## Validating the Setup

You could validate the status of this cluster, health of pods and whether all the components are up or not by using a few or all of the following commands.

To check if nodes are ready

***$ kubectl get nodes***

***$ kubectl get cs***

[ Expected output ]

***$ kubectl get nodes***

NAME STATUS ROLES AGE VERSION

kube-01 Ready master m v1.8.2

kube-02 Ready <none> m v1.8.2

kube-03 Ready <none> m v1.8.2

Additional Status Commands

***$ kubectl version***

***$ kubectl cluster-info***

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

***$ kubectl get events***

It will take a few minutes to have the cluster up and running with all the services.

### Possible Issues

* Nodes are node in Ready status
* kube-dns is crashing constantly
* Some of the systems services are not up

Most of the times, kubernetes does self heal, unless its a issue with system resources not being adequate. Upgrading resources or launching it on bigger capacity VM/servers solves it. However, if the issues persist, you could try following techniques,

Troubleshooting Tips

Check events

***$ kubectl get events***

Check Logs

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

[get the name of the pod which has a problem]

***$ kubectl logs <pod> -n kube-system***

e.g.

***$ kubectl logs kube-dns-545bc4bfd4-dh994 -n kube-system***

Error from server (BadRequest): a container name must be specified for pod kube-dns-545bc4bfd4-dh994, choose one of:

[kubedns dnsmasq sidecar]

***$ kubectl logs kube-dns-545bc4bfd4-dh994 kubedns -n kube-system***

I1106 14:41:15.542409 1 dns.go:48] version: 1.14.4-2-g5584e04

I1106 14:41:15.543487 1 server.go:70] Using

....

## Enable Kubernetes Dashboard

After the Pod networks is installled, We can install another add-on service which is Kubernetes Dashboard.

Installing Dashboard:

$ kubectl apply -f <https://raw.githubusercontent.com/venkat09docs/K8s-Resources/master/kubernetes/resources/dashboard.yaml>

This will create a pod for the Kubernetes Dashboard.

To access the Dashboard in the browser, run the below command

***$ kubectl describe svc kubernetes-dashboard -n kube-system***

Sample output:

Name: kubernetes-dashboard

Namespace: kube-system

Labels: k8s-app=kubernetes-dashboard

Annotations: kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"

v1","kind":"Service","metadata":{"annotations":{},"labels":{"k8s-app":"kubernetes-dashboar

d"},"name":"kubernetes-dashboard","namespace":...

Selector: k8s-app=kubernetes-dashboard

Type: NodePort

IP: 10.110.60.30

Port: <unset> 80/TCP

TargetPort: 9090/TCP

NodePort: <unset> 31000/TCP

Endpoints: 10.40.0.1:9090

Session Affinity: None

External Traffic Policy: Cluster

Events: <none>

Now check for the Load Balancer URL and go to the browser,

The Dashboard Looks like:



**==========================================================**

## Kubernetes Visualizer

In this chapter we will see how to set up kubernetes visualizer that will show us the changes in our cluster in real time.

### Set up

git clone https://github.com/venkat09docs/K8S-RESOURCES/

***$ kubectl apply -f K8S-RESOURCES/kubernetes/resources/deploy/***

[Sample Output]

serviceaccount/kube-ops-view created

clusterrole.rbac.authorization.k8s.io/kube-ops-view created

clusterrolebinding.rbac.authorization.k8s.io/kube-ops-view created

deployment.extensions/kube-ops-view created

ingress.extensions/kube-ops-view created

deployment.extensions/kube-ops-view-redis created

service/kube-ops-view-redis created

service/kube-ops-view created

Get the Load Balancer URL for the service.

***$ kubectl get svc***

Visit the port from the browser. You could add /#scale=2.0 or similar option where 2.0 = 200% the scale.

http://<LB\_URL>/#scale=2.0



**=========================================================**

## Deploying Pods

Life of a pod

* Pending : in progress
* Running
* Succeeded : successfully exited
* Failed
* Unknown

### Resource Configs

Each entity created with kubernetes is a resource including pod, service, deployments, replication controller etc. Resources can be defined as YAML or JSON. Here is the syntax to create a YAML specification.

**AKMS** => Resource Configs Specs

apiVersion: v1

kind:

metadata:

spec:

Spec Schema: https://kubernetes.io/docs/user-guide/pods/multi-container/

To list supported version of apis

***$ kubectl api-versions***

#### Writing Pod Spec

Lets now create the Pod config by adding the kind and specs to schme given in the file vote-pod.yaml as follows.

