Kubernetes

Initial page

Introducing Pods

Introducing Pods

- At the heart of Kubernetes
- Most important concept of Kubernetes
- Everything else either manages, exposes or is used by the pods
- Represents the basic building blocks
- Co-located group of containers
- Common to have one container per pod but not a requirement

Why cant we use the containers directly?

- Containers are designed to run only a single process per container
- If multiple process are running inside a single container it is our responsibility to manage all the process.
- Difficult to manage the logs since all the process with dump the logs on the same container
- Norm of working with kubernetes is to use single process per container
- Containers in the pod share the same network and hostnames
- Containers in the pod run under the same IPC namespace and communicate via the IPC
- When multiple containers are running ,Pod acts as a higher level construct, to manage all the containers together as a single unit. This is the responsibility of a pod
- Pod of containers runs a closely group of containers and run them as if they were running in a isolated environment.
- Best of both the worlds, You get the coherent of related group of process and yet treat individual process as isolated units.
- Kubernetes leverages Docker to run pod
- Pods are other resources can be created by posting the json or YAML file to K8s REST API endpoint
- Contains
    - Metadata
    - Specification

- Status

Creating a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: hn-service-pod
spec:
  containers:
  - image: classpathio/hn-service
    name: hn-service-container
    ports:
    - containerPort: 8011
      protocol: TCP
```

To get to know the options for creating the pod definition, we can use kubectl explain command

```
kubectl explain pods
```

```
kubectl explain pod.spec
```

To create a pod from the pod definition – Generic command to create any resource from yaml/json file

```
kubectl create -f pod-definition.yaml
```

To get the resource description from the resource use the below command

```
kubectl get po hn-service-pod -o yaml
kubectl get po hn-service-pod -o json
```

Output:

```
$ kubectl get pods

NAME              READY   STATUS    RESTARTS   AGE
hn-service-pod    1/1     Running   0          6m15s
nginx-pod         1/1     Running   0          11m
```

Viewing logs

- Container applications usually log to the standard output and error streams.

Command to check the logs

```
docker logs <container id>
```

Checking the logs with Kubernetes – Used when the pod contains a single container

```
kubectl logs hn-service-pod
```

With multiple containers in a single pod

```
kubectl logs hn-service-pod -c hn-service-container
```

Port forwarding to send requests to the pod from the local machine

```
kubectl port-forward hn-service-pod 8111:8111

Forwarding from 127.0.0.1:8111 -> 8111
Forwarding from [::1]:8111 -> 8111
```

Verifying the pod's response

```
curl localhost:8111
{ "hostname":"hn-service-pod" }
```

Organizing pods with labels

- Categorizing pods into subsets
- To operate on every pod belonging to certain group with a single action instead of performing the action on individual pod explicitly
- Done through labels

Introducing labels :

- Organizing resources – including pods
- Key-value pairs – attaching to resources
- Selecting resources using label-selectors
- Resources can have more than one labels
- Keys should be unique
- Can be created at creation time but can also be modified or add additional labels later without recreating the resource.

Specifying the label during creation time

```
apiVersion: v1
kind: Pod
metadata:
  name: hn-service-pod-label
  labels:
    env: dev
    creation_method: manual
spec:
  containers:
  - image: classpathio/hn-service
    name: hn-service-container
    ports:
    - containerPort: 8011
      protocol: TCP
```

```
kubectl create -f hn-service-pod-label.yaml

kubectl get po --show-labels
```

Listing by label names

```
kubectl get po -L creation_method
NAME                   READY   STATUS    RESTARTS   AGE   CREATION_METHOD
hn-service-pod         1/1     Running   0          51m
hn-service-pod-label   1/1     Running   0          12m   manual
nginx-pod              1/1     Running   0          56m
```

```
kubectl get po -L creation_method,env
NAME                   READY   STATUS    RESTARTS   AGE   CREATION_METHOD
hn-service-pod         1/1     Running   0          51m
hn-service-pod-label   1/1     Running   0          12m   manual
nginx-pod              1/1     Running   0          56m
```

Adding labels after the pod creation

```
kubectl label po hn-service-pod creation_method=manual notes=test
```

Overriding existing pod label value

```
kubectl label po hn-service-pod env=prod --overwrite
```

Check if the label value is updated

```
kubectl get po -L env
NAME                   READY   STATUS    RESTARTS   AGE   ENV
hn-service-pod         1/1     Running   0          66m   prod
hn-service-pod-label   1/1     Running   0          27m   dev
nginx-pod              1/1     Running   0          71m
```

Label selectors

- We can select the subset of pods and perform action on them.
- They act like filter criteria to shortlist the pods
- Label selectors can be used to select the resources based on
    - Contains or not a label with a key
    - Contains a label with a key and a value
    - Contains the label with a key and not matching value

```
kubectl get po -l creation_method=manual
```

```
kubectl get po -l env
```

```
kubectl get po -l '!env'
```

creation_method!=manual

env in (prod,devel)

env notin (prod,devel)

Use comma seperated label to apply multiple filters

Labels are at the worker node level

To categorize the node. Ex: GPU intensive, SSD's or spinning disks etc

```
kubectl get nodes
NAME                                            STATUS    ROLES     AGE     VE
ip-172-20-43-227.ap-south-1.compute.internal    Ready     master    115m    v1
ip-172-20-49-73.ap-south-1.compute.internal     Ready     node      114m    v1
ip-172-20-70-31.ap-south-1.compute.internal     Ready     node      114m    v1
ip-172-20-73-76.ap-south-1.compute.internal     Ready     node      114m    v1
```

```
kubectl label node ip-172-20-49-73.ap-south-1.comput e.internal gpu=true
node/ip-172-20-49-73.ap-south-1.compute.internal labeled
```

```
kubectl get nodes -L gpu
NAME                                            STATUS    ROLES     AGE     VE
ip-172-20-43-227.ap-south-1.compute.internal    Ready     master    120m    v1
ip-172-20-49-73.ap-south-1.compute.internal     Ready     node      118m    v1
ip-172-20-70-31.ap-south-1.compute.internal     Ready     node      118m    v1
ip-172-20-73-76.ap-south-1.compute.internal     Ready     node      118m    v1
```

Scheduling pods to a specific node label

```
apiVersion: v1
kind: Pod
metadata:
  name: hn-service-pod-label-gpu
  labels:
    env: dev
    creation_method: manual
spec:
  nodeSelector:
    gpu: "true"
  containers:
  - image: classpathio/hn-service
    name: hn-service-container
    ports:
    - containerPort: 8011
      protocol: TCP
```

Annotating pods

- Also key value pair like labels
- Aren't meant to hold information
- Cannot be selected unlike label selector
- Primary meant to be used by tools
- Some annotations are automatically added by the tools
- Used to introduce new features to the specification in a graceful manner
- Once the new features are agreed upon, the old ones are depricated

```
kubectl get po
NAME                       READY   STATUS    RESTARTS   AGE
hn-service-pod             1/1     Running   0          106m
hn-service-pod-label       1/1     Running   0          67m
hn-service-pod-label-gpu   1/1     Running   0          9m15s
nginx-pod                  1/1     Running   0          111m
```

Just like label's annotations can be added during creation time or added later

```
kubectl annotate po hn-service-pod classpathio/type= "backend API"
pod/hn-service-pod annotated
```

Run the `describe` command to see the annotation

```
kubectl describe po hn-service-pod
Name:         hn-service-pod
Namespace:    default
Priority:     0
Node:         ip-172-20-70-31.ap-south-1.compute.internal/172.20.70.31
Start Time:   Fri, 27 Mar 2020 06:34:34 +0000
Labels:       creation_method=manual
              env=prod
              notes=test
Annotations:  classpathio/type: backend API
              kubernetes.io/limit-ranger: LimitRanger plugin set: cpu requ
Status:       Running
IP:           100.96.1.3
IPs:
  IP:  100.96.1.3
```

Namespaces to group resources

- Grouping of resources can be done using `label` s
- Labels can overlap since multiple resources can container many labels
- Namespaces are used to split objects into separate non-overlapping groups
- Can be used to split resources based on tenant, environments etc.
- Resource names should be unique withing a namespace
- Works like packages in Java

Operations on a namespace

List all namespace

```
kubectl get ns
NAME              STATUS   AGE
default           Active   166m
kube-node-lease   Active   166m
kube-public       Active   166m
kube-system       Active   166m
```

Fetch the resources specific to a namespace

```
kubectl get po --namespace kube-system
NAME                                                                   REA
   RESTARTS   AGE
dns-controller-5769c5f8b6-nhznh                                        1/1
   0          172m
etcd-manager-events-ip-172-20-43-227.ap-south-1.compute.internal       1/1
   0          172m
etcd-manager-main-ip-172-20-43-227.ap-south-1.compute.internal         1/1
   0          172m
kops-controller-p4b8n                                                  1/1
   0          172m
[ec2-user@ip-172-31-36-76 pods]$ kubectl get po -n  kube-system
NAME                                                                   REA
   RESTARTS   AGE
dns-controller-5769c5f8b6-nhznh                                        1/1
   0          173m

```

Use -n instead of –namespace

Creating a namespace

```
apiVersion: v1
kind: Namespace

metadata:
  name: classpath-dev-namespace
```

```
kubectl create -f classpath-dev-namespace.yaml
namespace/classpath-dev-namespace created
```

Alternate way instead of creating from YAML file

```
kubectl create namespace classpath-qa-namespace
namespace/classpath-qa-namespace created
```

Creating resources inside the Namespacespace

```
kubectl create -f hn-service-pod-label.yaml -n "classpath-dev-namespace"
pod/hn-service-pod-label created
```

- When performing operationg like listing, deleting we need to specify the namespace using the `-n` or `--namespace`
- If not specified, the default namespace will be used from the configured context
- The context can be configured to set the default namespace using the `kubectl config` commands

```
kubectl config set-context classpath-dev-namespace --namespace
```

Stopping Pods

Deleting the pod by name

```
kubectl delete po nginx-pod
pod "nginx-pod" deleted
```

- First the K8s sends the `SIGTERM` signal to the process and waits for certain amount of time (30 seconds by default)
- If the process does not terminate, `SIGKILL` signal is issues to kill the process.
- Make sure that the process is shutdown gracefully (shutdown using the `SIGTERM` command)

Deleting multiple pods

```
kubectl delete po nginx-pod, hn-service-pod-label
```

```
pod "hn-service-pod-label" deleted
pod "nginx-pod" deleted
```

Deleting pods using the label selector

```
kubectl delete po -l env=dev
pod "hn-service-pod" deleted
```

Deleting all the resources including pods by deleting the namespace

```
kubectl delete ns classpath-dev-namespace
namespace "classpath-dev-namespace" deleted
```

Deleting the pods by retaining the namespace

```
kubectl delete po --all
pod "hn-service-pod-label-gpu" deleted
```

> Note: creating a pod using the `kubectl run` creates a ReplicationController and hence, deleting the pod will not work as the `RC` will spin up a new pod.

Deleting the pods, rc and the namespace

```
kubectl delete all --all
    pod "hn-service-pod-label-gpu" deleted
    replicationcontroller "test" deleted
    service "test" deleted
    service "test-http" deleted
```

Additional Notes

- First all represents all resources types (pods, rc)
- Second –all represents deleting all resources instead by name
- The command also list all the resources that were deleted
- The service will also get deleted but will be automatically created after sometime.

Replication Controller

Livess Probe

- Pod is scheduled to a Node
- `kubelet` on the node will run its container
- Keep the containers running as long as the pod exists
- In case of process crash, the kubelet restarts the container
- The process should not restart in case of unhealty pod
- Monitor the health from outside the container and not depend on the app
- Checking container liveness using `Liveness Probe`
  - HTTP GET probe (2xx and 3xx status codes are considered successfull)
  - TCP Socket (connection successfully established is considered successfull)
  - Exec (Exit status code of 0 is considered successfull)

Creating a Liveness Probe

```
apiVersion: v1
kind: Pod
metadata:
  name: hn-service-unhealthy
spec:
  containers:
  - image: classpathio/hn-service-unhealthy
    name: hn-service-unhealthy
    livenessProbe:
      httpGet:
        path: /
        port: 8111
```

```
apiVersion: v1
kind: Pod
metadata:
  name: hn-service-unhealthy
spec:
  containers:
  - image: classpathio/hn-service-unhealthy
    name: hn-service-unhealthy
    livenessProbe:
      httpGet:
        path: /
        port: 8111
```

```
kubectl create -f liveness-probe. yaml
pod/hn-unhealthy-service created
```

```
kubectl get po
NAME                   READY   STATUS    RESTARTS   AGE
hn-unhealthy-service   1/1     Running   0          57s
```

```
kubectl get po
NAME                   READY   STATUS    RESTARTS   AGE
hn-unhealthy-service   1/1     Running   0          57s
```

- `Restarts` column displays the number of times the pod restarted

Obtaining the logs

```
$kubectl logs hn-unhealthy-service  --previous

Inside the getHostname method of Controller
Number of requests  5
 Received request from 100.96.1.1You have hit the Server
Your current IP address : hn-unhealthy-service/100.96.1.6
Your current Hostname : hn-unhealthy-service
 Inside the getHostname method of Controller
Number of requests  6
 Inside the getHostname method of Controller
Number of requests  6
 Inside the getHostname method of Controller
Number of requests  6
2020-03-27 10:22:02.860  INFO 1 --- [extShutdownHook] o.s.s.concurrent.Thr
```

Configuring additional properties

