Kubernetes installation

Make sure install docker-ce completely

Docker installation

1. sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg-agent \

software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io -y

systemctl start docker

systemctl enable docker

1. swapoff -a

setenforce 0 (sed -i 's/enforcing/disabled/g' /etc/selinux/config)

sudo apt-get update && sudo apt-get install -y apt-transport-https curl

sudo apt-get update && sudo apt-get install -y apt-transport-https

2.curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

echo "deb https://apt.Kubernetes.io/ Kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/Kubernetes.list

sudo apt-get update

sudo apt-get install -y kubectl kubeadm kubelet

sudo apt-mark hold kubelet kubeadm kubectl

5.kubeadm init (--pod-network-cidr=10.240.0.0/16) # pod network is optional

6 mkdir -p $HOME/.kube

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

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

7. install kubeadm in both nodes (repeate the same step 2)

9. Must install the addon CNI (kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml)

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

or chose add-on (https://Kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/)

8. kubeadm join 10.0.0.32:6443 --token 70incq.2zq2wjxvu0psomat \

--discovery-token-ca-cert-hash sha256:a99e2188a408e6171d81c7855a9687c0023aaa3e51027a0d3866c21bbb00015a

always run this command as root

9. check the nodes are correcctly installed or not

kubectl get nodes

kubectl get pods --all-namespaces

10. if you last the token then run the following command to get the token [kubeadm token list ]

11. If you lost --discovery-token-ca-cert-hash valus also you can get it by runing following commands sequentially

openssl x509 -pubkey -in /etc/Kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | \

openssl dgst -sha256 -hex | sed 's/^.\* //'

12. kubectl create -f podname.yml

to deploy a pod

to know the which node the pod running is

#kubectl get pod -o wide

@Notes: a. pods delete method must be the way we create pods.

ex: if you create pod via replica controller then delete throught delete rc <name>

if you create pod via deployment method then delete throught delete deploy <name>

b. we can expose the app with ingress also but it is not a service type.

c. to delete the node drine the node

$kubectl drain <node name> --delete-local-data --force --ignore-daemonsets

$kubectl delete node <node name>

**13. Replication Controllers**

1. The link between replicate set and pods are labels

\*\*Replicaset vs Replication controller

a.there are multiple ways that container can crash

b.Replica basically used to Realiability, Load Balancing, Auto Scalling

Replicaset Replication controller

1...next gen of rc

2. Set based selector 2. Equality based selector

Labels & selectors

Labels: set of pods defined (Ex: app, tier, env)

Selectors: Two Types:

Equality based, set based

=,==,!= in, notin,exists

ex: environment= production environement in (production,qa)

tier != frontend tier notin (frontend,backend)

$ kubelet get po -l environemnt=production $ kubelet get po -l environment in production

in mainfest: in mainfest:

selector: selector:

environemnt: production MatchExpersion

tier: frontend -{key: environmnet, oreratoe: in, values:[prod,qa]}

-{key:tier. operator`; notin, values:[frontend,backend]

We use match lables with new resources

a. Replicasets

b.. Deploymnets

c. Jobs

D. Daemonset

Diff b/w match lables vss match expressions

Match lable used when lable key value is one

ex: app: nginx-app

match expression used when multiple key values exisiting

ex: - {key: tier, operator: in, values:[frontend]}

**Replica:**

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: tomcat

spec:

replicas: 2

selector:

matchLabels:

app: tomcat-app

# matchExpressions:

# - {key: tier, operator: In, values: [frontend]}

template:

metadata:

name: tomcat-pod

labels:

app: tomcat-app

tier: frontend

spec:

containers:

- name: tomcat-container

image: tomcat

ports:

- containerPort: 80

14. If you delete the pods with delete command $kubectl delete po <pods name>

This will automatically create another pods because In some cases it could also be running due to a job or daemonset.

Check the following and run their appropriate delete command.

$kubectl get daemonsets.app --all-namespaces

$kubectl get deployments --all-namespaces

$kubectl delete -n NAMESPACE deployment DEPLOYMENT

\* first delete the name replicaset

$ kubectl delete replicaset.apps/nginx-deployment-54

15. When we create deployment it will autiomatically create repilica set

Deploymnets:

Upgarding the app from version 1 to 2

a. upgrading with zero downtime

b. upgrading sequentially, one after the other

c. pause and resume upgrade process

d. Rollback upgrade to previous stable release

the problem can be sloved with deployments

1... deploymentents are like any other controllers but these are accountable for updates and roll backs.