Filename: K8S-RESOURCES/k8s-code/pods/vote-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: vote

labels:

app: python

role: vote

version: v1

spec:

containers:

- name: app

image: gvenkat/vote:v1

ports:

- containerPort: 80

protocol: TCP

### Launching and operating a Pod

To launch a monitoring screen to see what’s being launched, use the following command in a new terminal window where kubectl is configured.

***$ watch -n 1 kubectl get pods,deploy,rs,svc***

kubectl Syntax:

***$ kubectl apply --help***

***$ kubectl apply -f FILE***

To Launch pod using configs above,

***$ kubectl apply -f vote-pod.yaml***

To view pods

***$ kubectl get pods***

***$ kubectl get po -o wide***

***$ kubectl get pods vote***

To get detailed info

***$ kubectl describe pods vote***

[Output:]

Name: vote

Namespace: kube-system

Priority: 0

PriorityClassName: <none>

Node: kube-02/192.168.12.11

Start Time: Sun, 05 Aug 2018 18:02:00 +0000

Labels: app=python

role=vote

version=v1

Annotations: kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"v1","k

ind":"Pod","metadata":{"annotations":{},"labels":{"app":"python","role":"vote","version":"

v1"},"name":"vote","namespace":"kube-sys...

Status: Running

IP: 10.38.0.4

Containers:

app:

Container ID: docker://ff86b165aa3553503fea11b0161e976ee2e2d45f71aff05c3b0fe896094bc

908

Image: gvenkat/vote:v1

Image ID: docker-pullable://gvenkat/vote@sha256:9195942ea654fa8d8aeb37900be56192

15c08e7e1bef0b7dfe4c04a9cc20a8c2

Port: 80/TCP

Host Port: 0/TCP

State: Running

Started: Sun, 05 Aug 2018 18:03:00 +0000

Ready: True

Restart Count: 0

Environment: <none>

Mounts:

/var/run/secrets/kubernetes.io/serviceaccount from default-token-d7rt4 (ro)

Conditions:

Type Status

Initialized True

Ready True

ContainersReady True

PodScheduled True

Volumes:

default-token-d7rt4:

Type: Secret (a volume populated by a Secret)

SecretName: default-token-d7rt4

Optional: false

QoS Class: BestEffort

Node-Selectors: <none>

Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s

node.kubernetes.io/unreachable:NoExecute for 300s

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal Scheduled 4m default-scheduler Successfully assigned kube-system/vote to ku

be-02

Normal Pulling 4m kubelet, kube-02 pulling image "gvenkat/vote:v1"

Normal Pulled 3m kubelet, kube-02 Successfully pulled image "gvenkat/vote:v1"

Normal Created 3m kubelet, kube-02 Created container

Normal Started 3m kubelet, kube-02 Started container

Commands to operate the pod

***$ kubectl logs vote***

***$ kubectl exec -it vote sh***

Inside the container in a pod

*$ ifconfig*

*$ cat /etc/issue*

*$ hostname*

*$ cat /proc/cpuinfo*

*$ ps aux*

## Troubleshooting Tip

If you would like to know what’s the current status of the pod, and if it’s in an error state, find out the cause of the error, following command could be very handy.

***$ kubectl get pod vote -o yaml***

Let’s learn by example. Update pod spec and change the image to something that does not exist.

***$ kubectl edit pod vote***

This will open an editor. Go to the line which defines image and change it to a tag that does not exist

e.g.

spec:

containers:

- image: gvenkat/vote:xyz

imagePullPolicy: Always

where tag **xyz**  does not exist. As soon as you save this file, kubernetes will apply the change.

Now check the status,

***$ kubectl get pods***

NAME READY STATUS RESTARTS AGE

vote 0/1 ImagePullBackOff 0 7m

The above output will only show the status, with a vague error. To find the exact error, lets get the stauts of the pod.

Observe the **status** field.

***$ kubectl get pod vote -o yaml***

Now the status field shows a detailed information, including what the exact error. Observe the following snippet...

status:

...

containerStatuses:

....

state:

waiting:

message: 'rpc error: code = Unknown desc = Error response from daemon: manifest

for gvenkat/vote:latst not found'

reason: ErrImagePull

hostIP: 139.59.232.248

This will help you to pinpoint to the exact cause and fix it quickly.

Now that you are done experimenting with pod, delete it with the following command,

**$ kubectl delete pod vote**

***$ kubectl get pods***

## Attach a Volume to the Pod

Let’s create a pod for database and attach a volume to it. To achieve this, we will need to

* create a **volumes** definition
* attach volume to container using **VolumeMounts** property

Local host volumes are of two types:  
\* emptyDir  
\* hostPath

We will pick hostPath.