```yaml
livenessProbe:
   httpGet:
     path: /
     port: 8111
   initialDelaySeconds: 15
```

Points to consider:

- The Liveness probe should be light
- No business logic in the liveness probe
- Definition of the healthy service should be defined here
- No retry loops as it is handled by the K8s itself
- Should not take too long to complete
- Should not be CPU intensive

Replication Controller

- Replication controller is a K8s resource
- Ensures that the pods are always running.
- If a running pod goes missing, the replication controller crates a replacement pod across pods.

Operations of Replication controller

- Continuously monitors the running pods and ensures that the actual number of running pods match the desired number.
- If two many pods are running, the Replication controller, removes the excess pods.

> How there can be more pods than the actual definition?
>
> **Answer:**
>
> - Pods created manually

> - Changing the pods definition
> - Changing the desired number of pods in the template

Operations of a Replication Controller

- ReplicationController work on a set of pods which matches the label
- Replication controller's job is to ensure the actual running pods are matching the label-selector.
- Reconcile to match the desired count
- ReplicationController has three essential parts
  - Label selector – matching pods
  - Replica count – the desired count
  - Pod template – the template of pod (definition)
- All the above entries can be modified at any time, but only the replica count affects the current pods
- Changing the label selectors does not affect the existing pods
- Existing pods fall out of the scope
- Controller stops caring about the pods which are out of scope

Benefits of Replication Controller

- Ensures that the desired number of pods are always running, even in case of pod failure
- Sustains the node failure
- Enables horizontal scaling of pods
  - both manual and automatic

Creating a Replication controller

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: hn-service
spec:
  replicas: 3
  selector:
    app: hn-service-api
  template:
     metadata:
       labels:
         app: hn-service-api
     spec:
       containers:
       - name:  hn-service-container
         image: classpathio/hn-service
         ports:
          - containerPort: 8111
            protocol: TCP
```

Caution:

- If the selector does not match the pod selector, the API server validates and will not accept, if it is misconfigured
- K8s will automatically extract the label selector, if it is not part of the replication controller
- Use this approach and do not specify the label selector in the Replication controller

Creating a replication controller

```
$ kubectl create -f hn-service-rc.yaml
```

To verify the rc in action

```
$ kubectl get pods
```

Delete a pod to see if the RC creates a new pod

```
kubectl delete pod hn-service-5sz5t
```

```
kubectl get pods
```

Getting RC information

```
kubectl get rc
```

Explain: To get Additional information about the RC

```
kubectl describe rc hn-service

Name:           hn-service
Namespace:      default
Selector:       app=hn-service-api
Labels:         app=hn-service-api
Annotations:    <none>
Replicas:       3 current / 3 desired
Pods Status:    3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=hn-service-api
  Containers:
   hn-service-container:
    Image:        classpathio/hn-service
    Port:         8111/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age    From                    Message
  ----    ------            ----   ----                    -------
  Normal  SuccessfulCreate  7m27s  replication-controller  Created pod: hn
  Normal  SuccessfulCreate  7m27s  replication-controller  Created pod: hn
  Normal  SuccessfulCreate  7m27s  replication-controller  Created pod: hn
  Normal  SuccessfulCreate  4m8s   replication-controller  Created pod: hn
  Normal  SuccessfulCreate  3m12s  replication-controller  Created pod: hn
```

> Events should also be displayed at the bottom after running the describe command

Behind the scenes

- RC is notified about the deletion of pod event
- API server allows for clients to watch for changes/events
- The notification triggers the RC to verify the number of pods against the desired count to take appropriate action

Responding to node failure

List the pods with –o wide option to get the address of the node

```
kubectl get node -o wide
```

```
gcloud compute ssh gke-kubia-default-pool-b46381f1-zwko

Enter passphrase for key '/home/luksa/.ssh/google_compute_engine':

Welcome to Kubernetes v1.6.4!

...

sudo ifconfig eth0 down
```

```
kubectl get node
```

Moving the pods' in and out of the Replication Controller

- Pods created using the Replication Controllers are not tied up with the RC in any way

- Pod contain a `metadata.ownerReferences` field which we can use to idendify the replication controller the pod belongs to
- If we change the label of the pod, then it is no more managed by the RC and is as good as creating the pod manually
- The RC will spin a new pod, in that case since the pod definition changed
- Adding another label does not affect the RC as long as the RC matches the pod labels

```
kubectl label pod kubia-dmdck type=special

pod "kubia-dmdck" labeled
```

```
kubectl get pods --show-labels
NAME                 READY   STATUS    RESTARTS   AGE   LABELS
hn-service-4nc6d     1/1     Running   0          34m   app=hn-service-api
hn-service-755n6     1/1     Running   0          37m   app=hn-service-api
hn-service-g6zgl     1/1     Running   0          33m   app=hn-service-api
```

> Example of overriding the existing label value to drift from the rc definition which will notify the RC and RC will take action – In this case, creating a new pod from the template

```
kubectl label pod hn-service-4nc6d app=non-existent --overwrite
pod/hn-service-4nc6d labeled
```

```
kubectl get pods -L app
NAME                 READY   STATUS    RESTARTS   AGE   APP
hn-service-4nc6d     1/1     Running   0          36m   non-existent
hn-service-755n6     1/1     Running   0          39m   hn-service-api
hn-service-g6zgl     1/1     Running   0          35m   hn-service-api
hn-service-sl7s9     1/1     Running   0          23s   hn-service-api
```

Can be used to remove a unhealthy pod from the rc scope and deleted later

Editing the RC

```
kubectl edit rc hn-service
```

> This is open the YAML inside a editor which you can open and save the file
>
> After editing the RC template with the new pod definition, delete the pods so that the RC will create all the pods with the new definition – Can be used for updates

Changing the default editor

```
export KUBE_EDITOR="/usr/bin/nano"
```

Scaling Pods

- Scaling the pods using the RC – for experimental purpose only since it is not versioned

```
kubectl scale rc hn-service--replicas=10
```

Editing the file using declarative approach – `Can be versioned`

```
kubectl edit rc hn-service
```

Verify the RC definition after the changes

```
kubectl get rc
NAME          DESIRED    CURRENT    READY    AGE
hn-service    5          5          5        51m
```

Scaling down – Modifies the `spec.replicas` in the RC definition like the edit we did in the RC definition

```
kubectl scale rc hn-service --replicas=3
```

Deleting the Replication Controller

- Deleting an RC
  - Delete the RC but keep the existing pods and do not delete them – Default is delete the pods also along with the RC

```
kubectl delete rc hn-service --cascade=false

replicationcontroller "hn-service" deleted
```

> ⓘ Note: You can manage the pods again by creating a RC to match the pods label again

ReplicaSets

ReplicaSets

- ReplicaSets replaces the RC
- Should be used for all new work
- Use ReplicaSets instead of RC

ReplicaSets vs RC

- Works in the same fashion as `RC`
- More expressive pod selectors
- RC label selector only allows for matching pods including a certain label (not values of the label key/value)
- RS allows for selector allows matching pods that lack a label or pods which include a certain label key, regardless of the value
- Example:
  - RC cannot match pods with label env=production and env=dev at the same time
  - Match either pods with env=production or env-=dev at a single time
  - RS can match both of them at the same time and treat them as a single group
  - RC can also select the pods with only key regardless of the value. Example env=*

Defining a RS

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hn-service-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hn-service
  template:
    metadata:
      labels:
        app: hn-service
    spec:
      containers:
```

```
      - name: hn-service-container
        image: classpathio/hn-service
```

- API is in a different namespace
- Only different is the selector compared to a RC
- Instead of listing labels, the `RS` has `selector.matches` property instead of under `selector` property in RC

Creating a ReplicaSet

```
kubectl get rs
NAME            DESIRED   CURRENT   READY   AGE
hn-service-rs   3         3         3       2m37s
```

Describe Replica Set

```
Name:         hn-service-rs
Namespace:    default
Selector:     app=hn-service
Labels:       <none>
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=hn-service
  Containers:
   hn-service-container:
    Image:        classpathio/hn-service
    Port:         <none>
```

RS – selector example

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hn-service-rs-match-expression
```

```
spec:
  replicas: 3
  selector:
    matchExpressions:
      - key: app
        operator: In
        values:
          - hn-service-api
  template:
    metadata:
      labels:
        app: hn-service-api
    spec:
      containers:
      - name: hn-service-container
        image: classpathio/hn-service
```

> ⓘ Note:
>
> If multiple expressions are put, all of them should evaluate to true

Listing ReplicaSets

```
kubectl get rs
NAME                               DESIRED    CURRENT    READY    AGE
hn-service-rs-match-expression     3          3          3        3m13s
```

Listing Pods

```
kubectl get po
NAME                                     READY    STATUS     RESTARTS    AGE
hn-service-rs-match-expression-5mfb8     1/1      Running    0           2m33s
hn-service-rs-match-expression-ppnvl     1/1      Running    0           2m33s
hn-service-rs-match-expression-r6x2b     1/1      Running    0           2m33s
```

Deleting RS

```
kubectl delete rs hn-service-rs-match-expres sion
replicaset.apps "hn-service-rs-match-expression" deleted
```

DaemonSet

- Similar to `RC` and `RS`
- Pods are deployed to all the nodes
- Replicas does not matter since it is at the node level
- Selecting to deploy on a subset of nodes can be donesusing `node-Selector` property of the pod definition.
- Meant for run the system level services

Example

- Defining the `DaemonSet` yaml definition

```
apiVersion: apps/v1beta2
kind: DaemonSet
metadata:
name: ssd-monitor
spec:
selector:
  matchLabels:
    app: ssd-monitor
template:
  metadata:
    labels:
      app: ssd-monitor
  spec:
    nodeSelector:
      disk: ssd
    containers:
    - name: main
      image: luksa/ssd-monitor
```

Creating the Daemonset

```
kubectl create -f hn-daemonset.yaml
daemonset.apps/healthchecker created
```

List the DamesonSet

```
kubectl get ds
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
healthchecker   0         0         0       0            0           disk=
```

List all the Nodes

```
kubectl get nodes
```

Add the label to the nodes

```
kubectl label node ip-172-20-84-241.ap-south-1.compute.internal disk=ssd
node "ip-172-20-84-241.ap-south-1.compute.internal" labeled
```

List all the pods

```
kubectl get po
NAME                 READY     STATUS    RESTARTS   AGE
ssd-monitor-hgxwq    1/1       Running   0          35s
```

Remove the label from the node

```
kubectl label node ip-172-20-84-241.ap-south -1.compute.internal disk=hdd
node/ip-172-20-84-241.ap-south-1.compute.internal labeled
```

List the Node with -L option

```
kubectl get nodes -L disk
NAME                                           STATUS   ROLES    AGE   VER
ip-172-20-44-6.ap-south-1.compute.internal     Ready    node     48m   v1.
ip-172-20-52-93.ap-south-1.compute.internal    Ready    master   50m   v1.
ip-172-20-60-251.ap-south-1.compute.internal   Ready    node     48m   v1.
ip-172-20-84-241.ap-south-1.compute.internal   Ready    node     48m   v1.
```

Again list the pods

```
kubectl get po
No resources found in default namespace.
```

Running a single completable task

- In case of single completable task after which the process terminates
- Does not restart once the process completes successfully
- In case of job failure, we can configure to either `restart` or `exit`

Job Resource

Creating a JobResource

```
apiVersion: batch/v1
kind: Job
metadata:
  name: batch-job
spec:
 template:
   metadata:
     labels:
       app: batch-job
   spec:
     restartPolicy: OnFailure
     containers:
       - name: job
         image: classpathio/batch-job
```

Create a Job resource from the template

```
kubectl create -f hn-batch-job.yaml
job.batch/batch-job created
```

Test the JobResource

```
kubectl get job
NAME        COMPLETIONS   DURATION   AGE
batch-job   1/1           9s         10s
```

```
kubectl get po
NAME              READY     STATUS     RESTARTS   AGE
batch-job-28qf4   1/1       Running    0          4s
```

After the batch job is complete

```
kubectl get po
NAME              READY   STATUS      RESTARTS   AGE
batch-job-k4trj   0/1     Completed   0          110s
```

Check the logs

```
kubectl logs batch-job-k4trj
```

```
Starting the Batch Job - Tue Mar 31 2020 05:54:11 GMT+0000 (Coordinated Un
Completed the Job - Tue Mar 31 2020 05:54:16 GMT+0000 (Coordinated Univers
```

Delete the Job

```
kubectl delete job batch-job
job.batch "batch-job" deleted
```

Running multiple instance of the Job

Job definition

```
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-batch-job
spec:
 completions: 5
 template:
   metadata:
     labels:
       app: batch-job
   spec:
     restartPolicy: OnFailure
     containers:
       - name: job
         image: classpathio/batch-job
```

Creating the Job definition

```
kubectl create -f hn-multi-batch-job.yaml
job.batch/multi-batch-job created
```