2. Updates and Roll backs canbe done with changes version numbers in manifest file

3. For example we have five instance running we can deploy a app into them using with replicaset but we cannot update and roll back

4. With deployment mainfest file we can manage number of replicaset's and number of pods, (No need to write 3 different manifest file for 3 different types of deployments)

**Features**

1. Multiple repilica (When we create deploymentents we can multiple repilicas for high avaliability and load balancing)

2. Upgrading

3. Rollack

4. Scale up and down

5. Pause and Resume (used to test new applications)

Deploymnet types

1. Recreate (Stop V1 then deploy V2 so there will be a downtime)

2. Rollaing Updates (Rolling one after the other instances Default Update strategy)

3. Canary (its a type of rolling update but difference comes if we have 10 intances first deploy v2 then do some test if everything working fine then deploy v2 to remaing 8(usefull for testing))

4. Blue/Green (deployment of v2 to same no of instances running v1 if everting fine then traffic willbe switched to v2 at load balancer level)

Manifest file:

apiVersion: apps/v1

kind: deployment

metadata:

name: tomcat

labels: #label is optional but its usefull wjen managing deploymnet

app: tomcat-pod

spec:

repilicas: 2 #default value 1

selector: #used to manage pods

matchLabels:

app: tomcat

template: # this is the pod template

metadata:

labels:

app: tomcat # pod name

spec:

containers:

- name: tomcat-container

image: tomcat

port:

- containerPort:80

$kubectl create -f tomcat.yml

$kubectl get deploy -l app=tomcat-pod # when we do deploymnet it will automatically create replica set

#deployment.apps/tomcat-deployment created

$kubectl get rs -l app=tomcat-pod

$kubectl describe deploy tomcat-deployment

Use Case:

Update:

tyep1. $kubectl set image deploy tomcat-deployment tomcat-container=tomcat:9.0.30

type2. $kubectl edit deploy tomcat-deployment #edit the repilica count and version number then save it close it

$kubectl rollout status deployment/tomcat-deployment

Rollback:

type1: $kubectl rollout history deploymnet/tomcat-deployment

$kubectl rollout undo deployment/tomcat-deployment

ScalingUp:

$kubectl scale deployment tomcat-deployment --replicas=5

ScaleDown:

Same command but change the replica count to desired

$kubectl delete -f tomcat-deploymnet.yml

Services:

1. How does front end webapp eposed to outside world?

2. How do front end webapp connected to outside world?

3. How do we reslove Pod Ip changes, when they die?

a. Why we need Services?

Some cases we need to expose frontend application not backend application and in some other cases we use services (a1,a2,a3 are some use cases)

b. what is a service?

Grouping of pods within the cluster. Like frontend backend service dicovery between pods and connection between pods.

c. types of services?

3 types

1. Cluster Ip

2. Node Port

3. Load Balancer

a1. is it possible to have a perment IP addess?

a2. how do various components conect & communicate?

a3. how do applications are exposed to outside world?

NodePort:

1. Why we need port? To expose to outside world

2. What is node port? It’s a port used to expose app

3. Types of node port?

4. Scenarios?

Downsides:

a. You can only have one service per port

b. you can user only 30000-32767 port number

c. you your ip changes you have to deal with that.

Manifest file:

apiVersion: v1

kind: service

metadata:

name: my-service

labels: app=tomcat

spec:

selector:

app: tomcat #this should be equal to template app name

type: NodePort

ports:

- nodePort: 31000

port: 8080 #service port

targetport:8080 # actual container running port

$kubectl create -f tomcat-deploy.yml

$kubectl create -f tomcat-service.yml

$kubectl get service -l app=tomcat

$kubectl describe svc my-service

$kubectl delete svc my-service # pods delete because of we have created labels init

**LoadBalancer**

a. Why we Need

b. LoadBalancer

1. For example if you have your app running on different nodes and you have to access the app. in this scenarios ip of app changes everytime so to remove this constrain we need load balancer

b. in this kind of situations we need load balancer.