File: db-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: db

labels:

app: postgres

role: database

tier: back

spec:

containers:

- name: db

image: postgres:9.4

ports:

- containerPort: 5432

volumeMounts:

- name: db-data

mountPath: /var/lib/postgresql/data

volumes:

- name: db-data

hostPath:

path: /var/lib/pgdata

type: DirectoryOrCreate

To create this pod,

***$ kubectl apply -f db-pod.yaml***

***$ kubectl describe pod db***

***$ kubectl get events***

**Exercise**: Examine **/var/lib/pgdata** on the systems to check if the directory is been created and if the data is present.

## Creating Multi Container Pods

file: multi\_container\_pod.yml

apiVersion: v1

kind: Pod

metadata:

name: web

labels:

tier: front

app: nginx

role: ui

spec:

containers:

- name: nginx

image: nginx:stable-alpine

ports:

- containerPort: 80

protocol: TCP

volumeMounts:

- name: data

mountPath: /var/www/html-sample-app

- name: sync

image: gvenkat/sync:v1

volumeMounts:

- name: data

mountPath: /var/www/app

volumes:

- name: data

emptyDir: {}

To create this pod

***$ kubectl apply -f K8S-RESOURCES/k8s-code/pods/multi\_container\_pod.yaml***

Check Status

***$ kubectl get pods***

NAME READY STATUS RESTARTS AGE

web 2/2 Running 0 6m

***$ kubectl exec -it web sh -c nginx***

**$ kubectl exec -it web sh -c sync**

Observe what is common and what is isolated in two containers running inside the same pod using the following commands,

shared

*$ hostname*

*$ ifconfig*

isolated

*$ cat /etc/issue*

*$ ps aux*

*$ df -h*

## Exercise

Create a pod definition for redis and deploy.

**======================================================**

# Making application high available with Replication Controllers

If you are not running a monitoring screen, start it in a new terminal with the following command.

***$ watch -n 1 kubectl get pod,deploy,rs,svc***

### Setting up a Namespace

Check current config

***$ kubectl config view***

You could also examine the current configs in file **cat ~/.kube/config**

## Creating a namespace

Namespaces offers separation of resources running on the same physical infrastructure into virtual clusters. It is typically useful in mid to large scale environments with multiple projects, teams and need separate scopes. It could also be useful to map to your workflow stages e.g. dev, stage, prod.

Lets create a namespace called **instavote**

***$ cd K8S-RESOURCES/k8s-code/projects/instavote***

***$ cat instavote-ns.yaml***

[output]

kind: Namespace

apiVersion: v1

metadata:

name: instavote

Lets create a namespace

***$ kubectl get ns***

***$ kubectl apply -f instavote-ns.yaml***

***$ kubectl get ns***

And switch to it

***$ kubectl config --help***

***$ kubectl config get-contexts***

***$ kubectl config current-context***

***$ kubectl config set-context $(kubectl config current-context) --namespace=instavote***

***$ kubectl config view***

***$ kubectl config get-contexts***

**Exercise**: Go back to the monitoring screen and observe what happens after switching the namespace.

To understand how ReplicaSets works with the selectors lets launch a pod in the new namespace with existing specs.

***$ cd k8s-code/pods***

***$ kubectl apply -f vote-pod.yaml***

***$ kubectl get pods***

***$ cd ../projects/instavote/dev/***

Lets now write the spec for the Replica Set. This is going to mainly contain,

* replicas
* selector
* template (pod spec )
* minReadySeconds

file: vote-rs.yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: vote

spec:

replicas: 5

minReadySeconds: 20

selector:

matchLabels:

role: vote

matchExpressions:

- {key: version, operator: In, values: [v1, v2, v3]}

template:

Lets now add the metadata and spec from pod spec defined in vote-pod.yaml. And with that, the Replica Set Spec changes to

file: vote-rs.yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: vote

spec:

replicas: 5

minReadySeconds: 20

selector:

matchLabels:

role: vote

matchExpressions:

- {key: version, operator: In, values: [v1, v2, v3]}

template:

metadata:

name: vote

labels:

app: python

role: vote

version: v1

spec:

containers:

- name: app

image: gvenkat/vote:v1

ports:

- containerPort: 80

protocol: TCP

### Replica Sets in Action

***$ kubectl apply -f vote-rs.yaml --dry-run***

***$ kubectl apply -f vote-rs.yaml***

***$ kubectl get rs***

***$ kubectl describe rs vote***

***$ kubectl get pods***

**Exercise** :

* Switch to monitoring screen, observe how many replicas were created and why
* Compare selectors and labels of the pods created with and without replica sets

***$ kubectl get pods***

***$ kubectl get pods --show-labels***

### Exercise: Deploying new version of the application

kubectl edit rs/vote

Update the version of the image from **gvenkat/vote:v1** to **gvenkat/vote:v2**

Save the file. Observe if application got updated. Note what do you observe. Do you see the new version deployed ??