Test the Job

```
kubectl get jobs
NAME              COMPLETIONS   DURATION   AGE
multi-batch-job   1/5           18s        18s
```

Verify the Pods

```
kubectl get po
NAME                    READY   STATUS      RESTARTS   AGE
multi-batch-job-cmcsn   0/1     Completed   0          91s
multi-batch-job-qxwsg   0/1     Completed   0          72s
multi-batch-job-rmvnb   0/1     Completed   0          100s
multi-batch-job-sjzlb   0/1     Completed   0          82s
multi-batch-job-zgtvh   0/1     Completed   0          109s
```

Deleting multi-batch-job resource

```
kubectl delete job multi-batch-job
job.batch "multi-batch-job" deleted
```

Running multiple instance of Job in parallel

```
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-batch-job
spec:
 completions: 5
 parallelism: 2
 template:
   metadata:
     labels:
       app: batch-job
   spec:
     restartPolicy: OnFailure
     containers:
       - name: job
         image: classpathio/batch-job
```

Create the job resource

```
kubectl create -f hn-multi-batch-job-paralle l.yaml
job.batch/multi-batch-job created
```

```
kubectl get jobs
NAME              COMPLETIONS   DURATION   AGE
multi-batch-job   0/5           7s         7s
```

Check the status of pods

```
kubectl get po
NAME                    READY   STATUS      RESTARTS   AGE
multi-batch-job-drb98   0/1     Completed   0          15s
multi-batch-job-hvb6m   1/1     Running     0          4s
multi-batch-job-m6txz   1/1     Running     0          6s
multi-batch-job-vrmnr   0/1     Completed   0          15s
multi-batch-job-vxr6j   0/1     Completed   0          15s
```

After some time, check the status of pods

```
kubectl get po
NAME                    READY   STATUS      RESTARTS   AGE
multi-batch-job-drb98   0/1     Completed   0          87s
multi-batch-job-hvb6m   0/1     Completed   0          76s
multi-batch-job-m6txz   0/1     Completed   0          78s
multi-batch-job-vrmnr   0/1     Completed   0          87s
multi-batch-job-vxr6j   0/1     Completed   0          87s
```

Delete the multi-batch-job resource

```
kubectl delete job multi-batch-job
job.batch "multi-batch-job" deleted
```

CronJob Resource

Creating CronJob Resource

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: batch-job-every-fifteen-minutes
spec:
  schedule: "0,15,30,45 * * * *"
  startingDeadlineSeconds: 15
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: periodic-batch-job
        spec:
          restartPolicy: OnFailure
          containers:
          - name: main
            image: luksa/batch-job
```

Creating a CronJob from the definition

```
kubectl create -f hn-cron-job.yaml
cronjob.batch/batch-job-every-fifteen-minutes created
```

Fetching the cronjobs

```
kubectl get cronjobs
```

```
NAME                                SCHEDULE             SUSPEND    ACTIVE
batch-job-every-fifteen-minutes     0,15,30,45 * * * *   False      0
    95s
```

Deleting cron job resource

```
kubectl delete cronjob batch-job-every-fifte en-minutes
cronjob.batch "batch-job-every-fifteen-minutes" deleted
```

Services

Services

- K8s resource
- Single point of entry to a group of pods
- Has a constant IP address and port
- Provides Load balancing capabilities and allow to add/remove pods to/from clusters
- Label selector is used to group pods belongining to the service

---

Creating a Service resource

Creating a Service using the YAML definition

```yaml
apiVersion: v1
kind: Service
metadata:
  name: hostname-service
spec:
  ports:
  - port: 80
    targetPort: 8111
  selector:
    app: hn-service
```

Creating the service resource

```
kubectl  create -f hn-service.yaml
service/hostname-service created
```

Checking the service

```
kubectl get svc
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
hostname-service   ClusterIP   100.69.124.66   <none>        80/TCP    72s
kubernetes         ClusterIP   100.64.0.1      <none>        443/TCP   18m
```

- Once the Service is created, the pods can be accessed via the cluster IP
- The clusterIP can be used by all the other pods in the cluster
- The clusterIP is visible only within the cluster

Different ways to access the service from within the cluster

- Create a Pod and then invoke the services from within the pod
- `ssh` into the `node` and use the `curl` command
- use the `kubectl exec` command to issue a `curl` request from withing a pod

Testing the service from a pod

- Run the `kubectl get pods` to fetch the pods
- Obtaing the cluster IP using the `kubetcl get sv` command
- Call the service from the target pod using

Create the Pods using the RS which has the label as hn-service

```
kubectl create -f hn-service-rs.yaml
replicaset.apps/hn-service-rs created
```

List all the Pods

```
kubectl get po -L app
NAME                  READY   STATUS    RESTARTS   AGE   APP
hn-service-rs-6h7rf   1/1     Running   0          63s   hn-service
hn-service-rs-9qrgq   1/1     Running   0          63s   hn-service
hn-service-rs-b9r88   1/1     Running   0          63s   hn-service
```

```
kubectl exec hn-service-rs-w2llk -- curl -s http://100.69.124.66
{ "hostname":"hn-service-rs-r5kzs" }

kubectl exec hn-service-rs-w2llk -- curl -s http://100.69.124.66
{ "hostname":"hn-service-rs-kvss9" }

kubectl exec hn-service-rs-w2llk -- curl -s http://100.69.124.66
{ "hostname":"hn-service-rs-w2llk" }
```

> Note: the `--` signals end of command. everything later should be executed inside the pod.

Configuring session affinity

If we want all the requests to go to a the same pod, then use the `sessionAffinity` to `ClientIP` (`None` is the default)

```
apiVersion: v1
kind: Service
metadata:
  name: hostname-service-affinity
spec:
  sessionAffinity: ClientIP
  ports:
  - port: 80
    targetPort: 8111
  selector:
    app: hn-service
```

Note: There is no Cookie based session affinity since K8s does not operate at HTTP level. Services deal with `TCP` and `UDP` packets and dont care about the payload that they carry.

Create a Service with Client affinity

```
kubectl create -f hn-service-client-affinity.yaml
```

List the services

```
kubectl get svc
NAME                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT
hostname-service             ClusterIP   100.69.124.66   <none>        80/T
hostname-service-affinity    ClusterIP   100.70.128.16   <none>        80/T
kubernetes                   ClusterIP   100.64.0.1      <none>        443/
```

Affinity will always send the request to the same pod

```
kubectl exec hn-service-rs-w2llk -- curl -s http://100.70.128.16
{ "hostname":"hn-service-rs-kvss9" }

kubectl exec hn-service-rs-w2llk -- curl -s http://100.70.128.16
{ "hostname":"hn-service-rs-kvss9" }

kubectl exec hn-service-rs-w2llk -- curl -s http://100.70.128.16
{ "hostname":"hn-service-rs-kvss9" }

kubectl exec hn-service-rs-w2llk -- curl -s http://100.70.128.16
{ "hostname":"hn-service-rs-kvss9" }
```

Exposing multiple ports in the same Service

A single service can expose multiple port to forward to the pod's port

```yaml
apiVersion: v1
kind: Service
metadata:
  name: hostname-service
spec:
  ports:
  - name: http
    port: 80
    targetPort: 8080
  - name: https
    port: 443
    targetPort: 8443
  selector:
    app: hn-service
```

Using named ports

Target port can also be referred by name instead of port numbers. Name can be given for pod's port

First in the pod definition, define the name and the container port

```yaml
kind: ReplicaSet
metadata:
  name: hn-service-named-port-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      appNname: hn-service
  template:
    metadata:
      labels:
        appNname: hn-service
        app: hn-service-named-port
    spec:
      containers:
      - name: hn-service-container
        ports:
          - name: http
            containerPort: 8111
        image: classpathio/hn-service
```

In the service definition

```
apiVersion: v1
kind: Service
metadata:
  name: hostname-service-named-port
spec:
  ports:
  - port: 80
    targetPort: http
  selector:
    app: hn-service-named-port
```

> Advantages - Enables to change port numbers without having to change the service spec

Discovering Services

Through envrionmental variables When a pod starts, K8s initializes a set of `environmental variables` pointing to each service at that moment. These environmental variables are set only if the service is created before creating the pod.

Excercise

1. Delete all the pods by deleting the ReplicaSet

```
kubectl delete rs hn-service-named-port-rs
replicaset.apps "hn-service-named-port-rs" deleted
```

1. list all the pods after creating the `Replica-Set`

```
kubectl create -f hn-service-rs.yaml
```

List all the Pods

```
kubectl get po --show-labels
NAME                   READY   STATUS    RESTARTS   AGE   LABELS
hn-service-rs-fvhkw    1/1     Running   0          45s   app=hn-service
hn-service-rs-mqmmc    1/1     Running   0          45s   app=hn-service
hn-service-rs-wbmg8    1/1     Running   0          45s   app=hn-service
```

1. List the environmentsal variables by running the `env` command on the target pod

```
kubectl exec hn-service-rs-mqmmc env
```

Output

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib
HOSTNAME=hn-service-rs-mqmmc
KUBERNETES_SERVICE_HOST=100.64.0.1
HOSTNAME_SERVICE_NAMED_PORT_PORT_80_TCP_ADDR=100.66.116.116
KUBERNETES_SERVICE_PORT=443
KUBERNETES_PORT=tcp://100.64.0.1:443
HOSTNAME_SERVICE_NAMED_PORT_PORT=tcp://100.66.116.116:80
HOSTNAME_SERVICE_NAMED_PORT_PORT_80_TCP=tcp://100.66.116.116:80
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=100.64.0.1
HOSTNAME_SERVICE_NAMED_PORT_SERVICE_HOST=100.66.116.116
HOSTNAME_SERVICE_NAMED_PORT_SERVICE_PORT=80
HOSTNAME_SERVICE_NAMED_PORT_PORT_80_TCP_PROTO=tcp
HOSTNAME_SERVICE_NAMED_PORT_PORT_80_TCP_PORT=80
KUBERNETES_PORT_443_TCP=tcp://100.64.0.1:443
LANG=C.UTF-8
JAVA_HOME=/usr/lib/jvm/java-1.8-openjdk
JAVA_VERSION=8u212
JAVA_ALPINE_VERSION=8.212.04-r0
HOME=/home/spring
```

When a front-end pod, wants to communicate with the backend-pod, the backend-pod should expose a service which will be used by the front-end pod

> Note - Dashes in the service-name will be converted to _ in the environmental variables and all letters will be converted to UPPERCASE.

Discovering services through DNS

- The `kube-system` service contains a pod named `kube-dns` .
- All the pods in the cluster are automatically configured to use.
- K8s modified each containers DNS entry in `/etc/resolv.conf` file
- Any DNS query performed by the process running inside the pod will be handled by K8s DNS server which knows about all the services running inside the system.
- Each service gets a DNS entry in the internal DNS server
- Client pods that know the name of the service, can access the service using the `FQDN` instead of resorting to the `environmental variables`

Example: if the service name is `backend-database` , then the service name will be

```
hostname-service.default.svc.cluster.local
```

The `svc.cluster.local` is optional and the service name can be reffered by `backend-database` omiitting the suffix.

Excercise:

```
kubectl exec -it hn-service-rs-mcmp9 bash
```

Now, inside the container, run the `curl` command

```
bash-4.4$ curl http://hostname-service.default.svc.cluster.local
{ "hostname":"hn-service-rs-mcmp9" }

bash-4.4$ curl http://hostname-service.default
{ "hostname":"hn-service-rs-z25j6" }

bash-4.4$ curl http://hostname-service
{ "hostname":"hn-service-rs-mcmp9" }

bash-4.4$ cat /etc/resolv.conf
nameserver 100.64.0.10
search default.svc.cluster.local svc.cluster.local cluster.local ap-south-
options ndots:5
```

Note: You cannot ping the service because, the service's cluster IP is a virtual IP and only has meaning when combined with the service port.

Connecting to Services outside the cluster

- Client pods can connect to external services outside the cluster
- To leverage load balancing
- Service discovery

Using Service endpoints

> Endpoint resource is the link between service and the pods.

```
kubectl describe svc hostname-service
Name:              hostname-service
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          app=hn-service
Type:              ClusterIP
IP:                100.67.163.196
Port:              <unset>  80/TCP
TargetPort:        8111/TCP
Endpoints:         100.96.1.5:8111,100.96.2.6:8111,100.96.3.4:8111
Session Affinity:  None
Events:            <none>
```

- Endpoint resource is a list of IP addresses and ports exposing a service
- You can fetch the `Endpoint` resource just like any other resource

```
kubectl get endpoints hostname-service
NAME               ENDPOINTS                                         AGE
hostname-service   100.96.1.5:8111,100.96.2.6:8111,100.96.3.4:8111   45m
```

Manually configuring the Service endpoints

- Can be updated manually
- If we create a service without pod selector, endpoints will not even be created
- We then need to create the `Endpoint` resource manually to specify the list of endpoints for the service

Creating a service without a selector

> Before continuing the below steps delete the Replica-Set, Service and Endpoint resource

1. Create a service without the pod selector

```
apiVersion: v1
kind: Service
metadata:
  name: hn-endpiont-service
spec:
  ports:
  - port: 80
```

Create the Service from the manifest file

```
kubectl create -f hn-service-without-endpoints.yaml
```

Creating a Endpoint resource for a service without a selector

```
apiVersion: v1
kind: Endpoints
metadata:
  name: hn-endpiont-service
subsets:
  - addresses:
      - ip: 100.96.1.5
      - ip: 100.96.2.6
      - ip: 100.96.3.4
    ports:
      - port: 8111
```

```
kubectl create -f hn-endpoint.yaml
endpoints/hn-endpiont-service created
```

Create a ReplicaSet

```
kubectl create -f hn-service-rs .yaml
replicaset.apps/hn-service-rs created
```

List the Pods

```
kubectl get po
NAME                  READY   STATUS    RESTARTS   AGE
hn-service-rs-6lwvs   1/1     Running   0          43s
hn-service-rs-984s4   1/1     Running   0          43s
hn-service-rs-n46bk   1/1     Running   0          43s
```

Note

- The name of the service should match the service name
- To migrate the external service to pods running inside K8s, add the selector to the service, thereby makings its endpoints managed automatically
- Also, by removing the selector, from the service, the K8s stops updating its endpoints.

