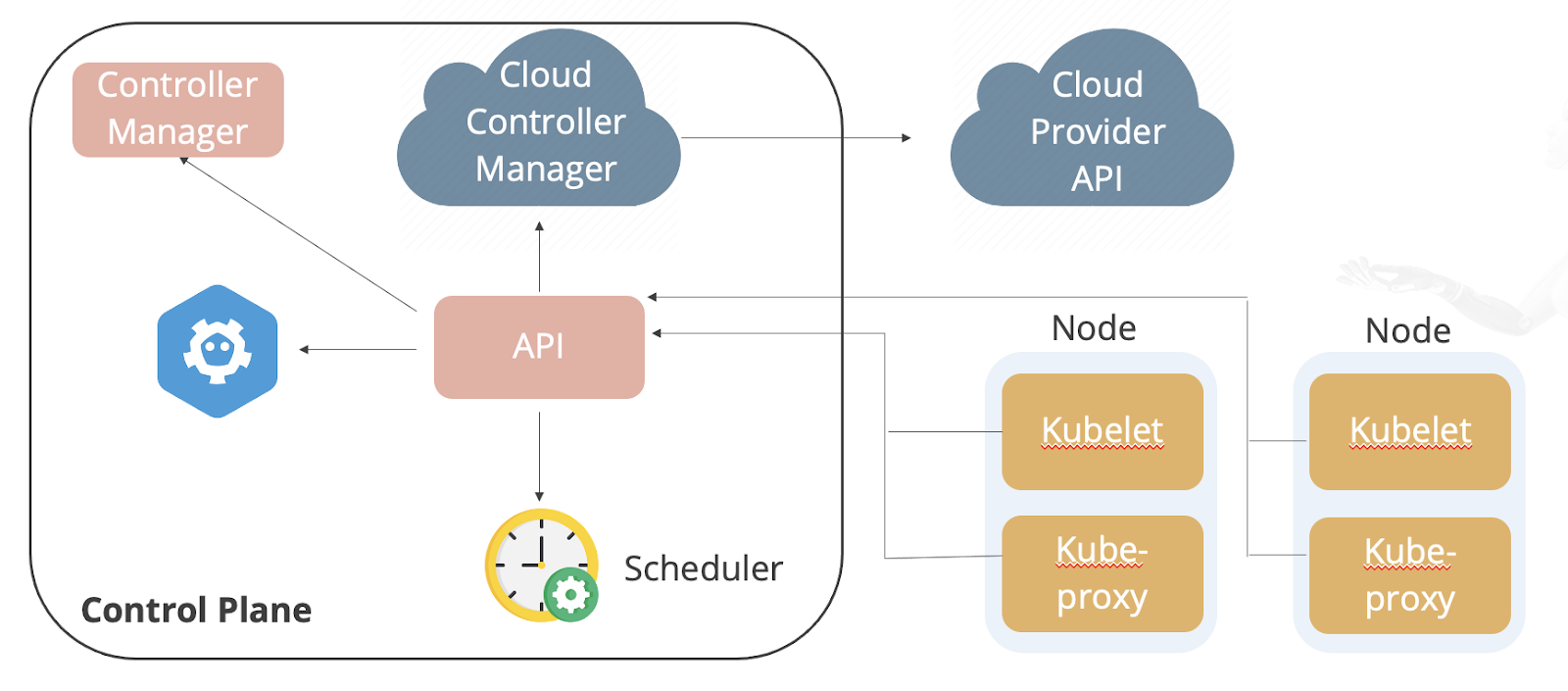




Node View - Many sizes of pods deployed



**Cloud Controller Manager (CCM)**

****

# Cluster Setup

## Master Node initialisation

sudo kubeadm init --node-name master

* Copy kubeadm join command

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

cat ~/.kube/config

Install Container Network Interface (**CNI**)

kubectl apply -f <https://docs.projectcalico.org/manifests/calico.yaml>

## Worker Nodes initialization - Worker1 & 2

**DONT COPY AND PASTE:**

sudo kubeadm join <Master IP Address>:6443 --token <token> \

--discovery-token-ca-cert-hash <some hash> **--node-name worker01**

sudo kubeadm join <Master IP Address>:6443 --token <token> \

--discovery-token-ca-cert-hash <some hash> **--node-name worker02**

**Note:** In case you need to find your unique token, run the command **sudo kubeadm token create --print-join-command**

## Verify Cluster setup

In the Master node,

kubectl get nodes

kubectl run nginxpod --image=nginx --port 80

kubectl get pods

## Deep-dive into Master setup

kubectl cluster-info

kubectl cluster-info dump > cluster-dump

kubectl get node worker01

kubectl describe node worker01 | less

# Look at Status(should be FALSE), Address, Capacity, and Events

kubectl get namespaces

Kubectl get pods -A

kubectl get pods -n kube-system

# Look into /etc/kubernetes/ - Config, manifests & pki

kubectl get pods -n kube-system -o wide | grep proxy

service kubelet status

## ETCD - Backup

Step 1: Get URLs and keys

kubectl describe pod etcd-master -n kube-system

Get client-URL, cert, key, and trusted-ca location

Step 2: Command

sudo snap install etcd

sudo apt install etcd-client

sudo chmod a+rw -R /etc/kubernetes/pki

sudo ETCDCTL\_API=3 etcdctl snapshot save etcd\_backup.db \

--endpoints https://172.31.20.239:2379 \

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

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

--cacert=/etc/kubernetes/pki/etcd/ca.crt

Step 3: Verify

sudo ETCDCTL\_API=3 etcdctl --write-out=table snapshot status etcd\_backup.db \

--endpoints https://<cluster-ip>:2379 \

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

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

--cacert=/etc/kubernetes/pki/etcd/ca.crt

## ETCD - Restore

Works only on multi-master etcd node, Don’t try in single master setup.

This command is only for Reference.

ETCDCTL\_API=3 etcdctl --write-out=table snapshot restore etcd\_backup.db \

--endpoints https://<cluster-ip>:2379 \

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

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

--cacert=/etc/kubernetes/pki/etcd/ca.crt

# Workloads

Kubernetes Objects - apiVersion, Kind, metadata, and Spec

## Pod

alias k=kubectl

k api-resources

k explain pod.spec.containers

k explain pod.spec.containers –recursive

### One-container-per-Pod

apiVersion: v1

kind: Pod

metadata:

name: nginx

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

kubectl apply -f <https://k8s.io/examples/pods/simple-pod.yaml>

kubectl get pods

kubectl describe pod nginx

# Looks for events and lifecyle

### Multi-Container-Pod

apiVersion: v1

kind: Pod

metadata:

name: multicontainer-pods

labels:

app: httpd

tier: frontend-backend

version: v1

spec:

containers:

#Container 01

- name: web

image: httpd

ports:

- containerPort: 80

#Container 02

- name: redis

image: redis

### Init-Container

apiVersion: v1

kind: Pod

metadata:

name: purple

spec:

containers:

- command:

- sh

- -c

- echo The app is running! && sleep 3600

image: busybox:1.28

name: purple-container

# Adding 2 init containers to execute sleep commands

initContainers:

- command:

- sh

- -c

- sleep 60

image: busybox:1.28

name: warm-up-1

- command: ["sh", "-c", "sleep 120"]

image: busybox:1.28

name: warm-up-2

kubectl get pods -w

# Both the init containers will get executed before the main container is started

# NAME READY STATUS RESTARTS AGE

# purple 0/1 Init:1/2 0 2m41s

# After 3mins(60+120seconds), the output will be

# NAME READY STATUS RESTARTS AGE

# purple 1/1 Running 0 3m7s

### Static-Pod

In worker01, Create a yaml file in

apiVersion: v1

kind: Pod

metadata:

name: static-web

labels:

role: myrole

spec:

containers:

- name: web

image: nginx

ports:

- name: web

containerPort: 80

protocol: TCP

In the master node,

k get pods -A

The pod will appear in default ns

Delete the static pod file in worker01

In the master node,

k get pods -A

The pod will Disappear in default ns

### Resource Limits

apiVersion: v1

kind: Pod

metadata:

name: rl-pod

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

resources:

requests: # Minimum Value

memory: "100Mi"

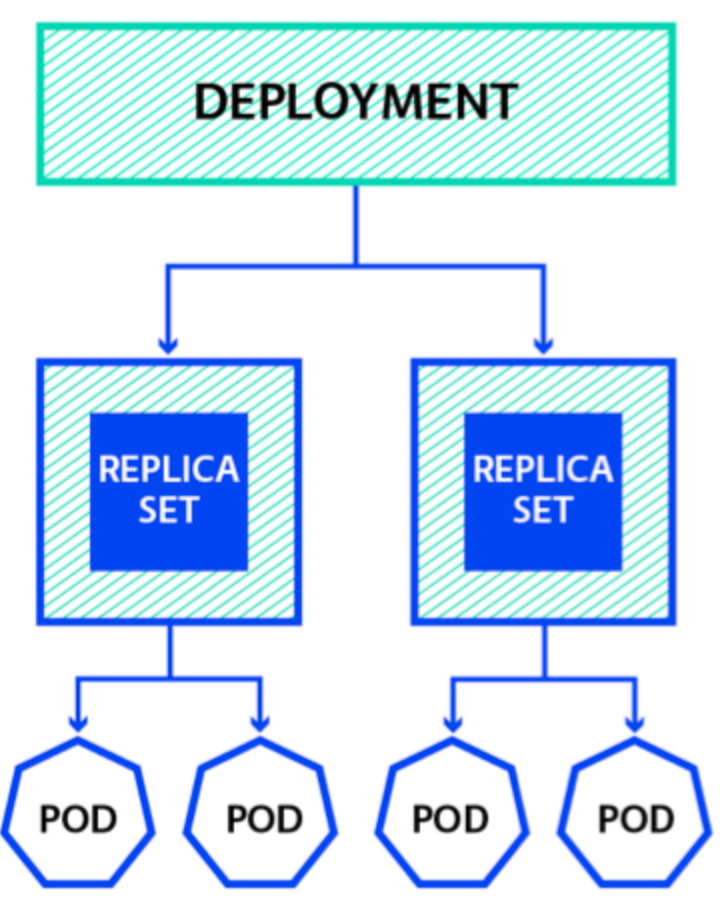
cpu: "250m" # 1 core = 1000m

limits: # Maximum Value

memory: "128Mi"

cpu: "300m"

## Deployment



apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml

kubectl get deployments

kubectl rollout status deployment/nginx-deployment

kubectl get rs

kubectl get pods --show-labels

### Update Deployment

kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1

kubectl rollout status deployment/nginx-deployment

kubectl get rs

kubectl get pods | grep nginx-deployment

kubectl describe deployment nginx-deployment

Setting wrong image

kubectl set image deployment/nginx-deployment nginx=nginx:1.161

kubectl rollout status deployment/nginx-deployment

Waiting for rollout to finish: 1 out of 3 new replicas has been updated...

kubectl get rs

kubectl get pods | grep nginx-deployment

kubectl describe deployment

kubectl rollout history deployment/nginx-deployment

kubectl rollout history deployment/nginx-deployment --revision=2

### Rolling Back to a Previous Revision

kubectl rollout undo deployment/nginx-deployment

kubectl rollout history deployment/nginx-deployment

kubectl rollout history deployment/nginx-deployment --revision=4

kubectl get deployment nginx-deployment

kubectl describe deployment nginx-deployment

# Check container image version/tag

k rollout undo deployment/nginx-deployment --to-revision=1

### Scaling deployment

kubectl scale deployment/nginx-deployment --replicas=5

kubectl get deployment nginx-deployment

kubectl get rs

kubectl get pods | grep nginx-deployment

kubectl describe deployment nginx-deployment

## Daemon Set

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd-elasticsearch

namespace: kube-system

labels:

k8s-app: fluentd-logging

spec:

selector:

matchLabels:

name: fluentd-elasticsearch

template:

metadata:

labels:

name: fluentd-elasticsearch

spec:

tolerations:

# this toleration is to have the daemonset runnable on master nodes

# remove it if your masters can't run pods

- key: node-role.kubernetes.io/master

operator: Exists

effect: NoSchedule

containers:

- name: fluentd-elasticsearch

image: quay.io/fluentd\_elasticsearch/fluentd:v2.5.2

resources:

limits:

memory: 200Mi

requests:

cpu: 100m

memory: 200Mi

volumeMounts:

- name: varlog

mountPath: /var/log

- name: varlibdockercontainers

mountPath: /var/lib/docker/containers

readOnly: true

terminationGracePeriodSeconds: 30

volumes:

- name: varlog

hostPath:

path: /var/log

- name: varlibdockercontainers

hostPath:

path: /var/lib/docker/containers

kubectl apply -f https://k8s.io/examples/controllers/daemonset.yaml

kubectl get ds

kubectl describe ds fluentd-elasticsearch

kubectl get pods -o wide | grep fluentd

## Jobs

apiVersion: batch/v1

kind: Job

metadata:

name: pi

spec:

template:

spec:

containers:

- name: pi

image: perl

command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]

restartPolicy: Never

backoffLimit: 4

kubectl apply -f job.yaml

kubectl describe jobs/pi

pods=**$(**kubectl get pods --selector=job-name=pi --output=jsonpath='{.items[\*].metadata.name}'**)**

echo $pods

kubectl logs $pods

## CronJobs

wget <https://k8s.io/examples/application/job/cronjob.yaml>

Note: Change version in yaml

kubectl create -f cronjob.yaml

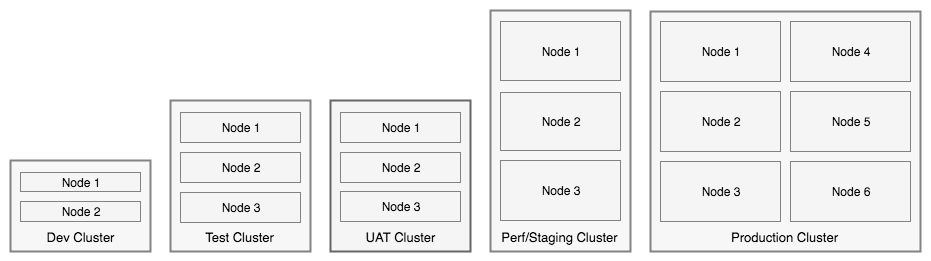
kubectl get cronjob hello

kubectl get jobs -w

kubectl delete -f cronjob.yaml

## Configuration basics

Introduction:



[All JAVA Spring Configuration item](https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#application-properties.data.spring.datasource.username)

[MySQL DB Configuration](https://github.com/spring-guides/gs-accessing-data-mysql/blob/main/complete/src/main/resources/application.properties)

a[pplication.properties](https://github.com/spring-guides/gs-accessing-data-mysql/blob/main/complete/src/main/resources/application.properties)

spring.datasource.url=jdbc:mysql:// \

${MYSQL\_HOST:localhost}:3306/db\_example

spring.datasource.username=springuser

spring.datasource.password=ThePassword

spring.datasource.driver-class-name =com.mysql.jdbc.Driver

For each environment, we define the individual application.properties



In application.properties, provide default profile

spring.profiles.active=dev

While deploying the jar at different environments,

java -jar app.jar -Dspring.profiles.active=prod

## Env

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"

- name: DEMO\_FAREWELL

value: "Such a sweet sorrow"

kubectl apply -f https://k8s.io/examples/pods/inject/envars.yaml

kubectl get pods -l purpose=demonstrate-envars

kubectl exec envar-demo -- printenv

## ConfigMaps

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"

*# file-like keys*

game.properties: |

*enemy.types=aliens,monsters*

*player.maximum-lives=5*

user-interface.properties: |

*color.good=purple*

*color.bad=yellow*

*allow.textmode=true*

K describe cm game-demo

**Using Env**

apiVersion: v1

kind: Pod

metadata:

name: dapi-test-pod01

spec:

containers:

- name: test-container

image: k8s.gcr.io/busybox

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

**env:**

- name: LIVES

valueFrom:

configMapKeyRef:

name: game-demo

key: player\_initial\_lives

- name: FILE\_NAME

valueFrom:

configMapKeyRef:

name: game-demo

key: ui\_properties\_file\_name

restartPolicy: Never

**Using EnvFrom**

apiVersion: v1

kind: Pod

metadata:

name: dapi-test-pod02

spec:

containers:

- name: test-container

image: k8s.gcr.io/busybox

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

**envFrom:**

- configMapRef:

name: game-demo

restartPolicy: Never

## Secrets

k describe secrets dev-db-secret

#Reading Secrets

## Direct assignment, where 'rootroot' is encode and stored in the file

k create secret `generic` app-secret --from-literal password=rootroot --dry-run -o yaml > secret.yml

### Decoding the text in secret.yml

echo "cm9vdHJvb3Q=" | base64 --decode

apiVersion: v1

kind: Secret

metadata:

name: mysecret

type: kubernetes.io/basic-auth

stringData:

username: admin

password: t0p-Secret

---

apiVersion: v1

kind: Pod

metadata:

name: secret-env-pod

spec:

containers:

- name: mycontainer

image: redis

env:

- name: SECRET\_USERNAME

valueFrom:

secretKeyRef:

name: mysecret

key: username

- name: SECRET\_PASSWORD

valueFrom:

secretKeyRef:

name: mysecret

key: password

restartPolicy: Never

kubectl exec -it secret-env-pod /bin/sh

echo $SECRET\_USERNAME

echo $SECRET\_PASSWORD

# Helm

QuickStart: https://helm.sh/docs/intro/quickstart/

sudo snap install helm --classic

helm version

helm repo add bitnami https://charts.bitnami.com/bitnami

helm repo update

helm search repo bitnami

#Can view many big tools are available

Charts:

<https://github.com/bitnami/charts/tree/master/bitnami>

Nginx -<https://github.com/bitnami/charts/tree/master/bitnami/nginx/templates>

helm install test-nginx bitnami/nginx

k get deploy

nginx-1631818923

k get pod

helm uninstall nginx-1631818923

k get deploy

## Creating application deployment using Helm

Traditional Method: without using Helm

<https://github.com/IBM/guestbook/tree/master/v2>

<https://github.com/IBM/guestbook/blob/master/v2/guestbook-deployment.yaml>

Helm way:

<https://github.com/IBM/helm101/tree/master/charts/guestbook>

<https://github.com/IBM/helm101/blob/master/charts/guestbook/templates/guestbook-deployment.yaml>

git clone https://github.com/IBM/helm101.git

cd /home/labsuser/troubleshooting/helm101/charts

helm install guestbook-demo ./guestbook/

k get deploy

k get svc

k get pods

# Scheduling

## NodeName

apiVersion: v1

kind: Pod

metadata:

name: node-name-pod

spec:

#Using Node Name

**nodeName: worker02**

containers:

- image: nginx

name: pod

ports:

- containerPort: 80

## NodeSelector

**kubectl label node worker2 environment**=production disktype=ssd

#unset Lables

**kubectl label node worker2 environment-**

kubectl get nodes --show-labels

apiVersion: v1

kind: Pod

metadata:

name: node-selector-pod

spec:

#Using Node labels

nodeSelector:

environment: production

containers:

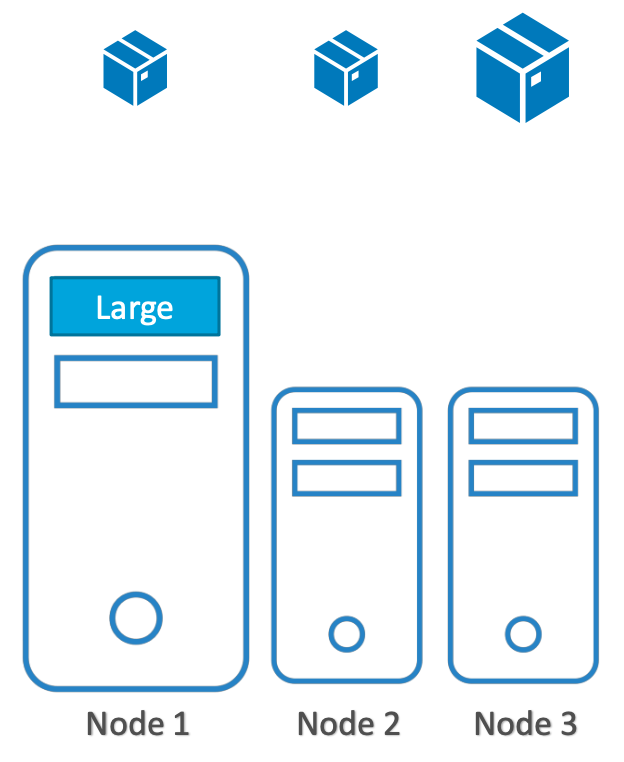
- image: nginx

name: pod

ports:

- containerPort: 80

## Node Affinity



apiVersion: v1

kind: Pod

metadata:

creationTimestamp: null

name: affinity-pod

spec:

containers:

- image: nginx

name: pod

ports:

- containerPort: 80

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: environment

operator: In

values:

- production

- test

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

preference:

matchExpressions:

- key: disktype

operator: In

values:

- ssd

## 

## Pod Affinity

**Goal:**

| **node-1** | **node-2** | **node-3** |
| --- | --- | --- |
| *webserver-1* | *webserver-2* | *webserver-3* |
| *redis-cache-1* | *redis-cache-2* | *redis-cache-3* |

**redis-cache**

apiVersion: apps/v1

kind: Deployment

metadata:

name: redis-cache

spec:

selector:

matchLabels:

app: store

replicas: 3

template:

metadata:

labels:

app: store

spec:

affinity:

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values:

- store

topologyKey: "kubernetes.io/hostname"

containers:

- name: redis-server

image: redis:3.2-alpine

**web-server**

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-server

spec:

selector:

matchLabels:

app: web-store

replicas: 3

template:

metadata:

labels:

app: web-store

spec:

affinity:

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values:

- web-store

topologyKey: "kubernetes.io/hostname"

podAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values:

- store

topologyKey: "kubernetes.io/hostname"

containers:

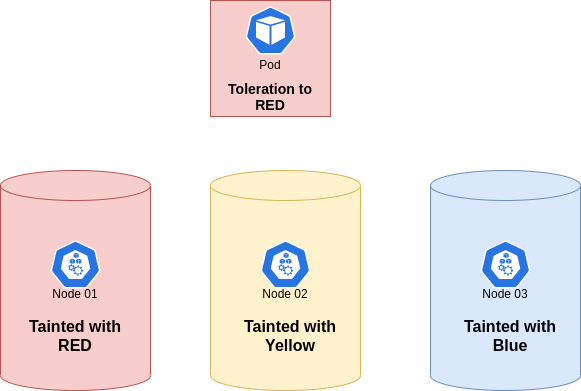
- name: web-app

image: nginx:1.16-alpine

k get pods -l app=store -o wide

k get pods -l app=web-store -o wide

## Taints and Tolerations



Example: Project reserved nodes, Windows containers, Special hardware - GPU

Taint a node:

kubectl taint node worker2 type=gpu:NoSchedule

Tolerate the taint in a Pod

apiVersion: v1

kind: Pod

metadata:

name: test-taint-pod

spec:

containers:

- name: nginxcontainer

image: nginx

tolerations:

- key: type

operator: Equal

value: gpu

effect: NoSchedule

Untaint a node:

kubectl taint node worker2 type=gpu:NoSchedule-

? **Node Affinity vs Pod Affinity**

* Node Affinity ensures that pods are hosted on particular nodes.
* Pod Affinity ensures two pods to be co-located in a single node.

? **Node Affinity vs Taint & Tolerations**

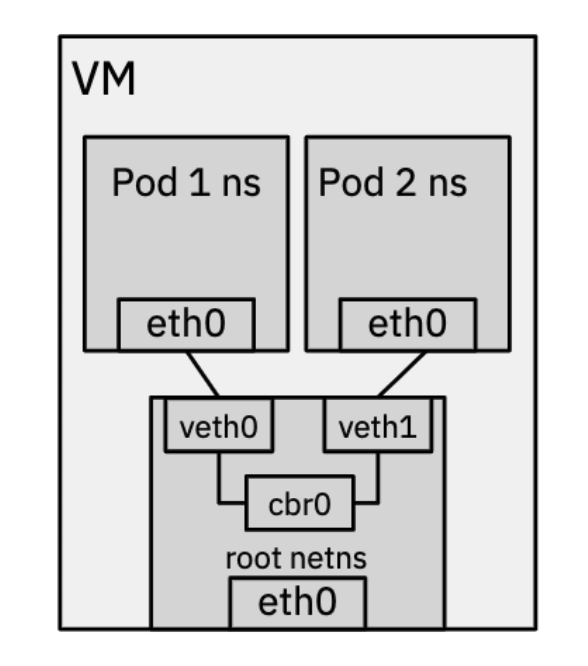
*Node affinity* is a property of [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) that *attracts* them to a set of [nodes](https://kubernetes.io/docs/concepts/architecture/nodes/) (either as a preference or a hard requirement).

*Taints* are the opposite -- they allow a node to repel a set of pods.

*Tolerations* are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints.

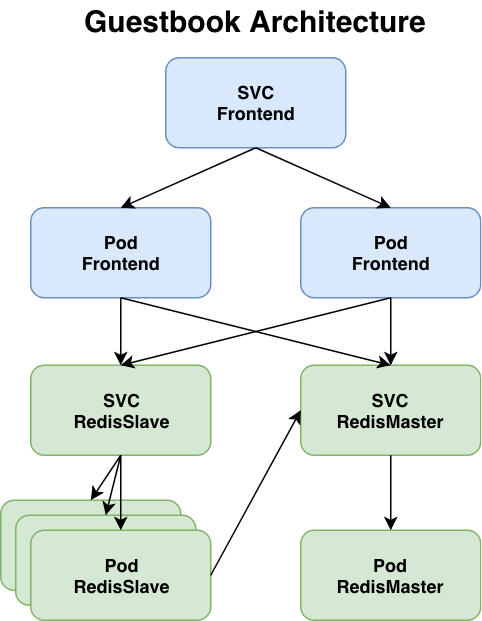
# Networking



Possible Communications

* Containers to Containers in same pod using Loopback
* Pod to Pod
* Service ←→ pods
* Service to Service in same namespace (svc-name.default)
* Service to Service in different namespace (svc-name.test)
* Pod to Internet
* External to Service

## Service



From kubesample.yaml,

apiVersion: v1

kind: Service

metadata:

name: redis

labels:

app: hello

tier: backend

role: master

spec:

ports:

- port: 6379

targetPort: 80

selector:

app: redis

tier: backend

role: master

#Default ns - Pod and Service deployment

k apply -f kubesample.yaml

k get pods; k get svc

#test ns - Pod and Service deployment

k create ns test

vi nginx.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-nginx

spec:

selector:

matchLabels:

run: my-nginx

replicas: 2

template:

metadata:

labels:

run: my-nginx

spec:

containers:

- name: my-nginx

image: nginx

ports:

- containerPort: 80

k apply -f nginx.yaml -n test

k expose deployment/my-nginx -n test

#IP and Networking

k get pod frontend-667b6d7d5d-9c9n6 -o yaml | grep -i podIP

#Check the Pod IP

k get ep frontend

#Check the List of Pod IPs

k get svc | grep frontend

#Check service IP

k exec frontend-667b6d7d5d-6p94c -- printenv | grep SERV

#All Service IP in default ns are listed

#Check DNS resolving in same ns

k exec -it frontend-667b6d7d5d-6p94c -- ping redis

#PING redis.default.svc.cluster.local (10.99.107.243): 56 data bytes

#Check DNS resolving in `test` ns

k exec -it frontend-667b6d7d5d-6p94c -- ping my-nginx.test.svc.cluster.local

#PING web-service.test.svc.cluster.local (10.99.107.204): 56 data bytes

#Access Applications within and across namespaces

#Since there is no CURL or nslookup tool installed in pods,

#lets use NetworkingImage - radial/busyboxplus:curl

kubectl run curl --image=radial/busyboxplus:curl -i --tty

#Same ns - default

[ root@curl:/ ]$ nslookup frontend

[ root@curl:/ ]$ curl http://frontend.default.svc.cluster.local

curl http://10-36-0-14.default.pod.cluster.local

#Diff ns - test

[ root@curl:/ ]$ nslookup my-nginx.test

[ root@curl:/ ]$ curl [http://my-nginx.test.svc.cluster.local](http://my-nginx.test.svc.cluster.local/)