Mainfest:

apiVersion: v1

kind: Service

metadata:

name: load-service

labels:

app: tomcat

spec:

selector:

app: tomcat-app # this should be equal to pod name

type: LoadBalaner

ports:

- nodePort:31000

port:8080

targetPort:8080

We can do this with a command also:

$kubectl expose deploy tomcat-deployment --name=tomcat-service --port:80 --target-port=80 --type=LoadBalancer

$kubectl create -f tomcat-deployment.yml

$kubectl create -f tomcat-loadbalancer.yml

$kubectl get service

$kubectl describe service load-service

**ClusterIP (Default service type)** # It will work only with in the cluster

In orginations we donot provide the database exposed to the outside world but they should connect internally for better function of the App

In this case we use clusterIp

Example:

Guest book app

Require:

1.Frontend page

2.Backend Database

3.Service

1. Conditons:

a. db donot expose to outside world

b. high avliability (Load Balancer)

Solution:

a. Redis DB master (Deploymnets)

b. Redis App Slave (")

c. Frontend App ("")

4. SVC redis DB server (Services)

5. SVC redis DB slave (")

6. SVC frontend (")

To access backend pods and connect with frontend pods we need frontend pods

Mainfest: Redis DB master Deploymnet

apiVersion: apps/v1

kind: Deployment

metadata:

name: redis-masterdb

labels:

app: db-master

spec:

replicas: 1

selector:

matchLabels:

app: db-master

role: master

tier: backend

template:

metadata:

labels:

app: db-master

role: master

tier: backend

spec:

containers:

- name: db-master

image: k8s.gcr.io/redis:e2e

ports:

- containerPort: 6379

Mainfest: redis DB slaves Deployment

apiVersion: apps/v1

kind: deployment

metadata:

name: db-slaves

labels:

app: db-slaves

spec:

replicas: 2

selector:

matchLabels:

app: db-slave

role: slave

tier: backend

template:

metadata:

labels:

app: db-slave

role: slave

tier: backend

spec:

containers:

- name: slave

image: gcr.io/google\_samples/gb-redisslave:v1

resources:

cpu: 100m

memory: 100Mi

port:

- containerPort: 6379

Mainfest: frontend pods

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend

labels:

app: guestbook

spec:

replicas: 2

selector:

matchLabels:

app: guestbook

tier: frontend

template:

metadata:

labels:

app: guestbook

tier: fronend

spec:

containers:

- name: php-redis

image: gcr.io/google-samples/gb-frontend:v4

port:

- containerPort: 80

Mainfest: Services-master

apiVersion: v1

kind: Service

metadata:

name: db-master-service

labels:

app: db-master

role: master

tier: backend

spec:

selector:

app: db-master

tier: backend

ports:

- name: redis

port: 6379

targetPort: 6379

type: ClusterIP

Manifest: Services slave

apiVersion: v1

kind: Service

metadata:

name: db-master-service

labels:

app: redis-service

role: salve

tier: backend

spec:

ports:

- port: 6379

targetPort: 6379

type: ClusterIP

selector:

app: redis

role: slave

tier: backend

Manifest: service loadbalancer

apiVersion: v1

kind: Service

metadata:

name: froentend svc

labels:

app: guestbook

tier: frontend

spec:

ports:

- port: 80

type: LoadBalancer

selector:

app: redis

tier: frontend

**Volumes:**

**emptyDir Volume:**

1.it will delete automatically when the pod is deleted and also data in the pod will be lost.

mainfest file:

apiVersion: v1

kind: pod

metadata:

name: test-ed

spec:

containers:

- image: k8s.gcr.io/test-webserver

name: test-container

volumeMounts:

- name: cache-volume

mountPath: /cache

volumes:

- name: cache-volume

emptyDir: {}

$

**hostpath:**

a. mounts a file or directory from the host node filesystem into your pod

b. remains even the pod terminated

c. similar to docker volume

d. host issues migh cause problem to hostpath

Mainfest file:

apiVersion: v1

kind: pod

metadata:

name: redis-hostPath

spec:

containers:

- image: redis

name: redis-container

volumeMounts:

- name: test-vol

mountPath: /test-mnt

volumes:

- name: test-vol

hostPath:

path: /test-vol

$k create -f <filename>.yml

$k get po

$k exec redis-hostPath df /test-mnt

$k exec redis-hostPath -it -- /bin/sh # login into container

gcePersistentDisk: almost aws and azure also same conditions.

a. unlike emptyDir the data will be preserved even after the pod is removed.

b. they canbe mounted on RW only on one node and RO on many number of nodes.