Alias for external service

- Specify the service with `type` field set to `ExternalName`

```
apiVersion: v1
kind: Service
metadata:
  name: external-service
spec:
  type: ExternalName
  externalName: someapi.somecompany.com
  ports:
  - port: 80
```

Exposing services to external Clients

Three ways to make the service accessible to extrenal clients

- Setting the service type to `NodePort`
  - Opens the port at the cluster node level
- Setting the service type to `LoadBalancer`
  - Extension of `NodePort`
  - Makes services accessible thourhg a dedicated *Load Balancer*
  - Redirects traffic to the node port across all the nodes
  - Clients connect to the service using the *Load balancer* IP address
- Creating the service as `Ingress` resource
  - Operates at the *HTTP* level

NodePort service

- All the nodes will have a dedicated port opened at the node level
- Can be accessed through Service's internal cluster IP and reserverd node port

```
apiVersion: v1
kind: Service
metadata:
name: kubia-nodeport
spec:
type: NodePort
ports:
- port: 80
  targetPort: 8080
  nodePort: 30123
selector:
  app: kubia
```

List the service

```
$ kubectl get svc kubia-nodeport
NAME              CLUSTER-IP        EXTERNAL-IP    PORT(S)         AGE
kubia-nodeport    10.111.254.223    <nodes>        80:30123/TCP    2m
```

> Note: The external ip now shows as `<nodes>`

Invoking the service using the nodes

```
$ curl http://130.211.97.55:30123
You've hit kubia-ym8or
$ curl http://130.211.99.206:30123
You've hit kubia-xueq1
```

> Need for the Load Balancer: The Client's cannot access the service in case of the Node failure

External Load Balancer

- Extension of NodePort
- Featuer of cloud providers
- K8s cluster supports automatic provision of load balancer from the cloud infrastructure
- LB will havve public IP and client's will connect to the LB's IP address
- If the cloud provider does not support LB, then the Nodeport will be used
- Service will behave like a NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: hn-service-loadbalancer
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8111
  selector:
    app: hn-service
```

```
kubectl create -f hn-service-loadbalancer.yaml
service/hn-service-loadbalancer created
```

```
kubectl get svc
NAME                         TYPE           CLUSTER-IP       EXTERNAL-IP
                                                PORT(S)         AGE
hn-service-loadbalancer      LoadBalancer   100.67.50.236    aa8b726f0507d43dd
kubernetes                   ClusterIP      100.64.0.1       <none>
                                                443/TCP         15m
```

Bring up the pods with ReplicaSet

```
kubectl get po --show-labels
NAME                        READY    STATUS     RESTARTS    AGE    LABELS
```

```
hn-service-rs-9pk89   1/1     Running   0          31s   app=hn-service
hn-service-rs-np87q   1/1     Running   0          31s   app=hn-service
hn-service-rs-tv6xq   1/1     Running   0          31s   app=hn-service
```

```
curl http://aa8b726f0507d43dda45a79f5ebe881d -722734132.ap-south-1.elb.ama
{ "hostname":"hn-service-rs-9pk89" }
```

To reduce the additional network hop, set the `externalTrafficPolicy` inside the `spec` section and set the value to `Local`

```
spec:
  externalTrafficPolicy: Local
  ...
```

> Note:
>
> - By using this feature, if the pod does not exists, then the connection will hang
> - Also, the load balancer will not the connection will be forwarded to the same pod

Ingress resource

- Each LoadBalancer service requires its own load balancer with its own Pulic IP address
- Single Ingress requires only on public IP address when providing access to multiple services
- The `host` and the `path` will determin the service to be forwarded the request to.
- Operates at the `HTTP` layer and hence provide features like cookie based session affinity

Creating Ingress resource

Creating an Ingress resource

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: hn-service-ingress
spec:
  rules:
  - host: hostnameservice.classpath.com
    http:
      paths:
      - path: /
        backend:
          serviceName: hostname-service
          servicePort: 80
```

```
kubectl create -f hn-service-ingress.yaml
ingress.extensions/hn-service-ingress created
```

```
kubectl get ingress
NAME                 HOSTS                           ADDRESS   PORTS   AGE
hn-service-ingress   hostnameservice.classpath.com             80      21s
```

Create the Service

```
kubectl create -f hn-service.yaml
service/hostname-service created
```

Create the ReplicaSet backed with the Servcie created above

```
create -f hn-service-rs.yaml
```

```
replicaset.apps/hn-service-rs created
```

List the Service

```
kubectl get svc
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
hostname-service    ClusterIP   100.65.248.53   <none>        80/TCP    2m2
kubernetes          ClusterIP   100.64.0.1      <none>        443/TCP   27m
```

List the Pods

```
kubectl get pods --show-labels
NAME                   READY   STATUS    RESTARTS   AGE   LABELS
hn-service-rs-dzqml    1/1     Running   0          33s   app=hn-service
hn-service-rs-nll4z    1/1     Running   0          33s   app=hn-service
hn-service-rs-q9g7g    1/1     Running   0          33s   app=hn-service
```

List the ingress-service

```
kubectl get svc nginx-ingress --namespace=nginx-ingress
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP
                                    PORT(S)                          AGE
nginx-ingress   LoadBalancer   100.71.193.132   a9a7caaf3056d4566aa316dffa
```

Perform nslookup to fetch the IP address

```
nslookup a9a7caaf3056d4566aa316dffabeb368-17 8615173.ap-south-1.elb.amazon
Server:         172.31.0.2
Address:        172.31.0.2#53

Non-authoritative answer:
Name:   a9a7caaf3056d4566aa316dffabeb368-178615173.ap-south-1.elb.amazonaw
Address: 13.232.18.133
Name:   a9a7caaf3056d4566aa316dffabeb368-178615173.ap-south-1.elb.amazonaw
```

```
Address: 13.126.101.254
```

Configugre the DNS to point the domain to the IP address in the `/etc/hosts` entry file

```
13.126.101.254 hostnameservice.classpath.com
```

Accessing the service with the domain name

```
curl http://hostnameservice.classpath.com
{ "hostname":"hn-service-rs-q9g7g" }
```

Exposing multiple services through the same Ingress

Expose multiple paths on the same host

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: hn-service-ingress
spec:
  rules:
  - host: hostnameservice.classpath.com
    http:
      paths:
      - path: /
        backend:
          serviceName: hostname-service
          servicePort: 80
     - path: /hosts
        backend:
          serviceName: hostname-service
          servicePort: 80
```

Expose multiple hosts

```
spec:
  rules:
  - host: hostnameservice.classpath.com
    http:
      paths:
      - path: /
        backend:
          serviceName: hostname-service
          servicePort: 80
  - host: bar.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: bar
          servicePort: 80
```

Configuring TLS

- Delete the `hn-service-ingress` resource

Create a secret

```
kubectl create -f hn-secret.yaml
secret/hn-secret created
```

```
kubectl get ingresses
NAME                 HOSTS                            ADDRESS   PORTS   AGE
hn-service-ingress   hostnameservice.classpath.com              80      34m
```

```
kubectl delete ingress hn-service-ingress
ingress.extensions "hn-service-ingress" deleted
```

- Create a Manifest file

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: hn-service-ingress-https
spec:
  tls:
  - hosts:
    - hostnameservice.classpath.com
    secretName: hn-secret
  rules:
  - host: hostnameservice.classpath.com
    http:
      paths:
      - path: /
        backend:
          serviceName: hostname-service
          servicePort: 80
```

```
kubectl create -f hn-service-ingress-https.yaml
ingress.extensions/hn-service-ingress-https created
```

```
kubectl get ingresses
NAME                         HOSTS                            ADDRESS   PORTS
hn-service-ingress-https     hostnameservice.classpath.com              80, 4
```

Accessing the service

```
curl https://hostnameservice.classpath.com --insecure
{ "hostname":"hn-service-rs-q9g7g" }
```

Readiness Probe

- Invoked periodically and determine whether the specified pod should receive client requests or not.
- If the container's rediness probe returns success, it's signalling that it is ready to accept client's requests.
- Defining readiness probe is specific to each container
- K8s can check if the container is ready by hitting the `GET` request specified
- Its the container's responsibility to accepts the `GET` endpoint and perform the checks and return the status.
- When a container is started, K8s can be configured to perform the readiness probe after a configured amount of time
- After the first readiness probe, the K8s performs the probe frequently
- If the pod is not ready, it is removed from the service
- If the pod is ready, it is added to the service
- Unlike the liveness probe, the pod will not be killed or restarted
- Liveness probe is to keep healthy pods by killing and replacing with the healthy pods
- Readiness probe is to ensure that only the pods which can accept client's request are added to the service

Types of Readiness Probes

- Exec - Process is executed. Process exit status code determines the readiness
- HTTP GET - HTTP status code determines the readiness of the container
- TCP Socket - opens up the TCP connection to a specified port. The connection establishment determines, if the service is ready.

Creating a Readiness probe

Create the `ReplicaSet` definition

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: hn-service-readinessrs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nodejs-readiness
  template:
    metadata:
      labels:
        app: nodejs-readiness
    spec:
      containers:
      - name: nodejs-readiness-container
        image: classpathio/nodejs-readiness-app
        readinessProbe:
          httpGet:
            path: /readiness
            port: 8080
          initialDelaySeconds: 10
          periodSeconds: 5
```

```
kubectl create -f hn-service-readiness-rs.yaml
replicaset.apps/hn-service-readinessrs created
```

- If the file exists, it returns 0 indicating success
- Non 0 if the file does not exists indicating failure

List all the pods

```
kubectl get po
NAME                            READY   STATUS    RESTARTS   AGE
hn-service-readinessrs-52m69    0/1     Running   0          71s
hn-service-readinessrs-87mk4    0/1     Running   0          71s
hn-service-readinessrs-rbzcx    0/1     Running   0          71s
```

```
kubectl exec -it hn-service-readinessrs-87mk 4 bash
root@hn-service-readinessrs-87mk4:/usr/src/app#
```

```
curl http://localhost:8080/readiness
{"status":"Not up"}

```

```
curl http://localhost:8080/init -XPOST
{"status":"success"}
```

```
kubectl get po
NAME                           READY   STATUS    RESTARTS   AGE
hn-service-readinessrs-52m69   0/1     Running   0          4m26s
hn-service-readinessrs-87mk4   1/1     Running   0          4m26s
hn-service-readinessrs-rbzcx   0/1     Running   0          4m26s
```

Hit the Service from the ec2-instance

```
 kubectl exec hn-service-readinessrs-87mk4 cu rl http://localhost:8080/rea
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
                                 Dload  Upload   Total   Spent    Left  Sp
100    15  100    15    0     0   2035      0 --:--:-- --:--:-- --:--:--
{"status":"Up"}
```

Optional steps

- Make the Pod not-ready again

```
kubectl exec hn-service-readinessrs-87mk4 --  curl http://localhost:8080/d
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
                                 Dload  Upload   Total   Spent    Left  Sp
100{"status":"down"}    17  100    17    0     0   7692      0 --:--:-- --
```

- All Pods should now be in `Not-Ready` state again

```
kubectl get po
NAME                           READY   STATUS    RESTARTS   AGE
hn-service-readinessrs-52m69   0/1     Running   0          8m30s
hn-service-readinessrs-87mk4   0/1     Running   0          8m30s
hn-service-readinessrs-rbzcx   0/1     Running   0          8m30s
```

Headless service

- In cases when the clients need to talk to all the pods
- Can get the IP address of all the pods by connecting to the API server, but not ideal since it will be decoupled with the K8s.
- Setting up the `ClusterIP` to `None` , K8s does not return the cluster IP. The DNS server will return the list of pod IP addresses.
- Clients talk directly to the pod's IP address rather than the service proxy

Creating a headless service

```
apiVersion: v1
kind: Service
metadata:
  name: hostname-service
spec:
  ClusterIP: None
  ports:
  - name: http
    port: 80
    targetPort: 8080
  - name: https
    port: 443
    targetPort: 8443
  selector:
    app: hn-service
```

Create the service using the `kubectl create` command

Inspect the `Cluser IP` field by running the `kubectl describe` command