Commands executed in CURL pod,

6 nslookup frontend

7 ping frontend

8 nslookup redis

9 nslookup redis-slave

10 nslookup redis-slave.default

11 nslookup my-nginx

12 nslookup my-nginx.test

13 hostname

14 nslookup 192-168-133-199

15 nslookup 192-168-133-199.default

16 nslookup curl

17 curl frontend-5885b9dcbd-4mnpv

18 nslookup frontend  
curl frontend

19 curl http://frontend.default.svc.cluster.local

20 nslookup my-nginx.test

21 curl http://my-nginx.test.svc.cluster.local

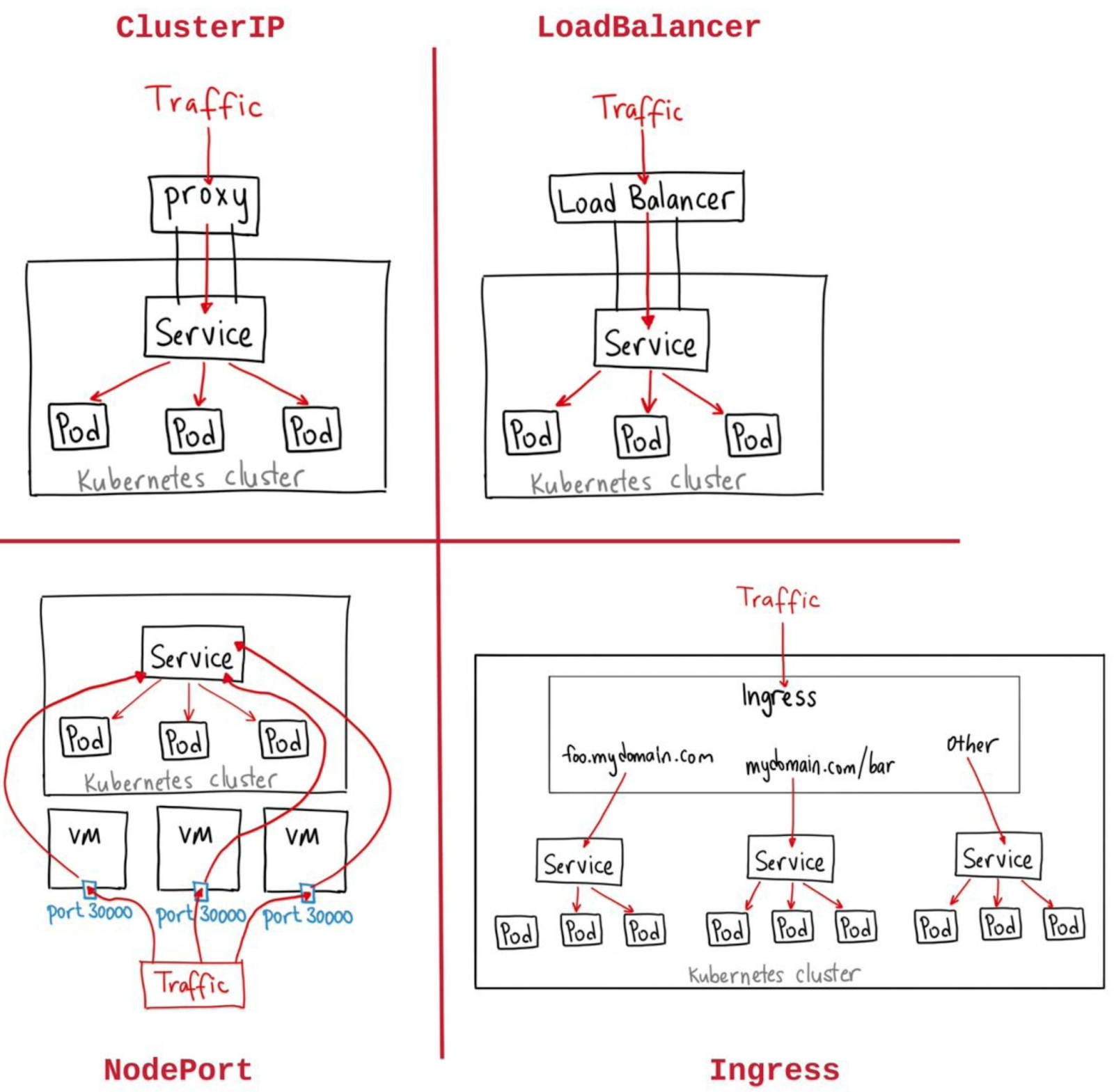
22 nslookup 192-168-133-199.default.pod.cluster.local

23 curl http://192-168-133-199.frontend.default.svc.cluster.local

k delete -f kubesample.yaml

## Service Types

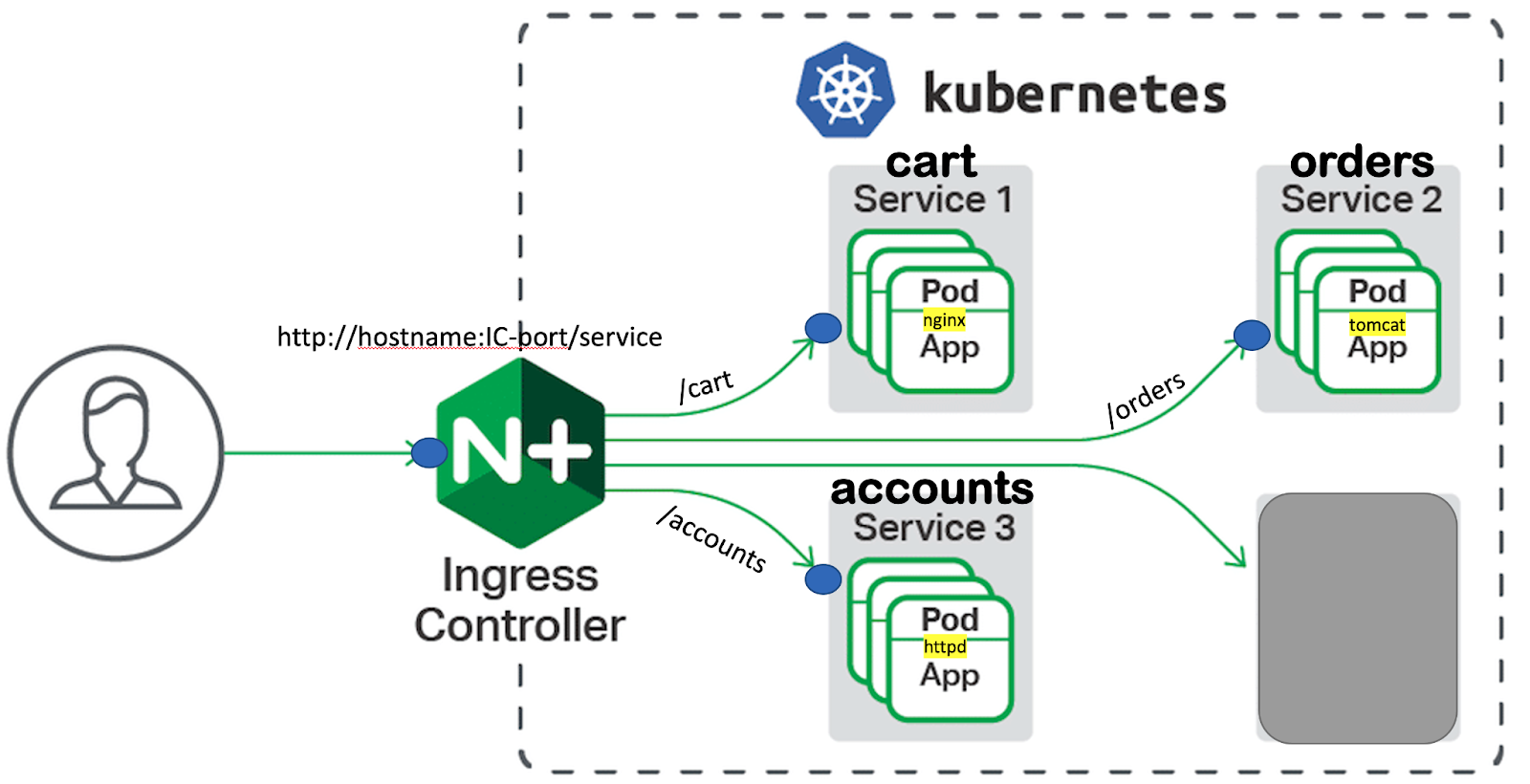
* ClusterIP(default): Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType.
* NodePort: Exposes the Service on each Node's IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service routes, is automatically created. You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>.
* LoadBalancer: Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created