Restrictions:

a. Must create disk before the use of disk

b. nodes on which pod runing must be a GCP vm

c. Zone must be equal

**Persistent Volumes:**

a. Why persistent volume?

There are differne storage infra so we need to create different type of api/yml files to intract with them, this will create unnecessary over head

so to over come this there is persistent volumes.

they are two types:

1. Persistent volume

2. Persistent Volume claim

Persistent Volume: PV

1. It is a piece of storge in cluster.

2. These are present out of the life cycles also.

Persistent Volume claim: PVC

it is a request for storage

ex: a developer need a storage space with some access policy such as read and write control.

Life cycle:

provisioning ----> Binding ---> Using ---> Reclaiming

Provisioning:

The PV can be provisioning in 2 ways

1. Static {there PV are created by system admin}

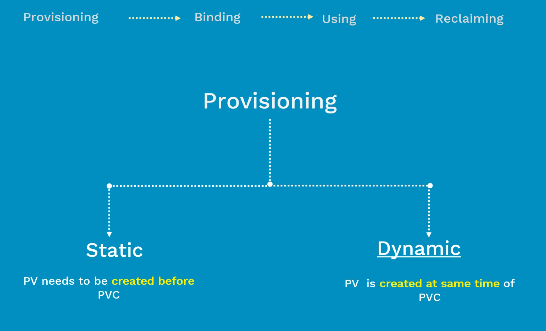
2. Dynamic

1. Static pv:

This should be created before the PVC created.

2. Dynamic Pv

PV is created at the same time of pvc



Binding:

After provision of PV when a developer create a request for storage (PVC) the a loop on the Kubernetes master check for the availability of PV

Using:

After submitting the PVC Kubernetes api inspect and allocate the volume then start using it.

Reclaiming:

When the user done with the volume they can delete the PVC, so the PV can be reclaimed. The volume can be reclaimed, recreated, deleted, and retained.

**Static PV:**

**PersistanceVolume:**

ApiVersion: v1

kind: Persistentvolume

metadata:

name: pv-gce

spec:

capacity:

storage: 15Gi

accessModes:

* ReadWriteOnce

storageClassName: slow

gcePersistanceDisk:

pdName: my-disk-123

fstype: ext4

Persistence Claim:

apiVersion: v1

Kind: PersistentClaim

Metadata:

name: persistentVolumeClaim

spec:

resource:

requests:

storage: 15Gi

accessModes:

-ReadWriteOnce

storageClassName: slow

Referencing Claim in pod:

apiVersion: v1

kind : pod

metadata:

name: pv-pod

spec:

containers:

- name: test-container

image: nginx

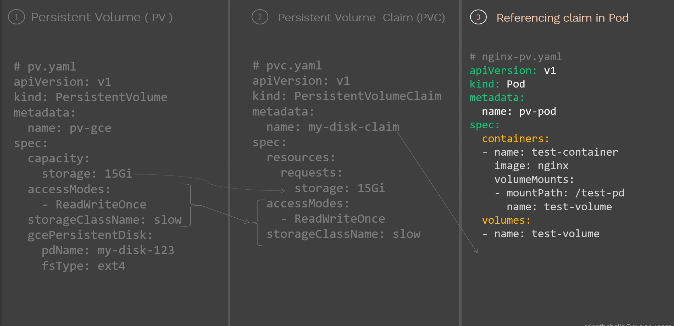
volumeMounts:

* mountPath: /test-pd

name: test-volume

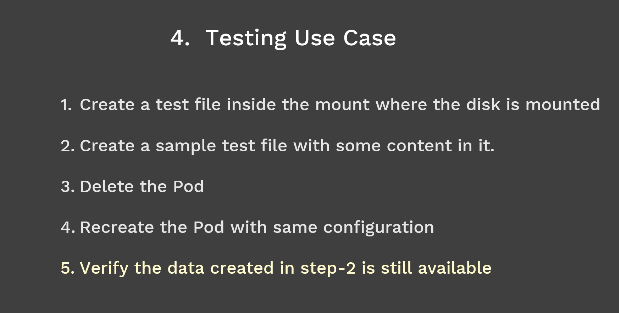
volumes:

- name: test- volume



To create pod: $kubectl create –f <name>

**Testing and use cases:**



Step 1:

$kubectl exec pv-pod –it /bin/bash

$df –h /test-pod

Step 2:

$touch file1.txt

$ cat “this is mount test” > file1.txt

$ exit

Step3: $kubectl delete –f <name> # name match with the first create name

Step4: $kubectl create –f <name> # name match with the first create name

Step5: $kubectl exec pv-pod –it /bin/bash

$cat file1.txt

Output: if you get “this is mountu test “then good to go.

**Dynamic PV:**

1. storage class
2. persistent volume claim
3. reference claim in pod

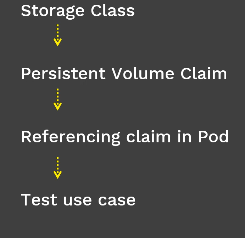
* In dynamic PV no need to create a PV, when we create a PVC it will automatically create a PV.
* In dynamic PV we don’t create PV’s manually we create storage class.



This storage class is one time setup. The beautify of the storage class dev team no need to worry about how much storage available inside the storage pool.

The admin can set default storage class as <fast, slow, Distributed>

When the described class to available then dev team can raise a PVC then it will automatically create the required PV.



Storage Class: assume we have different types of storage class available like SSD, HDD, NFS.

So we can create different types of storage class like fast slow medium.

1. Storage class:

apiVersion: v1

kind: StorageClass

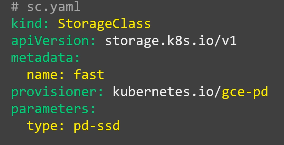
metadata:

name: fast

provisioner: kubernetes.io/gce-pd #(aws-ebs)

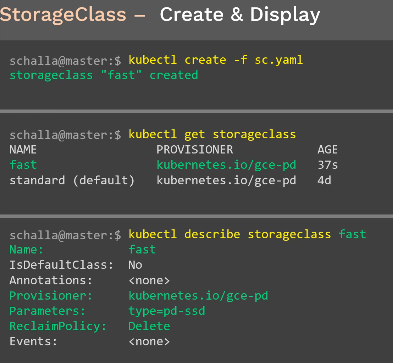
parameters:

type: pd-ssd



$kubectl create –f sc.yml

$kubectl get storageclass



1. Persistent volume claim:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-disk-claim-1

spec:

resource:

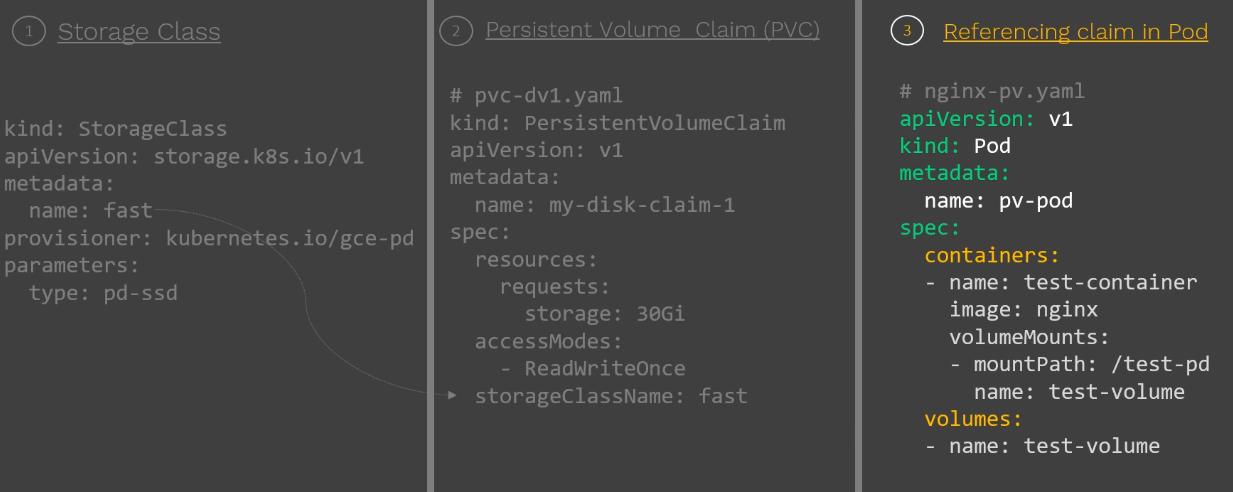
requests:

storage: 30Gi

accessModes:

-ReadWriteOnce

storageClassName: fast



**ConfigMaps**

Config Maps: used to separate your configuration from pods

To manage the configuration in the container we basically use this configmaps

Use case: as every pod contains some configuration information; if in case we need to give access to the container or expose it to outside world we have to protect the data, in that case we use configmaps.

Store configuration data as Key-Value pairs.

You must create the configmaps before referencing it to the pod Spec

Key = Name of the file

Values = Content of the file

Config maps can be created in 3 different ways:

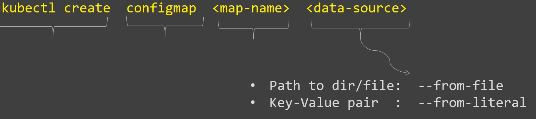
1. Configuration files

2. Command line argument

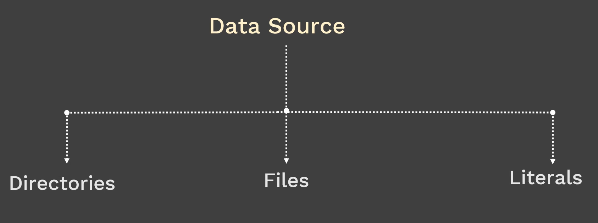
3. Environment Variable

Syntax:

Kubectl create configmap <map-name> <data-source>

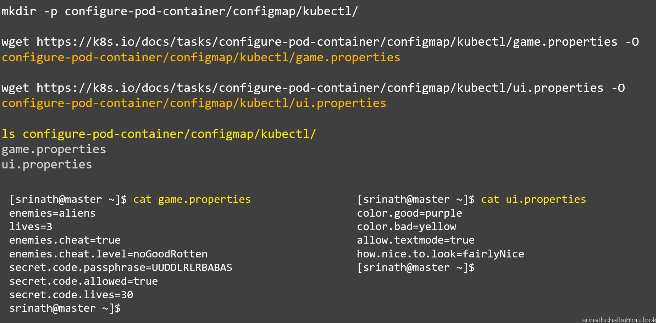


Data Source:



Directories:

Mkdir –p configure-pod-container/configmap/kubectl/



$kubectl create configmap game-config –from-file=configure-pod-conainer/config/kubectl/

$kubectl get configmap game-config

**Files:**

$curl –OL https://k8s.io/examples/pods/config/redis-config

$kubectl create configmap redis-config –form-file=redis-config

**Manifest:**

apiVersion: v1

kind: Pod

metadata:

name: redis

spec:

containers:

- name: redis

Image: redis

volumeMounts:

-mountPath: /redis-master

Name: config

Volumes:

* Name: config

configmap:

name: redis-config

items:

-key: redis-config

Path: redis.conf

$kubectl exec –it redis redis-cli

**Literals:**

$kubectl create configmap special-config –from-literal=special.how=very

apiVersion: v1

kind: pod

metadata:

name: test-pod

spec:

containers:

-name: test-container

Image: busybox

command: [“/bin/sh” ,“-c”, “env” ]

env:

-name: SPECIAL\_LEVEL\_KEY

valueFrom:

configMapKeyRef:

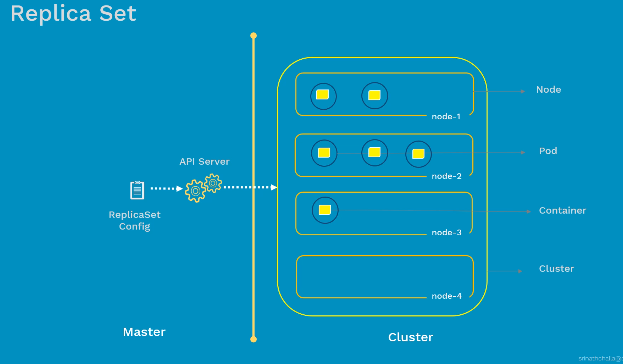
name: special-config

key: special.how

restartPolicy: Never

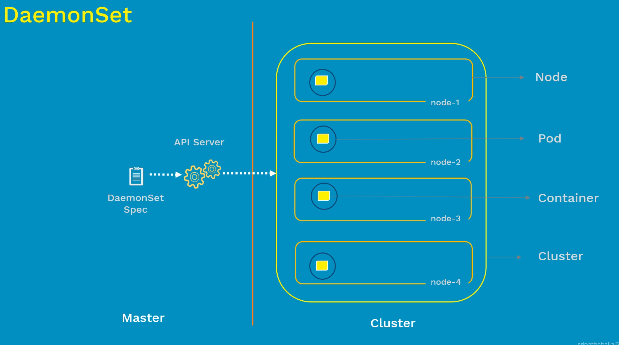
**DaemonSet:**

Replicaset ensure that specified no of pods running inside the cluster.



Ex: we have 3 nodes, pods are running on the 2 nodes only, what if we have to deploy monitoring app to the every node, in such case replicaset will nt work.

Solution is DaemonSet.



Problem with the deamonset is If we delete the deamon set it will delete the all pods in the cluster.

Manifest:

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: flurnntd-ds

spec:

template:

metadata:

labels:

name: fluentd

spec:

cotainers:

-name: fluentd

Imag: fluentd

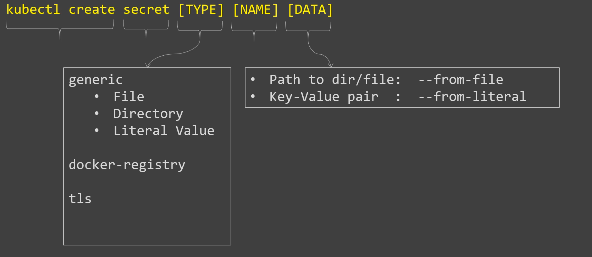
selector:

matchLabels:

* Name: flunted

**Secrets:**

$kubectl create secret [TYPE] [NAME] [DATA]



$echo –n ‘admin’ > ./username.txt

$echo –n ‘234567890’ > ./password.txt

$kubectl create secret generic user-pass –from-file=./username.txt –from-file=./password.txt

$kubectl get secrete

Manually:

$echo –n “admin” | base64

sfrgretgegt4r

$echo –n “1234567890” | base64

gett34erfwfw

apiVersion: v1

kind: Secret

metadata:

name: my-secret

type: Opaque

data:

username: sfrgretgegt4r

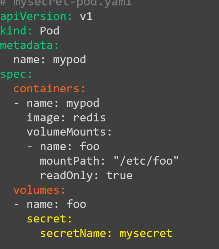
password: gett34erfwfw

$kubectl create –f my-secret.yaml

Decode Secret:

$echo –n ‘sfrgretgegt4r | base64 –decode



 apiVersion: v1

kind: pod

metadata:

name: mypod

spec:

containers:

-name: mypod

image: redis

volumeMount:

-name: volume

mounthPath: /test

readOnly= true

volumes:

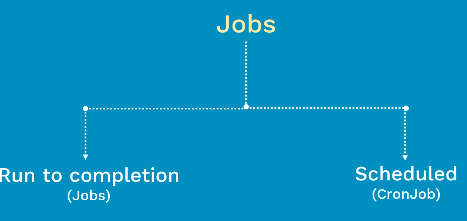
-name: volume

Secret:

secretName: mysecret



**Jobs:**



Run to completion:

There are run completion jobs and these will exit when exit code gets 0.

Each job create one or more jobs, ensure they are successfully terminated.

Can run a multiple pods in parallel, can scale up using kubectl scale command.

Manifest file:

apiVersion: batch/v1

knd: job

metadata:

name: countdown

spec:

template:

metadata:

name: countdown

spec:

containers:

-name: centos

Imahe: centos

command:

* “bin/bash”
* “-c”
* “for I in 9876321; do echo $i;done”

restartPolicy: never

Scheduled: (CronJob)

1. Create separate namespaces

2. Rename the node

3. Endpoint # Imp

4. Magic Variable