```
$ kubectl exec <pod name>
```

Volumes

Volume

- Process running inside the pods share the resources like CPU, RAM, Network interfaces
- Disks are not shared , container inside the pod has its own isolated filesystem
- File system comes from the container's image
- `Volumes` are not top level resources like `RC` , `RS`
- Is part of the pod definition
- Share the same lifecycle as of the pod
- Volume's content will be persisted across pod restarts
- Volumne is available for all the containers inside the pod
- It must be mounted in each container that need access to the volume
- Can mount the volume in any location of the file system

Available Volume types

- emptyDir - a simple directory
- hostPath - a directory on the node mounted to the pods
- gitRepo - Volume initialized by cloning the contents from the git repository
- nfs - NFS share mount into the pod
- awsElasticBlockStorage - AWS EBS volume
- azureDisk - Azure
- persistentVolumeClaim - dynamically provisioned persistent storage

Using emptyDir volume

- Simple solution
- Starts with a empty directory
- Volume lifetime is tied to the pod's lifetime
- Useful for sharing data between containers in the same pod

```
apiVersion: v1
kind: Pod
metadata:
  name: fortune
spec:
  containers:
  - image: luksa/fortune
    name: html-generator
    volumeMounts:
    - name: html
      mountPath: /var/htdocs
  - image: nginx:alpine
    name: web-server
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP
  volumes:

  - name: html

    emptyDir: {}
```

```
kubectl get po
NAME     READY   STATUS    RESTARTS   AGE
fortune  2/2     Running   0          15s
```

Testing the pod

```
$ kubectl port-forward fortune 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

Test the pod by invoking the HTTP request

```
curl http://localhost:8080
Noise proves nothing.  Often a hen who has merely laid an egg cackles
as if she laid an asteroid.
```

Try invoking the HTTP request after some time

```
curl http://localhost:8080
Q:      What's the difference between a Mac and an Etch-a-Sketch?
A:      You don't have to shake the Mac to clear the screen.
```

- You should see a different message this time
- The volume is created on the worker node hosting the pod
- You can also set the `medium` to `Memory` which creates an in-memory

Using Git Repo

- Similar to `emptyDir` but the content will be populated by cloning from git repository
- It is not kept in-sync in case of updates to the repository

```
apiVersion: v1
kind: Pod
metadata:
  name: gitrepo-volume-pod
spec:
  containers:
  - image: nginx:alpine
    name: web-server
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP
  volumes:
  - name: html
    gitRepo:
      repository: https://github.com/luksa/kubia-website-example.git
      revision: master
      directory: .
```

```
kubectl create -f gitrepo-volume-rs.yaml
replicaset.apps/gitrepo-rs created
```

```
kubectl get rs
NAME         DESIRED   CURRENT   READY   AGE
gitrepo-rs   2         2         2       27s
```

Two Options

- Using Services and Ingress
- Port Forwarding

Service and Ingress

```
kubectl create -f git-repo-service.yaml
service/git-repo-service created
```

Create a Ingress Manifest

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: git-repo-ingress
spec:
  tls:
  - hosts:
    - blog.classpath.io
    secretName: cafe-secret
  rules:
  - host: blog.classpath.io
    http:
      paths:
      - path: /
        backend:
          serviceName: git-repo-service
          servicePort: 80

```

```
kubectl create -f volumes/git-repo-ingress-https.yaml
ingress.extensions/hn-service-ingress created
```

```
kubectl get ingresses
NAME                 HOSTS                           ADDRESS   PORTS     A
hn-service-ingress   hostnameservice.classpath.com             80, 443   4
```

Edit the `/etc/hosts` entry with the `hostnameservice.classpath.io`

```
13.232.188.216  curl https://blog.classpath.io
```

```
curl https://blog.classpath.io --insecure
<html>
<body>
  <h1> Hello World - v2!! </h1>
</body>
</html>

curl https://blog.classpath.io --insecure
<html>
<body>
  <h1> Hello World !! </h1>
</body>
</html>
```

Port forwarding - Simple :-)

```
kubectl port-forward gitrepo-rs-4x4w5 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

In another terminal run the curl command

```
curl http://localhost:8080
<html>
<body>
  <h1> Hello World !! </h1>
</body>
</html>
```

Delete the Pod

```
kubectl delete po gitrepo-rs-4x4w5
pod "gitrepo-rs-4x4w5" deleted
```

List the Pods

```
kubectl get po
NAME               READY   STATUS    RESTARTS   AGE
gitrepo-rs-4kr6b   1/1     Running   0          39s
gitrepo-rs-m97wr   1/1     Running   0          50m
```

```
kubectl port-forward gitrepo-rs-4kr6b 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

```
curl http://localhost:8080
<html>
<body>
  <h1> Hello World - v2!! </h1>
</body>
</html>
```

Old Pod with older version of git

```
kubectl port-forward gitrepo-rs-m97wr 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
```

```
curl http://localhost:8080
<html>
<body>
  <h1> Hello World !! </h1>
</body>
</html>
```

- When the pod is deleted, the volume is also deleted

Hostpath Volume

- Volume is not deleted even if the pod is deleted
- Volume's content is stored on the node
- System level services (DaemonSet) can leverage this feature
- Makes the pod sensitive to the node it is scheduled on

Using Persistent storage

Create a Volume in Elastic Block Storage and create a tag with key as ***KubernetesCluster*** and value as ***classpath.k8s.local***

Also create the Volume inside the same availablity zone

Create a Pod definition

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  volumes:
  - name: mongodb-data
    awsElasticBlockStore:
      volumeID: vol-09267ef3713fb2ede
      fsType: ext4
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
```

- Pod contains a single container and a single volume baked by the Persistent disk
- Mounted the volume at `/data/db`
- Now run the `MongoDB` inside the shell

```
kubectl create -f mongo-db-data.yaml
```

```
kubectl exec -it mongodb mongo
MongoDB shell version: 3.2.8
```

```
connecting to: mongodb://127.0.0.1:27017
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    http://docs.mongodb.org/
Questions? Try the support group
    http://groups.google.com/group/mongodb-user
...
>
```

Insert some documents inside the MongoDB

```
> use store
switched to db store
> db.cart.insert({name:'classpath'})
WriteResult({ "nInserted" : 1 })
> db.cart.find()
{ "_id" : ObjectId("5e85f350f5b6271cca763c1d"), "name" : "classpath" }
```

Recreating the pod by deleting the pod

```
kubectl delete pod mongodb
pod "mongodb" deleted

kubectl create -f mongo-db-data.yaml
pod "mongodb" created
```

Retriving the persisted data after recreating the pod

```
kubectl exec -it mongodb mongo

> use store
switched to db store
> db.cart.find()
{ "_id" : ObjectId("5e85f350f5b6271cca763c1d"), "name" : "classpath" }
```

Decoupling the pods from underlying storage technology

- K8s allows us to decouple the underlying storage technology
- Makes our application to switch between technology easily

PersistentVolumes and PersistentVolumeClaim

Steps to use the PersistentVolume and PersistentVolumeClaim

Cluster Admin sets up the PersistentVolume

- Cluster admin sets up the storage and registers with K8s
- Registering the volume is done by creating a PersistentVolume resource through K8s API Server
- When creating the PersistentVolume, the admin specifies the size and access mode it supports (read / write)

Cluster user then creates a PersistentVolumeClaim

- User creates a `PersistentVolumeClaim` resource
- Specifies the minimum size and the access mode
- User submits the `PersistentVolumeClaim` to K8s's API server
- K8s finds the suitable `PersistentVolume` an binds to the volume to the `PersistentVolumeClaim`
- The PersistentVolumeClaim is now ready to be mounted on any pods
- Other users cannot use the same persistentVolumeclaim until the resource is release by deleting the resource.

Creating the PersistentVolume

Creating PersistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
  - ReadWriteOnce
  - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  awsElasticBlockStore:
    volumeID: vol-09267ef3713fb2ede
    fsType: ext4
```

```
kubectl create -f aws-persistent-volume.yaml
```

Fetch the PersistentVolume

```
kubectl get pv
```

Creaing a PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
  - ReadWriteOnce
  storageClassName: ""
```

```
kubectl create -f mongodb-pvc.yaml
```

- K8s will find the appropriate `PersistentVolume` and binds to the claim
- The `PersistentVolume` should be larger than the `PersistentVolumeClaim`
- The volume's access modes must include the access modes requested by the claim.

List the PersistentVolumeClaim

```
$ kubectl get pvc
NAME          STATUS    VOLUME       CAPACITY   ACCESSMODES   AGE
mongodb-pvc   Bound     mongodb-pv   1Gi        RWO,ROX       3s
```

- RWO—ReadWriteOnce— Only a single node can mount the volume for reading and writing.
- ROX—ReadOnlyMany—Multiple nodes can mount the volume for reading.
- RWX—ReadWriteMany—Multiple nodes can mount the volume for both reading and writing.

List the `PersistentVolume`

```
$ kubectl get pv
NAME         CAPACITY   ACCESSMODES   STATUS   CLAIM                AGE
```

Using the PersistentVolumeClaim inside the Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
```

```
      protocol: TCP
  volumes:
  - name: mongodb-data
    persistentVolumeClaim:
      claimName: mongodb-pvc
```

```
kubectl create -f mongodb-with-pvc.yaml
```

Retrieve the data persisted

```
kubectl exec -it mongodb mongo

> use store
switched to db store
> db.cart.find()
{ "_id" : ObjectId("5e85f350f5b6271cca763c1d"), "name" : "classpath" }
```

Recycling PersistentVolumeClaim

```
$ kubectl delete pod mongodb
pod "mongodb" deleted
$ kubectl delete pvc mongodb-pvc
persistentvolumeclaim "mongodb-pvc" deleted
```

Status of `PersistentVolumeClaim` after deleting

```
kubectl get pv
NAME        CAPACITY  ACCESSMODES  STATUS    CLAIM               REASON AG
mongodb-pv  1Gi       RWO,ROX      Released  default/mongodb-pvc        5m
```

```
kubectl get pvc
```

```
NAME           STATUS    VOLUME       CAPACITY    ACCESSMODES    AGE
mongodb-pvc    Pending                                           13s
```

Dynamic provision of PersitentVolumes

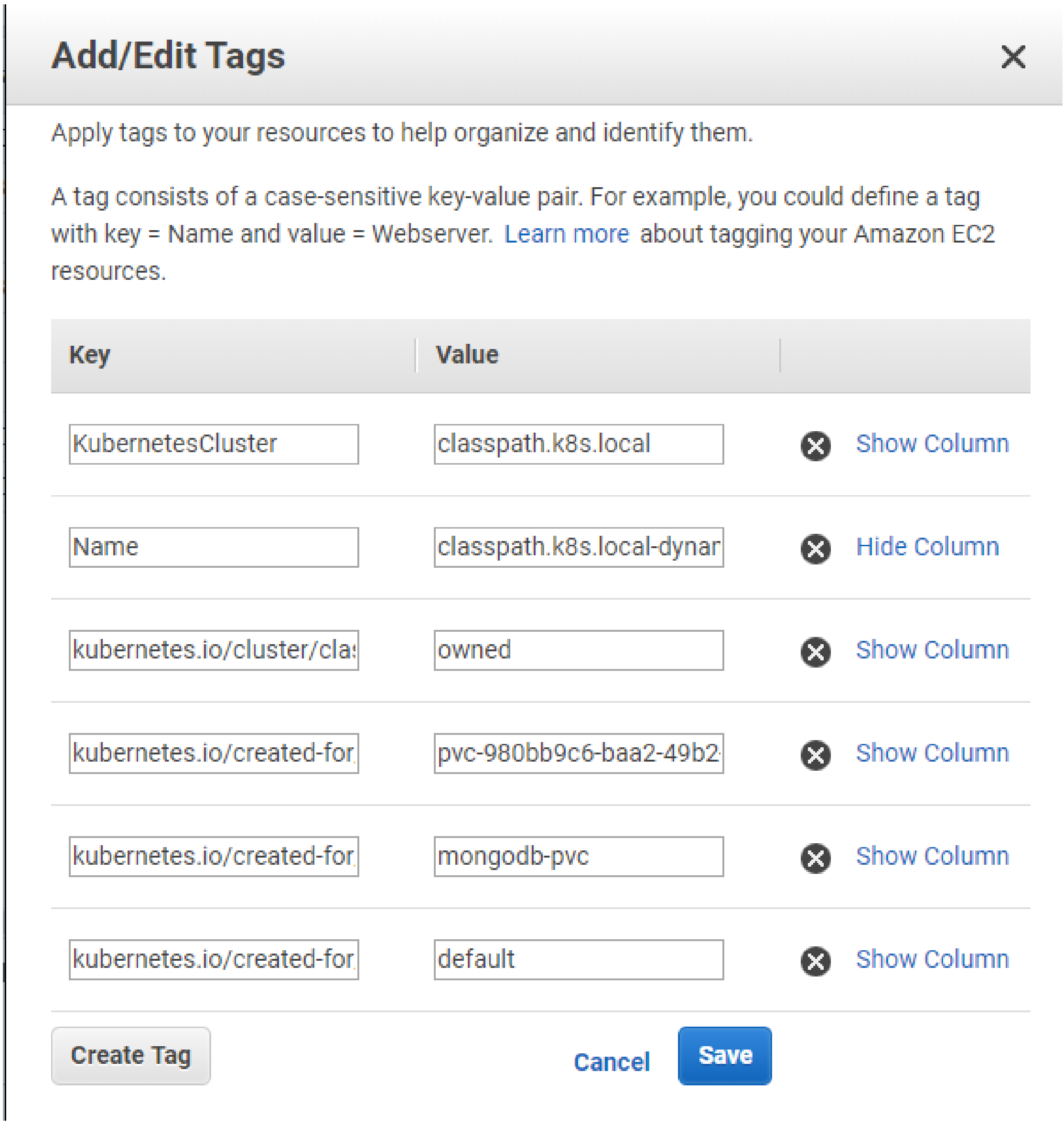
- PersistentVolume is still vendor dependent
- Cluster admin will have to configure the space upfront
- Cluster admin can instead deploy a PersistentVolume provisioner
- Define one or more `StorageClass` in their `PersistentVolumeClaim`
- Provisioner will take into account when provisioning the persistent storage
- This ways, the provisioner can provision from logically infinite storage and not upfront create the volume.
- `StorageClass` resource aren't namespace
- K8s have provisioner for most cloud providers
- Custom provisioner should be deployed in case of a on-premise
- Cluster admin will pre-provision one or more `StorageClass` instead of `PersistentVolume`

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