## Ingress

Kubernetes Ingress controllers:

* Accept traffic from outside the Kubernetes platform, and load balance it to pods (containers) running inside the platform
* Are configured using the Kubernetes API to deploy objects called “Ingress Resources”
* Monitor the pods running in Kubernetes and automatically update the load‑balancing rules when pods are added or removed from a service
* **Content-based routing**:
  + *Host-based routing*. For example, routing requests with the host header foo.example.com to one group of services and the host header bar.example.com to another group.
  + *Path-based routing*. For example, routing requests with the URI that starts with /serviceA to service A and requests with the URI that starts with /serviceB to service B.



kubectl apply -f<https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/baremetal/deploy.yaml>

k get all -n ingress-nginx

Get nginx controller endpoint

kubectl get svc -n ingress-nginx ingress-nginx-controller

# Note down the Port number 80:**32184**/TCP,443:32649/TCP

ingress-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: cart

name: cart

spec:

replicas: 1

selector:

matchLabels:

app: ing1

template:

metadata:

labels:

app: ing1

spec:

containers:

- image: nginx

name: nginx

ports:

- containerPort: 80

---

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: orders

name: orders

spec:

replicas: 1

selector:

matchLabels:

app: ing3

template:

metadata:

creationTimestamp: null

labels:

app: ing3

spec:

containers:

- env:

- name: TOMCAT\_PASSWORD

value: rootroot

image: bitnami/tomcat

name: tomcat

ports:

- containerPort: 8080

---

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: accounts

name: accounts

spec:

replicas: 1

selector:

matchLabels:

app: ing2

template:

metadata:

labels:

app: ing2

spec:

containers:

- image: httpd

name: httpd

ports:

- containerPort: 80

---

k apply -f ingress-deployment.yaml

kubectl expose deployment cart --name cart --port 80

kubectl expose deployment orders --name orders --port 80 --target-port 8080

kubectl expose deployment accounts --name accounts --port 80

Ingress.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: ingress-demo

annotations:

kubernetes.io/ingress.class: "nginx"

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

spec:

rules:

- http:

paths:

- path: /cart

pathType: Prefix

backend:

service:

name: cart

port:

number: 80

- path: /account

pathType: Prefix

backend:

service:

name: accounts

port:

number: 80

- path: /orders

pathType: Prefix

backend:

service:

name: orders

port:

number: 80

k apply -f ingress.yaml

Reference:

<https://kubernetes.github.io/ingress-nginx/examples/rewrite/>

| nginx.ingress.kubernetes.io/rewrite-target | Target URI where the traffic must be redirected |
| --- | --- |

k describe ingress ingress-demo

How to get IngressPort?

kubectl get svc -n ingress-nginx ingress-nginx-controller

# Note down the Port number 80:**32184**/TCP,443:32649/TCP

In the Master Browser, http://<master-hostname>:ingressPort/service

http://ip-172-31-19-216:**32184**/cart

http://ip-172-31-19-216:**32184**/orders

[http://ip-172-31-19-216:](http://ip-172-31-19-216:31176/accounts)**32184**/accounts

# Storage

kubectl explain pod.spec.volumes

kubectl explain PersistentVolume and PersistentVolumeClaim

Download all the yaml files in Storage folder,

## Non-Persistent/Ephemeral Volume

### emptyDir

#Demo01: emptyDir Volume

k apply -f pod-vol-emptyDir.yaml

k get pod sharevol

k exec sharevol -c c1 -- ls /tmp

k exec sharevol -c c2 -- ls /tmp

k exec sharevol -c c2 -- touch /tmp/file1

#Delete and recreate the pod to check data

k exec sharevol -c c2 -- ls /tmp

#Data will not be there

### Host Directory

#Demo02: Directory Volume

k apply -f pod-vol-dir.yaml

k get pod webapp

k exec webapp -- ls /log

k exec webapp -- touch /log/k8s.txt

k get pods -o wide | grep webapp

#With the worker node identified, Get into that worker node

ls /var/log/

#check for k8s.txt file and add some text

sudo vi /var/log/k8s.txt

k exec webapp -- cat /log/k8s.txt

#file content will be same

## Volume Config

### ConfigMap

apiVersion: v1

kind: Pod

metadata:

name: configmap-pod

spec:

containers:

- name: mypod

image: redis

ports:

- containerPort: 80

volumeMounts :

- name: sample

mountPath: /home/labsuser/configfiles/

readOnly: true

volumes :

- name: sample

configMap:

name: game-demo

### Secret

apiVersion: v1

kind: Pod

metadata:

name: secret-pod

spec:

containers:

- name: mypod

image: redis

volumeMounts:

- name: foo

mountPath: "/etc/foo"

readOnly: true

volumes:

- name: foo

secret:

secretName: mysecret

## Persistent Volume

k explain pod.spec.volumes | grep persistentVolumeClaim

k explain pv.spec

k explain pv.spec.cinder

k explain pv.spec.awsElasticBlockStore

#RWO can be used one containers

k apply -f pv.yaml

k get pv

## Persistent Volume Claim

#PV Status before PVC creation

k get pv

k apply -f pvc.yaml

#PVC and PV Status after PVC creation

k get pvc

k get pv

#Pod using PVC

#PVC Status before pod creation

k get pvc

k apply -f pod-pvc.yaml

#PVC Status after pod creation

k get pvc

#Verify Volume by creating a file

k exec -it task-pv-pod -- touch /usr/share/nginx/html/index.html

k exec -it task-pv-pod -- ls /usr/share/nginx/html/

Identify the worker node, where pod is running/scheduled

k get pod task-pv-pod -o wide

In worker nodes, Example: worker02

ls /mnt/data

index.html

k delete -f pod-pvc.yaml

Verify file exists in worker node, Example: worker02

ls /mnt/data

index.html

k apply -f pod-pvc.yaml

k exec -it task-pv-pod -- ls /usr/share/nginx/html/

index.html

## Access Modes

### ReadWriteOnce

Above examples used ReadWriteOnce mode

### ReadWriteMany

#using RWM, same PVC can be attached to many container

k apply -f pv-rwm.yaml

k get pv

k apply -f pvc-rwm.yaml

k get pvc

k get pv

k apply -f pod-pvc-rwm.yaml

k get pod sharevol-rwm

k exec sharevol-rwm -c c1 -- ls /tmp

k exec sharevol-rwm -c c2 -- ls /tmp

# Resource quota

Create ns - quotaz

k create ns quotaz

apiVersion: v1

kind: ResourceQuota

metadata:

name: mem-cpu-demo

spec:

hard:

requests.cpu: "1"

requests.memory: 1Gi

limits.cpu: "2"

limits.memory: 2Gi

k apply -f quota -n quotaz

#### Pod 01

apiVersion: v1

kind: Pod

metadata:

name: quota-mem-cpu-demo

namespace: quotaz

spec:

containers:

- name: quota-mem-cpu-demo-ctr

image: nginx

resources:

limits:

memory: "800Mi"

cpu: "800m"

requests:

memory: "600Mi"

cpu: "400m"

#### Pod 02

apiVersion: v1

kind: Pod

metadata:

name: quota-mem-cpu-demo-2

namespace: quotaz

spec:

containers:

- name: quota-mem-cpu-demo-2-ctr

image: redis

resources:

limits:

memory: "2Gi"

cpu: "800m"

requests:

memory: "700Mi"

cpu: "400m"

ERROR: forbidden: exceeded quota: mem-cpu-demo,

requested: requests.memory=700Mi,used: requests.memory=600Mi, limited: requests.memory=1Gi

# Troubleshooting

## Kubernetes cluster

### **Troubleshooting the control plane**

### **Listing the nodes in a cluster**

First thing to check if your cluster is working fine or not is to list the nodes associated with your cluster.

kubectl **get** nodes

Make sure that all nodes are in Ready state.

### **List the control plane pods**

If your nodes are up and running, the next thing to check is the status of Kubernetes components. Run,

kubectl **get** pods -n kube-system

If any of the pods are *restarting or crashing*, look into the issue. This can be done by getting the pod's description. For example, in my cluster *kube-dns* is crashing. In order to fix this first check the deployment for errors.

kubectl **describe** deployment -n kube-**system** kube-dns

Log files

Master

If your deployment is good, the next thing to look for is log files. The locations of log files are given below...

/var/log/kube-apiserver.log - For API Server logs

/var/log/kube-scheduler.log - For Scheduler logs

/var/log/kube-controller-manager.log - For Replication Controller logs

If your Kubernetes components are running as pods, then you can get their logs by following the steps given below. Keep in mind that the *actual pod's name may differ* from cluster to cluster...

kubectl logs -n kube-**system** -f kube-apiserver-node1

kubectl logs -n kube-**system** -f kube-scheduler-node1

kubectl logs -n kube-**system** -f kube-controller-manager-node1

## Worker Nodes

In your worker, you will need to check for errors in kubelet's log...

sudo journalctl -u kubelet

sudo service kubelet status

If not running,   
sudo service kubelet restart

## Application Troubleshooting

### Basics

1. Yaml with issue - typo

apiVersion: app/v1

kind: Deployment

metadata:

name: nginx

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: ngnx

spec:

containers:

- name: nginx

image: ngnix:latest

ports:

- containerPort: 80

Yaml with Fix

apiVersion: apps/v1 # Fix 1/3

kind: Deployment

metadata:

name: nginx

labels:

app: nginx

spec:

replicas: 1

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx # Fix 2/3

spec:

containers:

- name: nginx

image: nginx:latest # Fix 3/3

ports:

- containerPort: 80

1. ImagePullBackOff Error

apiVersion: apps/v1

kind: Deployment

metadata:

name: confused-imager

spec:

selector:

matchLabels:

app: whatever

replicas: 1

template:

metadata:

labels:

app: whatever

spec:

containers:

- name: something

image: simpleservice:0.5.0

**Fix:**

Pod events will show “ImagePullBackOff”

apiVersion: apps/v1

kind: Deployment

metadata:

name: confused-imager

spec:

selector:

matchLabels:

app: whatever

replicas: 1

template:

metadata:

labels:

app: whatever

spec:

containers:

- name: something

**image: alpine**

1. Out of Memory (OOM)

apiVersion: apps/v1

kind: Deployment

metadata:

name: wegotan-oomer

spec:

selector:

matchLabels:

app: oomer

replicas: 1

template:

metadata:

labels:

app: oomer

spec:

containers:

- name: greedymuch

image: centos:7

command:

- sh

- '-c'

- "sleep 5 && yes | tr \\n x | head -c 500m | grep n && sleep 1000"

resources:

limits:

memory: 400M

- name: shell

image: centos:7

command:

- sh

- '-c'

- sleep 1000

Fix:

apiVersion: apps/v1

kind: Deployment

metadata:

name: wegotan-oomer

spec:

selector:

matchLabels:

app: oomer

replicas: 1

template:

metadata:

labels:

app: oomer

spec:

containers:

- name: greedymuch

image: centos:7

command:

- sh

- '-c'

- "sleep 5 && yes | tr \\n x | head -c **500m** | grep n && sleep 1000"

resources:

limits:

**memory: 600M**

- name: shell

image: centos:7

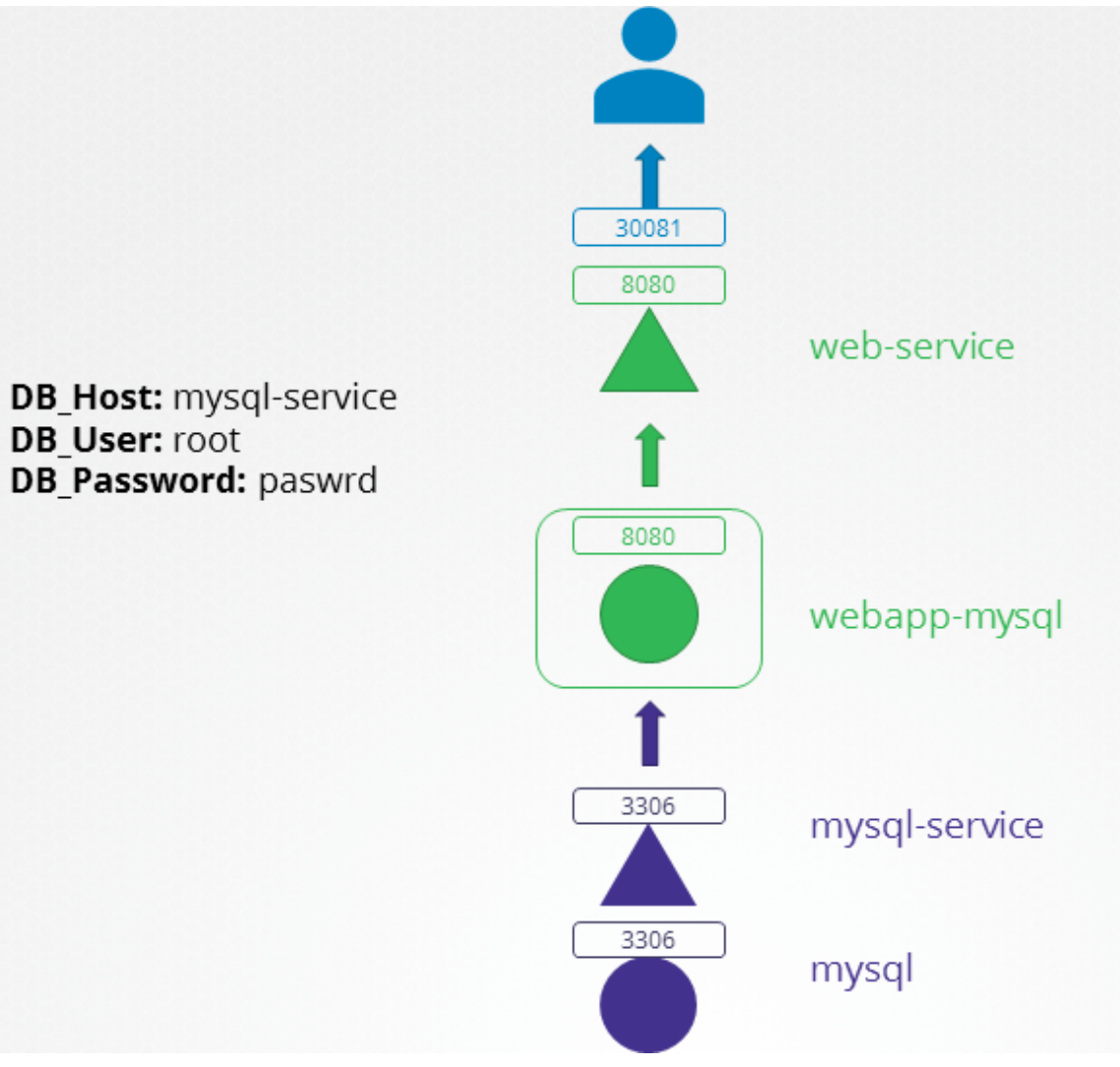
command:

- sh

- '-c'

- sleep 1000

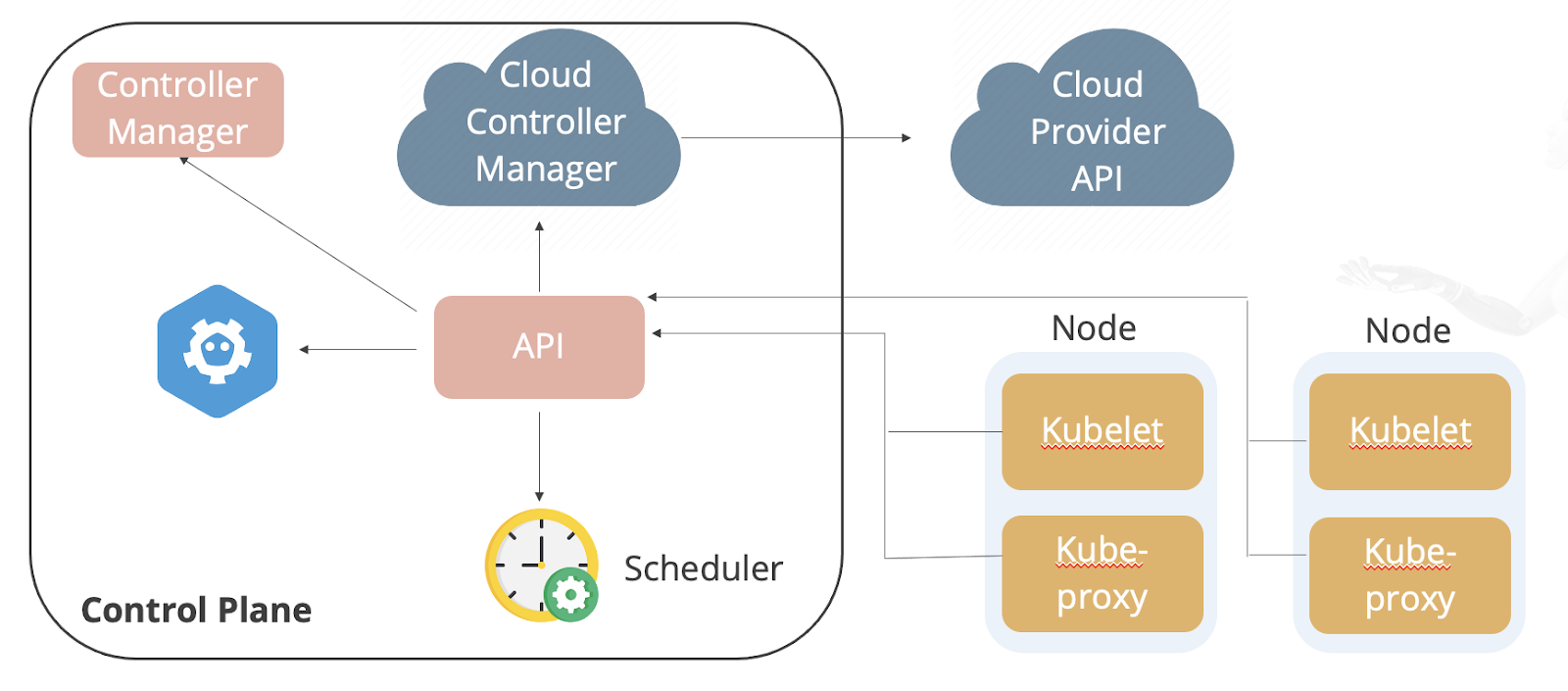
### 2-Tier App



Access the app url - [http://](about:blank)localhost:30081 for troubleshooting

# Azure Kubernetes Service

**Cloud Controller Manager (CCM)**

****

## 

## Create AKS

kubectl get cluster-info

kubectl get nodes

Look for a master node :)

kubectl describe node <node-name>

Observe the Labels being created based on Region and Zones

## Workloads

## Rolling update and rollback

## Service

### Creating Load Balancer - Service

## Storage - PV and PVC

### Creating PVC with dynamic PV

Creating sc

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

labels:

addonmanager.kubernetes.io/mode: EnsureExists

kubernetes.io/cluster-service: "true"

name: managed-premium-01

resourceVersion: "391"

uid: 08be1a5c-7fcf-4603-91eb-354e2fe27fa3

parameters:

cachingmode: ReadOnly

kind: Managed

storageaccounttype: Premium\_LRS

provisioner: disk.csi.azure.com

reclaimPolicy: Delete

volumeBindingMode: Immediate

Creating PVC

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: azure-managed-disk

spec:

accessModes:

- ReadWriteOnce

storageClassName: managed-premium-01

resources:

requests:

storage: 5Gi

k get pvc

k get pv