### Exercise: Self Healing Replica Sets

List the pods and kill some of those, see what replica set does.

***$ kubectl get pods***

***$ kubectl delete pods vote-xxxx vote-yyyy***

where replace xxxx and yyyy with actual pod ids.

Questions:

* Did replica set replaced the pods ?
* Which version of the application is running now ?

Lets now delete the pod created independent of replica set.

***$ kubectl get pods***

***$ kubectl delete pods vote***

Observe what happens. \* Does replica set take any action after deleting the pod created outside of its spec ? Why?

**==========================================================**

# Exposing Application with a Service

Types of Services:

* ClusterIP
* NodePort
* LoadBalancer
* ExternalName

***$ kubectl get pods***

***$ kubectl get svc***

Sample Output:

NAME READY STATUS RESTARTS AGE

voting-appp-j52x 1/1 Running 0 2m

voting-appp-pr2xz 1/1 Running 0 m

voting-appp-qpxbm 1/1 Running 0 5m

### Setting up monitoring

If you are not running a monitoring screen, start it in a new terminal with the following command.

***$ watch -n 1 kubectl get pod,deploy,rs,svc***

## Writing Service Spec

Lets start writing the meta information for service.

Filename: vote-svc.yaml

---

apiVersion: v1

kind: Service

metadata:

name: vote

labels:

role: vote

spec:

And then add the spec to it. Refer to Service (v1 core) api at this page https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.10/

---

apiVersion: v1

kind: Service

metadata:

name: vote

labels:

role: vote

spec:

selector:

role: vote

ports:

- port: 80

targetPort: 80

type: LoadBalancer

Save the file.

Now to create a service:

***$ kubectl apply -f vote-svc.yaml --dry-run***

***$ kubectl apply -f vote-svc.yaml***

***$ kubectl get svc***

Now to check which port the pod is connected

***$ kubectl describe service vote***

Check for the Load Balancer URL here

Go to browser and check with LB URL

Sample output will be:



## Internal Service Discovery

* Visit the vote app from browser
* Attemp to vote by clicking on one of the options

observe what happens. Does it go through?

Debugging,

***$ kubectl get pod***

***$ kubectl exec vote-xxxx ping redis***

[replace xxxx with the actual pod id of one of the vote pods ]

keep the above command on a watch. You should create a new terminal to run the watch command.

e.g.

***$ watch kubectl exec vote-kvc7j ping redis***

where, vote-kvc7j is one of the vote pods that I am running. Replace this with the actual pod id.

Now create **redis** service

***$ kubectl apply -f redis-svc.yaml***

***$ kubectl get svc***

***$ kubectl describe svc redis***

Watch the ping and observe if its able to resolve **redis** by hostname and its pointing to an IP address.

e.g.

PING redis (10.102.77.6): 56 data bytes

where **10.102.77.6** is the ClusterIP assigned to the service.

What happened here?

* Service **redis** was created with a ClusterIP e.g. 10.102.77.6
* A DNS entry was created for this service. The fqdn of the service is **redis.instavote.svc.cluster.local** and it takes the form of my-svc.my-namespace.svc.cluster.local
* Each pod points to internal DNS server running in the cluster. You could see the details of this by running the following commands

***$ kubectl exec vote-xxxx cat /etc/resolv.conf***

[replace vote-xxxx with actual pod id]

[sample output]

nameserver 10.96.0.10

search instavote.svc.cluster.local svc.cluster.local cluster.local

options ndots:5

where **10.96.0.10** is the ClusterIP assigned to the DNS service. You could co relate that with,

***$ kubectl get svc -n kube-system***

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kube-dns ClusterIP 10.96.0.10 <none> 53/UDP,53/TCP h

where, **10.96.0.10** is the ClusterIP assigned to **kube-dns** and matches the configuration in **/etc/resolv.conf** above.

##### Creating Endpoints for Redis

Service is been created, but you still need to launch the actual pods running **redis** application.

Create the endpoints now,

***$ kubectl apply -f redis-deploy.yaml***

***$ kubectl describe svc redis***

[sample output]

Name: redis

Namespace: instavote

Labels: role=redis

tier=back

Annotations: kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"v1","kind":"Se

rvice","metadata":{"annotations":{},"labels":{"role":"redis","tier":"back"},"name":"redis","names

pace":"instavote"},"spec"...