```
create -f aws-storage-class.yaml
storageclass.storage.k8s.io/standard created
```

```
kubectl get sc
```

```
NAME            PROVISIONER            AGE
default         kubernetes.io/aws-ebs   4m47s
gp2 (default)   kubernetes.io/aws-ebs   4m47s
standard        kubernetes.io/aws-ebs   70s
```



Add/Edit Tags ✕

Apply tags to your resources to help organize and identify them.

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. Learn more about tagging your Amazon EC2 resources.

Key	Value	
KubernetesCluster	classpath.k8s.local	⊗ Show Column
Name	classpath.k8s.local-dynar	⊗ Hide Column
kubernetes.io/cluster/cla:	owned	⊗ Show Column
kubernetes.io/created-for	pvc-980bb9c6-baa2-49b2	⊗ Show Column
kubernetes.io/created-for	mongodb-pvc	⊗ Show Column
kubernetes.io/created-for	default	⊗ Show Column

PersitentVolume

Creating a `PersistentVolumeClaim` to use dynamic provisioning.

```
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:
  name: mongodb-sc-pvc
spec:
  storageClassName: standard
  resources:
    requests:
      storage: 100Mi
  accessModes:
    - ReadWriteOnce
```

```
kubectl create -f aws-pvc-with-storage-class.yaml

persistentvolumeclaim/mongodb-sc-pvc created
```

List the PersistentVolumeClaim

```
 kubectl get pvc
NAME             STATUS   VOLUME                                     CAPAC
mongodb-sc-pvc   Bound    pvc-aa387010-f5fb-4429-8380-628c708e8a26   1Gi
         standard       19s
```

List the PersistentVolume

```
kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLA
STATUS     CLAIM                    STORAGECLASS   REASON   AGE
pvc-980bb9c6-baa2-49b2-aa84-066862270284   1Gi        RWO            Retai
Released   default/mongodb-pvc      standard                14m
pvc-aa387010-f5fb-4429-8380-628c708e8a26   1Gi        RWO            Retai
Bound      default/mongodb-sc-pvc   standard                71s
pvc-f5346f31-4c44-480b-adb2-fcad0818be46   1Gi        RWO            Retai
Released   default/mongodb-pvc      standard                3m52s
```

Create the Mongodb Pod

```
kubectl create -f mongodb-with-pvc.yaml
pod/mongodb created
```

```
Events:
  Type    Reason                  Age   From
        Message
  ----    ------                  ----  ----
        -------
  Normal  Scheduled               64s   default-scheduler
        Successfully assigned default/mongodb to ip-172-20-55-45.ap-south-
  Normal  SuccessfulAttachVolume  58s   attachdetach-controller
        AttachVolume.Attach succeeded for volume "pvc-aa387010-f5fb-4429-8
  Normal  Pulling                 55s   kubelet, ip-172-20-55-45.ap-south-
  Normal  Pulled                  41s   kubelet, ip-172-20-55-45.ap-south-
  Normal  Created                 41s   kubelet, ip-172-20-55-45.ap-south-
  Normal  Started                 41s   kubelet, ip-172-20-55-45.ap-south-
```

Assignment:

1. Create the Mongodb Pod
2. Log into to mongo shell
3. Insert the record and exit
4. Delete the Pod
5. Recreate the Pod
6. Log into the Mongo shell
7. Find the inserted record

Delete the Pod

```
kubectl delete po mongodb
pod "mongodb" deleted
```

Delete the PVC

```
kubectl delete pvc mongodb-sc-pvc
persistentvolumeclaim "mongodb-sc-pvc" deleted
```

Delete the Storage Class

```
kubectl delete sc --all
storageclass.storage.k8s.io "default" deleted
storageclass.storage.k8s.io "gp2" deleted
storageclass.storage.k8s.io "standard" deleted
```

Delete the PV

```
kubectl delete pv --all
persistentvolume "pvc-980bb9c6-baa2-49b2-aa84-066862270284" deleted
persistentvolume "pvc-aa387010-f5fb-4429-8380-628c708e8a26" deleted
persistentvolume "pvc-f5346f31-4c44-480b-adb2-fcad0818be46" deleted
```

ConfigMaps

ConfigMaps

By default when we first start building our application

- First pass the arguments using command line
- Then when the arguments become difficult to manage, move them to external configuration file
- In containerized applications we pass the arguments using the environmental variables. Ex: mysql password

Passing Environmental Variables inside a Container

- The config file has to be baked into the image directly
- Similar to hard coding in the source file since it requires to rebuild the image every time we change the configuration entries
- Anyone having access to the image can see all the information including sensitive information.
- Alternative method would be to mount the volume containing the config file into the container

Ways of Configuring properties

- Passing command line arguments to containers
- Setting custom environmental variable for each container
- Mounting configuration files into containers through a special type of volume

Defining command line arguments in Docker

- Command that gets executed inside the container contains two parts namely command and arguments.
- `ENTRYPOINT` – command to be executed
- `CMD` – specifies the arguments to be passed to the command

Then the image can be run without passing the command

```
docker run <image>
```

Or override the arguments which was set under the `CMD` in the dockerfile

```
docker run <image> <arguments>
```

```
shell-form - ENTRYPOINT node app.js

exec-form - ENTRYPOINT ["node", "app.js"]
```

> Always use the `exec` mode of running docker images

Overriding command and arguments in K8s

- In K8s we can override both the `ENTRYPOINT` and the `CMD`

```
kind: Pod
spec:
  containers:
  - image: some/image
    command: ["/bin/command"]
    args: ["arg1", "arg2", "arg3"]
```

> The command and args fields can’t be updated after the pod is created.

Docker command	Kubernetes Command
ENTRYPOINT	**command**
CMD	**args**

Running with custom delay interval

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: talkingcow-args
spec:
  containers:
  - image: classpathio/talking-cow-args
    name: talkingcow-args
    args: ["4"]
```

To pass more than one or few arguments, you can follow the below syntax

```yaml
args:
    - one
    - two
    - "10"
```

Setting Environmental variables for a container

- Containerized applications often use env variables as a source of configuration option
- K8s allows to specify custom list of env variables for each container of the pod
- Currently no option exists to set the env variables at the pod level and inherit by its containers
- The environmental variables cannot be updated after the pod is created

Defining environmental variable in the pod definition

Define env at the container level and not at the pod level

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: talkingcow-env
spec:
  containers:
  - image: classpathio/talking-cow-env
    name: talkingcow-env-container
    env:
    - name: INTERVAL
      value: "15"
```

Environment variables can be reffered inside the pod definition

```yaml
env:
- name: FIRST_VAR
  value: "foo"
- name: SECOND_VAR
  value: "$(FIRST_VAR)bar"
```

> The SECOND_VAR value will be "foobar".

> *Hard coding the **environmental variables** in the pod definition makes it difficult to manage across different environments as we would need to maintain different pod definitions for each environments.*

> ***Decouple configurations out of the POD descriptor with the help of ConfigMaps***

ConfigMaps

- ConfigMap is a map containing key and value
- Value containing simple strings to full config files
- Application does not need to know that the config map exists
- Application does not read the config map directly
- Contents of the map is either sent as either env variables or as files in a volume
- Can also be passed as command line arguments referred with `${env_var}`
- This allows for keeping multiple manifest files with same name, each for different environments

Creating a Config Map

```
kubectl create configmap config-sleep-10 --from-literal=sleep-interval=10
```

> All examples use the create command to create the configmaps and secret.

ConfigMap with multiple keys

```
kubectl create configmap myconfigmap

   --from-literal=foo=bar --from-literal=bar=baz --from-literal=one=two
```

View the YAML definition

```
apiVersion: v1
data:
  sleep-interval: "4"
kind: ConfigMap
metadata:
  name: fortune-interval-4

```

Creating the ConfigMap using the yaml definition descriptor

```
$ kubectl create -f config-map-interval.yaml
```

Creating a config entry from the contents of a file

```
kubectl create configmap my-config --from-file=config-map-files.conf
```

> In the above case, the filename will be used as a key

Can also specify the key explicitly

```
kubectl create configmap my-config --from-file=customkey=config-file.conf
```

Import all the files from a directory

```
kubectl create configmap my-config --from-file=/path/to/dir
```

> In the above case, each file name will be the key with value referring to the contents of the file

Combine all of them like below:

```
kubectl create configmap my-config --from-file=foo.json  --from-file=bar=f
```

Getting the values of the configmap inside the container

- Setting an environmental variable
- Command line argument
- Volume to expose ConfigMap entries as files

Passing as Environmental Variable

```
apiVersion: v1
kind: Pod
metadata:
  name: talkingcow-configmap-env
spec:
  containers:
  - image: classpathio/talking-cow-env
    name: talkingcow-env-configmap-container
    env:
    - name: INTERVAL
      valueFrom:
        configMapKeyRef:
          name: fortune-interval-4
          key: sleep-interval
```

`INTERVAL` is the environmenal variable name and the value is picked up from the key named `sleep-interval` inside the `fortune-config` ConfigMap

If the referenced ConfigMap doesn’t exist when you create the pod. Kubernetes schedules the pod normally and tries to run its containers.

The container referencing the non-existing ConfigMap will fail to start, but the other container will start normally.

> If you then create the missing ConfigMap, the failed container is started without requiring you to recreate the pod.

Passing all the environmental properties at once

```
spec:
  containers:
  - image: some-image
    envFrom:
    - prefix: CONFIG_
      configMapRef:
        name: my-config-map
...
```

Passing as Command Line arguments

- First initialize the env variable with the value of configmap entry
- Refer that variable inside the argument

```
apiVersion: v1
kind: Pod
metadata:
  name: talkingcow-configmap-args
spec:
  containers:
  - image: classpathio/talking-cow-env
    name: talkingcow-env-configmap-container
    env:
    - name: INTERVAL
      valueFrom:
        configMapKeyRef:
          name: fortune-interval-4
          key: sleep-interval
    args: ["$(INTERVAL)"]
```

Exposing configMap entries as volumes

- For passing entire configuration files we can use configMap volume
- Special type of Volume Mount
- Will expose each entry of the ConfigMap as a file
- Process running inside the container can obtain the entry’s value by reading the contents of the file
- Can also be used to send simple config entries and single values.

Creating a Configmap Volume

```
server {
  listen              82;
  server_name         www.classpath.io;

  gzip on;
  gzip_types text/plain application/xml;

  location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
  }
}

```

Delete the ConfigMap

```
kubectl delete configmap config-map-name
```

Creating a configmap with the config-map-files stored on the disk

Steps

1. Create a new directory called **config-map-files**
2. Store the nginx-conifuguration file inside the directory
3. Create a configmap resource with the below command

```
kubectl create configmap nginx-config --from-file=config-map-files
configmap/nginx-config created
```

```
kubectl get configmaps
NAME           DATA   AGE
nginx-config   2      65s
```

```
kubectl get configmaps nginx-config -o yaml
apiVersion: v1
data:
  nginx-configuration.conf: |
    erver {
      listen              82;
      server_name         www.classpath.io;

      gzip on;
      gzip_types text/plain application/xml;

      location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
      }
    }
  sleep-interval.conf: |
    4
kind: ConfigMap
metadata:
  creationTimestamp: "2020-04-06T04:26:43Z"
  name: nginx-config
  namespace: default
  resourceVersion: "968"
  selfLink: /api/v1/namespaces/default/configmaps/nginx-config
  uid: 51f57c9c-d3d1-4777-8214-c9109f5eec02
```

For Nginx, the server will load all the configuration file under /etc/nginx/conf.d directory.

> We need to place our configuration under that folder and not replace the nginx.conf file

Pod definition

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-configmap-volume
spec:
  containers:
  - image: nginx:alpine
    name: webserver
    volumeMounts:
    - name: configuration
      mountPath: /etc/nginx/conf.d
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP
  volumes:
  - name: configuration
    configMap:
      name: nginx-config
```