Selector: app=redis

Type: ClusterIP

IP: 10.97.182.252

Port: <unset> 6379/TCP

TargetPort: 6379/TCP

Endpoints: 10.38.0.7:6379,10.40.0.6:6379

Session Affinity: None

Events: <none>

Again, visit the vote app from browser, attempt to register your vote and observe what happens now.

**=======================================================**

**Creating a Deployment**

A Deployment is a higher level abstraction which sits on top of replica sets and allows you to manage the way applications are deployed, rolled back at a controlled rate.

Deployment has mainly two responsibilities,

* Provide Fault Tolerance: Maintain the number of replicas for a type of service/app. Schedule/delete pods to meet the desired count.
* Update Strategy: Define a release strategy and update the pods accordingly.

File: vote-deploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: vote

spec:

strategy:

type: RollingUpdate

rollingUpdate:

maxSurge: 2

maxUnavailable: 1

revisionHistoryLimit: 4

paused: false

replicas: 8

minReadySeconds: 20

selector:

matchLabels:

role: vote

matchExpressions:

- {key: version, operator: In, values: [v1, v2, v3]}

template:

metadata:

name: vote

labels:

app: python

role: vote

version: v2

spec:

containers:

- name: app

image: gvenkat/vote:v2

ports:

- containerPort: 80

protocol: TCP

This time, start monitoring with --show-labels options added.

***$ watch -n 1 kubectl get pod,deploy,rs,svc --show-labels***

Lets create the Deployment. Do monitor the labels of the pod while applying this.

***$ kubectl apply -f vote-deploy.yaml***

Observe the chances to pod labels, specifically the **pod-template-hash**.

Now that the deployment is created. To validate,

***$ kubectl get deployment***

***$ kubectl get rs --show-labels***

***$ kubectl get deploy,pods,rs***

**Scaling a deployment**

To scale a deployment in Kubernetes:

***$ kubectl scale deployment/vote --replicas=12***

***$ kubectl rollout status deployment/vote***

Sample output:

Waiting for rollout to finish: 5 of 12 updated replicas are available...

Waiting for rollout to finish: 6 of 12 updated replicas are available...

deployment "vote" successfully rolled out

You could also update the deployment by editing it.

***$ kubectl edit deploy/vote***

[change replicas to 15 from the editor, save and observe]

**Rolling Updates in Action**

Now, update the deployment spec to apply

file: vote-deploy.yaml

spec:

...

replicas: 15

...

labels:

app: python

role: vote

version: v3

...

template:

spec:

containers:

- name: app

image: gvenkat/vote:v3

apply

***$ kubectl apply -f vote-deploy.yaml***

***$ kubectl rollout status deployment/vote***

Observe rollout status and monitoring screen.

***$ kubectl rollout history deploy/vote***

***$ kubectl rollout history deploy/vote --revision=1***

**Undo and Rollback**

file: vote-deploy.yaml

spec:

containers:

- name: app

image: gvenkat/vote:rgjerdf

apply

***$ kubectl apply -f vote-deploy.yaml***

***$ kubectl rollout status***

***$ kubectl rollout history deploy/vote***

***$ kubectl rollout history deploy/vote --revision=xx***

where replace xxx with revisions

Find out the previous revision with sane configs.

To undo to a sane version (for example revision 3)

***$ kubectl rollout undo deploy/vote --to-revision=3***

**=====================================================**

# Mini Project: Deploying Multi Tier Application Stack

In this project , you would write definitions for deploying the vote application stack with all components/tiers which include,

* vote ui
* redis
* worker
* db
* results ui

## Tasks

* Create deployments for all applications
* Define services for each tier applicable
* Launch/apply the definitions

Following table depicts the state of readiness of the above services.

| **App** | **Deployment** | **Service** |
| --- | --- | --- |
| vote | ready | ready |
| redis | ready | ready |
| worker | TODO | n/a |
| db | ready | ready |
| results | TODO | TODO |

**Specs:**

* worker
  + image: gvenkat/worker:latest
* results
  + image: gvenkat/vote-result
  + port: 80
  + service type: NodePort

### Deploying the sample application

To create deploy the sample applications,

***$ kubectl create -f projects/instavote/dev***

Sample output is like:

deployment "db" created

service "db" created

deployment "redis" created

service "redis" created

deployment "vote" created

service "vote" created

deployment "worker" created

deployment "results" created

service "results" created

#### To Validate:

***$ kubectl get svc -n instavote***



This will load the page where you can vote.

To check the result:

***$ kubectl get service result***



This is the page where you should see the results for the vote application stack.

**=========================================================**