Verifying if Nginx is using the Configuration

```
curl -H "Accept-Encoding: gzip" -I http://localhost:8000
HTTP/1.1 200 OK
Server: nginx/1.17.9
Date: Mon, 06 Apr 2020 05:16:10 GMT
Content-Type: text/html
Last-Modified: Tue, 03 Mar 2020 17:36:53 GMT
Connection: keep-alive
ETag: W/"5e5e95b5-264"
Content-Encoding: gzip
```

Examining the contentes of ConfigMap in the Pod

```
kubectl exec nginx-configmap-volume ls /etc/nginx/conf .d
nginx-configuration.conf
sleep-interval
```

- Both entries of ConfigMap have been added as files to the directory
- Could create multiple configmaps to specific container

Exposing certain ConfigMaps entries in the Volume

To define specific entries as files in ConfigMap volume, use the Volumes `items` attribute

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-configmap-volume
spec:
  containers:
  - image: nginx:alpine
    name: webserver-config
    volumeMounts:
    - name: configuration
      mountPath: /etc/nginx/conf.d
      readOnly: true
    ports:
    - containerPort: 82
      protocol: TCP
  volumes:
  - name: configuration
    configMap:
      name: nginx-config
      items:
      - key: nginx-configuration.conf
        path: gzip.conf
```

Examining the contentes of ConfigMap in the Pod this time

```
$ kubectl exec fortune-configmap-volume -c web-server ls /etc/nginx/conf.d
my-nginx-config.conf
```

> Mounting a directory hides the existing files in that directory as long as the directory is mounted in Linux systems

Mounting individual ConfigMap entries as files without hiding other files in the directory

- Add the `subPath` property on the `volumeMount` allows to mount either a single file or single directory from the volume instead of mounting the entire volume

```
spec:
  containers:
  - image: some/image
    volumeMounts:
    - name: myvolume
      mountPath: /etc/someconfig.conf
      subPath: myconfig.conf
```

Setting the file permissions for files in the ConfigMap

```
volumes:
  - name: config
    configMap:
      name: fortune-config
      defaultMode: "6600"
```

Updating apps configuration without restarting the app

- Configuration entries passed via env variables and command line arguments cannot be updated while the proces is running
- Using ConfigMap and exposing through volume makes updating the configuration change without restarting the container
- Its up to the process to detect the changes and reload them

Editing the ConfigMap

```
kubectl edit configmap fortune-config
```

Edit the configuration changes

```
$ kubectl exec fortune-configmap-volume -c web-server
➥   cat /etc/nginx/conf.d/my-nginx-config.conf
```

Signallng the process to reload the configuration

```
$ kubectl exec fortune-configmap-volume -c web-server -- nginx -s reload
```

Now make the request and verty the response

```
kubectl port-forward fortune-configmap-volume 8080:80 &
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
$ curl -H "Accept-Encoding: gzip" -I localhost:8080
HTTP/1.1 200 OK
Server: nginx/1.11.1
Date: Thu, 18 Aug 2016 11:52:57 GMT
Content-Type: text/html
Last-Modified: Thu, 18 Aug 2016 11:52:55 GMT
Connection: keep-alive
ETag: W/"57b5a197-37"
Content-Encoding: gzip
```

⚠ If file is mounted instead of whole volume, the file will not be updated

> Containers should be immutable
>
> Don't bypass the immutability by updating the configmap while containers are running
>
> If App's do not support dynamic reloading, this is not an issue

Secrets

- Secrets are K8s resource to store sensitive configuration data
- Similar to ConfigMaps - Holds key-value pairs
- Can be passed to containers as `environment variables`
- Expose secret entries as files in a Volume
- K8s ensures that the secrets is only distributed to the nodes that run the pods that need the secret
- On the Nodes, the secrets are stored `in-memory` and not persisted to disk

Default secret token in each container

- Every pod has a secret volume attached to it by default

```
kubectl descrie pod-name
```

```
Volumes:
  default-token-cfee9:
    Type:       Secret (a volume populated by a Secret)
    SecretName: default-token-cfee9
```

Secrets are first class K8s resource

```
kubectl get secrets
```

```
kubectl describe secrets
```

Secrets contains three entries

- ca.cert
- namespace
- token

The above represents everything needed to securely talk to `API server`

```
kubectl descrie pod

...
Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-cfee9
...
```

```
kubectl exec mypod ls /var/run/secrets/kubernetes.io/serviceaccount/
ca.crt
namespace
token
```

Creating a Secret

```
openssl genrsa -out https.key 2048
openssl req -new -x509 -key https.key -out https.cert -days
3650 -subj /CN=www.classpath.io
```

Create a dummy file called `domain`

```
echo www.classpath.io > domain
```

Create a secret file

```
kubectl create secret generic nginx-https --from-fil e=certs/https.key --fro
kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-cs2f7   kubernetes.io/service-account-token   3      7m8s
nginx-https           Opaque                                3      2m20s
```

Comparing ConfigMaps and Secrets

- The contents of the Secrets are Base 64 encoded but for ConfigMaps the contents are in plain text
- The secrets can contain binay data and not only plain-text
- Base 64 encoded values allows to store the binary data in YAML or JSON format
- Maximum size of the Secret is 1 MB

```
$ kubectl get secret fortune-https -o yaml
apiVersion: v1
data:
  domain: d3d3LmNsYXNzcGF0aC5pbwo=
  https.cert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCekNDQ...
  https.key: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcE...
kind: Secret
metadata:
  creationTimestamp: "2020-04-06T07:50:27Z"
  name: nginx-https
  namespace: default
  resourceVersion: "1073"
  selfLink: /api/v1/namespaces/default/secrets/nginx-https
  uid: 4356b830-3068-43fe-ae50-2963c97bce39
type: Opaque
```

Using String data field

- To set non binay data using the StringData field
- Write only - Only used to set values
- When retrieving using the `kubectl get -o yaml`, the string data will not be shown.
- Will be shown under the `data` section in base-64 encoded format.

- App need not decode the values but can read the contents or look up from environmental variables and use it directly

```
kind: Secret
apiVersion: v1
stringData:
  key: value
data:
  https.cert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCekNDQ...
  https.key: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcE...
```

Using the Secrets in the pod

Create the configmap with the configuration files

```
kubectl create configmap nginx-conf --from-file=config-maps/config-map-files
```

```
kubectl get configmaps
NAME         DATA   AGE
nginx-conf   2      61s
```

Updated Pod definition

```
# Two containers
# Container one - talkingcow-env
# Container two - Nginx server with https certificate

apiVersion: v1
kind: Pod
metadata:
  name: nginx-https-pod
spec:
  containers:
  - image: classpathio/talking-cow-env
    name: talking-cow-container
    env:
```

```
    - name: INTERVAL
      valueFrom:
        configMapKeyRef:
          name: nginx-conf
          key: sleep-interval
  - image: nginx:alpine
    name: web-server
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    - name: config
      mountPath: /etc/nginx/conf.d
      readOnly: true
    - name: certs
      mountPath: /etc/nginx/certs/
      readOnly: true
    ports:
    - containerPort: 80
    - containerPort: 443
  volumes:
  - name: html
    emptyDir: {}
  - name: config
    configMap:
      name: nginx-conf
      items:
      - key: nginx-configuration.conf
        path: https.conf
  - name: certs
    secret:
      secretName: nginx-https
```

```
kubectl create -f pod-nginx-configmap-https.yaml
```

```
kubectl get po
NAME               READY   STATUS    RESTARTS   AGE
nginx-https-pod    2/2     Running   0          5s
```

```
kubectl exec -it nginx-https-pod -c web-server sh
/ #
```

Verify the Server has picked up the Secret

```
kubectl port-forward fortune-https 8443:443 &
Forwarding from 127.0.0.1:8443 -> 443
Forwarding from [::1]:8443 -> 443
$ curl https://localhost:8443 -k
```

```
curl https://localhost:8443 -k -v
* Rebuilt URL to: https://localhost:8443/
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRE
* successfully set certificate verify locations:
*   CAfile: /etc/pki/tls/certs/ca-bundle.crt
  CApath: none
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: CN=www.classpath.io
*  start date: Apr  6 07:44:46 2020 GMT
*  expire date: Apr  4 07:44:46 2030 GMT
*  issuer: CN=www.classpath.io
*  SSL certificate verify result: self signed certificate (18), continuing
> GET / HTTP/1.1
> Host: localhost:8443
> User-Agent: curl/7.61.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx/1.17.9
< Date: Mon, 06 Apr 2020 15:47:19 GMT
< Content-Type: text/html
< Content-Length: 45
< Last-Modified: Mon, 06 Apr 2020 15:24:52 GMT
< Connection: keep-alive
< ETag: "5e8b49c4-2d"
< Accept-Ranges: bytes
<
<html>
<body>
  <h1>Hello world</h1>
</body>
```

```
kubectl logs nginx-https-pod -c web-server
2020/04/06 14:05:52 [error] 6#6: *1 directory index of "/usr/share/nginx/h
127.0.0.1 - - [06/Apr/2020:14:05:52 +0000] "GET / HTTP/1.1" 403 153 "-" "c
2020/04/06 14:06:30 [error] 6#6: *2 directory index of "/usr/share/nginx/h
127.0.0.1 - - [06/Apr/2020:14:06:30 +0000] "GET / HTTP/1.1" 403 153 "-" "c
127.0.0.1 - - [06/Apr/2020:14:07:19 +0000] "GET / HTTP/1.1" 400 255 "-" "c
127.0.0.1 - - [06/Apr/2020:14:08:44 +0000] "GET / HTTP/1.1" 400 255 "-" "c
```

Exposing Secrets through Environment Variables

Instead of configMapKeyRef use secretKeyRef

```
env:
    - name: FOO_SECRET
      valueFrom:
        secretKeyRef:
          name: fortune-https
          key: foo
```

Exposing secrets via Env variables is not a best practice as it can be logged in the application logs. Also, the child process will inherit all the environment variables of the parent process making the data vulnerable to leak.

> Always use the **`Volume`** instead of environment variables

Image Pull Secrets

- When using K8s to pull images from private repositories
- To pull images from private `dockerhub` registry, there are two steps
  - Create a Secret holding the Dockerhub credentials
  - Reference the secret in the imagePullSecrets field in the mod manifest

1. Creaing a image secret

Instead of `generic` secret use the `docker-registry` Secret

```
$ kubectl create secret docker-registry mydockerhubsecret \
  --docker-username=myusername --docker-password=mypassword \
  --docker-email=my.email@provider.com
```

```
kubectl describe mydockerhubsecret
```

- Contains a single entry called `.dockercfg` which is equivalent to `.dockercfg` inside the home directory, which got created when running the `docker login` command

2. Using the Secret in the Pod definition

```
apiVersion: v1
kind: Pod
metadata:
  name: private-pod
spec:
  imagePullSecrets:
  - name: mydockerhubsecret
  containers:
  - image: username/private:tag
    name: main
```

Deployment

Types of deployment

- Delete all the existing pods and start the new pods
  - brief downtime
  -
- Start all the new pods. Once they are up, delete the old pods
  - Two versions of pods for brief time
  - Consume more resources
  - The new version should not break the schema/contract for the old pods

Deployment using Kubectl

- Old way of deployment
- Now obsolete and should not be used for newer deployment
- Deployment is driven by the client and not driven through the master
- Inconsistent state of Pods and ReplicationControllers on the server, if there is an issue (network failures)
- Its imperative
- The process should be closely monitor till the `kubectl` command is successfully executed.

Deployment using Deployment Resource

- Higher level resource
- On top of ReplicaSet
- Creates a ReplicatSet in the background
- When using deployment, actual pods and created and managed by the Replicasets of Deployment resource and not by the Deployment directly
- Coordiantes between old version of ReplicaSet and new version of ReplicaSet
- There will be no change in the Service.

Creating a Deployment Resource

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
```

Ensure deleting all the ReplicationController

```
$ kubectl delete rc --all
```

Create the deployment

```
$ kubectl create -f kubia-deployment-v1.yaml --record
```

```
Note: Make sure you pass in the record argument
```

Verify the Pre deployment configuration for Pods, RS

Check the deployment status

```
$ kubectl rollout status deployment kubia
deployment kubia successfully rolled out
```

Ensure there are three pods running

```
$ kubectl get po
```

Verify the ReplicaSets

```
$ kubectl get replicasets
```

Updating the Deployment

- Modify the Pod template
- Done declaratively
- K8s will transition the system to the new state.

Deployment Strategies

- `RollingUpdate`
  - Default strategy
  - Removes pod one by one while adding the new pods
  - Application is available throught the deployment process
  - Use this, when the application supports both versions of the app to run parallelly
- `Recreate`
  - Deletes all the old pods at one and creates new pods
  - Use this, when your deployment does not support multiple parallel versions and old ones to be stopped completely before the new ones are deployed.
  - Application will be unavailable for short period of time

Starting the deployment process

```
$ kubectl patch deployment kubia -p '{"spec": {"minReadySeconds": 10}}'
```

```
Note: path command is used to update one or few property of a resource witho
```

Triggerring the Rolling update

- To test the status of deployment, continously run the `curl` command in a terminal.

```
$ while true; do curl http://130.211.109.222; done
```

- Update the deployment to the new version

```
$ kubectl set image deployment kubia nodejs=luksa/kubia:v2
```

Verify the deployment by listing the ReplicaSets

```
$ kubectl get rs
```

Rolling back a deployment

Rolls the deployment to the previous version

```
$ kubectl rollout undo deployment kubia
```

- Rolling the deployment to the specific version
  - Control the revision history with `revisionHistoryLimit` property (default is 2

```
$ kubectl rollout undo deployment kubia --to-revision=1
```

Pausing the Rollout

- Typically done for `canary` release
- One or small fractions of pods will be deployed with the new versions

```
$ kubectl set image deployment kubia nodejs=luksa/kubia:v4

$ kubectl rollout pause deployment kubia
```

Resuming the Rollout

```
$ kubectl rollout resume deployment kubia
```

Kubernetes Architecture

Kubernetes Architecture

Kubernetes cluster is split into two parts

- K8s control plane
- Worker nodes

Control Plane: - Controls and makes the entire cluster function.

Components of Control plane

- Etcd - distributed persistent storage
- API server
- Scheduler
- Controller Manager

Components running on worker nodes

- Kubelet
- K8s service proxy (kube-proxy)
- Container runtime (docker, rkt)

Add on components

- K8s Dashboard
- DNS Server
- Ingress Controller
- Container Network Interface

Checking the status of Control plane

```
kubectl get componentstatuses
NAME                     STATUS    MESSAGE              ERROR
scheduler                Healthy   ok
```

```
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

- K8s system components do not talk to each other
- K8s components only talk to the API server
- API server only talks to the etcd
- None of the components talk to the etcd
- Components modify the cluster state by talking to the API server

Running multiple instances of individual components

- All the components of the worker nodes need to run on the same node
- Components of the Control plane can be split across multiple servers
- More than one instance of Controle plane can be run to ensure HA
- Multiple instance of etcd and API server can be running in parallel on different server
- Only single instance of Scheduler and ControllerManager can be active at any given time keeping others in the stand-by mode

Use of etcd

- All the K8s resources like pods, RC, RS, Secrets, Services are stored in etcd.
- The manifest files survive the API server restarts and failures.
- Its a key-value data store
- Distributed and hence, more than one etcd server can run increasing the availability and performance
- All the components indirectly read and write entries to etcd using API server
- Only API server talks to etcd directly
- When multiple etcd servers are running in a clustered environment, consistency is achieved using consensus algorithm (RAFT)
- Use odd numbers of etcd to set up cluster

K8s stores all the entries of etcd under /registry directory

```
etcdctl ls /registry
```

Use of API server

- Central component which is used by all the components and clients
- Provides CRUD interface for querying and modifying the cluster state in a consistent manner using REST API
- Stores data in the etcd.
- Provides validation of objects
- Handles optimistic locking on resources
- Authentication and Authorization can be configured through plugins
- Admission control plugin
  - AlwaysPullImages - to force to pull the images when pod is created
  - ServiceAccount – to apply default service account, if not specefied
  - ResourceQuota – Ensures pods in a namespace uses as much CPU and memory which is allocated
- Store the metadata in the etcd after validating the resource objects
- Clients watch for changes by opening an HTTP connection to the API server
- Clients will receive a stream of modifications to the watched objects
- Every time there is an update, the clients will be notified with the updated object

Example: client watching for any changes using the watch flag

```
kubectl get pods –watch
```

Scheduler

- Wait for the newly created pod through the API server's watch
- Assign a node to each new pod that does not have a node set
- Update the pod definition through the API server
- The API server then notifies the Kubelet that the pod is scheduled
- Kubelet on the node sees that the pod is scheduled and thus creates and runs the pod's container
- The Kubelet then creates and runs the pod containers
- Selecting the node can be as simple as selecting a random node and can be as complex as selecting through machine learning algorithms

- We can run multiple Schedulers in which case, the `schedulerName` property should be mentioned in each pod specification

Default Scheduling Alorithm

- List all the available node that the pod can scheduled to
- Prioritize the acceptable node and choose the best one
- In case of conflicting prioritized nodes, use the round-robin algorithm

Controller Managers

- API Server just stores the resources in the ETCD
- Scheduler assigns the Node
- Other active components are needed to make sure to attain the desired state specified in the resources deployed through API server
- This job is performed by ControllerManagers
- Resources are the descriptors while the ControllerManager is the worker component that does the actual job.
- Controllers run a reconciliation loop by using the watch mechanism to be notified when there is a change and also perform a re-list operation to ensure no events are missed out
- Controllers do not talk to each other directly and only connects to the API server through `watch` mechanism.

List of Controllers

- ReplicationManager
- ReplicaSet
- DaemonSet
- DeploymentController
- NodeController
- ServiceController
- EndpointController
- NamespaceController
- PersistentVolumeController

Kubelet

- Kubelet and Service Proxy both run on the worker node
- Initially registers with the Node by creating a Node resource on the API server
- Contrinously montiro the API server for pods that have been scheduled to the node and start the containers
- Tells the configured container runtime to pull the image from the registry

Service Proxy

- To ensure that the clients can connect to Services defined through the API server
- When a service is backed by more than on pods, the proxy does the load balancing across those pods

Chain of Events of a Deployment Resource

- Kubectl submits the Deployment resource manifest to API Server through a POST request
- The API Server validates the manifest and stores in `etcd`
- DeploymentController client is notified about the event
- DeploymentController creates a ReplicaSet manifest through API Server
- ReplicatSetController then picks up the ReplicaSet which considers the replica count and manages the next state
- The Scheduler then chooses the best node and assigns the pods the node
- The Kubelet on the worker node then inspects the pod definition and instructs the container runtime to pull the image and run the containers

Securing K8s API Server

- Pods talk to the API server to retrive or change the state of resources which are deployed in the cluster
- To Authenticate with the API server, service accounts are mounted inside the pod

Authentication

- Authentication plugins will be configured on the API server
- Same holds good for Authorization plugins which are configured on the API server
- After the API Server receives a request, it goes through a list of authentication plugins
- The plugin extracts the username, userId and groups that the user belongs to and passes this information back to the API server
- Once the information is retrieved, the remaining authentication plugins are discarded and continues to the authorization phase
- K8s does not store the user and group information on the server.

Types of Authentication plugins

- Client certificate
- Authentication token passed in as HTTP header
- Basic HTTP authentication

Users

- Actual humans
- Pods (processes inside the containers)
- Users are managed by external systems
- Pods use a mechanism called `Service Accounts`
- Service Accounts are managed by Service Accounts resources

Groups

- Both humans and Service Accounts can belong to one or more groups
- Groups are useed to grant permissions to serveral users at once instead of granting permissions at individual level

- Groups are represented by strings
- There are build in groups
  - `system:unauthenticated`
  - `system:authenticated`
  - `system:serviceaccounts`
  - `sysmts:serviceaccounts:namespace`

Service Accounts

- Every Pod is mounted with a secret volume
- The pod authenticates with the API server by sending the contents of the file under ` `var/run/secrets /kubernetest.io/serviceaccount/token`
- API server after successfull authentication allows to perform the operation
- Each pod is associated with a service account which represents the identity of the application
- The token field holds the SA authentication token
- The Authentication plugin authenticates the token, extracts the username, groups and UserId information and passes back to the API Server
- API Server then passes these information to Authorization plugin which determine if the app can perform the operation
- Each pod is associated with only one Service Account in the pod's namespace
- Multiple pods can use the same Service Account
- Each namespace contains a default Service Account

Format of service account username

```
system:serviceaccount:<namespace>:<service account name>
```

```
kubectl get sa
```

> ⓘ sa is the short hand version of Service Account

Service Account are tied to Authorization

- SA can be assigned to the pod in the pod's specification
- If not specified, the pod will use the default service account in the namespace
- different SA can be assigned to different pods

Creating Service Account

- Create additional SA apart from default SA for greater cluster security
- Pods that should not access cluster level resources should run under a constrained SA

```
kubectl create serviceaccount foo
serviceaccount "foo" created
```

```
kubectl describe sa foo
Name:               foo
Namespace:          default
Labels:             <none>

Image pull secrets: <none>

Mountable secrets:  foo-token-qzq7j

Tokens:             foo-token-qzq7j
```

- The above command creates a custom secret and associates with the SA

```
$ kubectl describe secret foo-token-qzq7j
...
ca.crt:         1066 bytes
namespace:      7 bytes
token:          eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

- By default, a pod can mount any Secret it wants.

- Pod’s ServiceAccount can be configured to only allow to mount Secrets that are listed as mountable Secrets on the Service-Account.

Assigning a SA to Pod

- In the Pod manifest set the SA in the` `spec.serviceAccountName` field
- SA can be set when creating the pod.
- SA cannot be changed later

```
apiVersion: v1
kind: Pod
metadata:
  name: curl-custom-sa
spec:
  serviceAccountName: foo
  containers:
  - name: main
    image: tutum/curl
    command: ["sleep", "9999999"]
  - name: ambassador
    image: luksa/kubectl-proxy:1.6.2
```

Confirm that the SA secret is mounted in the Pod

```
$ kubectl exec -it curl-custom-sa -c main cat /var/run/secrets/kubernetes.
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...
```

Securing the cluster with RBAC (Authorization Plugin)

- System wide Plugin
  - Clients send `GET` , `POST` , `PUT` , `DELETE` HTTP request
  - Resources are Pods, Service, Secrets etc
  - Specify the permissions for Standard resources and non-resource URLs
  - RBAC rules are configured throught 4 resources
  - Role and ClusterRoles
  - RoleBinding and ClusterRoleBinding

- Role and RoleBinding are resource level resource
- ClusterRole and ClusterRoleBinding are cluster level resource
- Multiple Roles and RoleBindings can be created in a namespace

1. Create namespaces and run the pods

```
$ kubectl create ns foo
namespace "foo" created

$ kubectl run test --image=luksa/kubectl-proxy -n foo
deployment "test" created

$ kubectl create ns bar
namespace "bar" created

$ kubectl run test --image=luksa/kubectl-proxy -n bar
deployment "test" created
```

2. Fetch the name of the pod

```
$ kubectl get po -n foo
NAME                       READY     STATUS    RESTARTS   AGE
test-145485760-ttq36       1/1       Running   0          1m
```

3. Login to the pod

```
$ kubectl exec -it test-145485760-ttq36 -n foo sh
/ #
```

4. Perform the above steps for the other pod as well

Create a Role resource

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: foo
  name: service-reader
rules:
- apiGroups: [""]
  verbs: ["get", "list"]
  resources: ["services"]
```

> plural form should be used when specifying the resources

```
$ kubectl create -f service-reader.yaml -n foo
role "service-reader" created
```

Using the command directly to create the role

```
$ kubectl create -f service-reader.yaml -n foo
role "service-reader" created
```

Binding the Role to Service Account

- Role defines what actions can be performed
- RoleBinding specefies who can perform
- RoleBinding can bind a role to user, SA or a group
- RoleBinding always referes to a single role
- A Role can bind to multiple subjects (users, groups, sevice accounts)
-

Creating a RoleBinding resource

```
$ kubectl create rolebinding test --role=service-reader --serviceaccount=f
rolebinding "test" created
```

Describe the RoleBinding

```
kubectl get rolebinding test -n foo -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test
  namespace: foo
  ...
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: service-reader
subjects:
- kind: ServiceAccount
  name: default
  namespace: foo
```

Include the SA from another namespace in Rolebinding

Edit the existing Rolebinding

```
$ kubectl edit rolebinding test -n foo
```

Reference another SA in the current RoleBinding

```
subjects:
- kind: ServiceAccount
  name: default
  namespace: bar
```

ClusterRole and ClusterRoleBinding

- Role and RoleBinding are namespaced resource
- Clusterlevel resources are managed by ClusterRole and ClusterRoleBinding
- To manage non-resource URL
- The default SA cannot list the PV, even after the clusterRole has bound to SA

```
kubectl create clusterrole pv-reader --verb=get,list  --resource=persisten
clusterrole "pv-reader" created
```

```
kubectl get clusterrole pv-reader -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pv-reader
  resourceVersion: "39932"
  selfLink: ...
  uid: e9ac1099-30e2-11e7-955c-080027e6b159
rules:
- apiGroups:
  - ""
  resources:
  - persistentvolumes
  verbs:
  - get
  - list
```

```
kubectl create rolebinding pv-test --clusterrole=pv-reader  --serviceaccou
rolebinding "pv-test" created
```

- To access cluster level resources from the service account, bind the ClusterRoleBinding

Delete the RoleBinding

```
$ kubectl delete rolebinding pv-test
```

```
rolebinding "pv-test" deleted
```

Create ClusterRoleBinding

```
$ kubectl create clusterrolebinding pv-test --clusterrole=pv-reader
➥  --serviceaccount=foo:default
clusterrolebinding "pv-test" created
```

Allowing to access non-resource URL's

Predefined ClusterRoles and ClusterRoleBinding

Cluster role system:discovery

```
$ kubectl get clusterrole system:discovery -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: system:discovery
  ...
rules:
- nonResourceURLs:
  - /api
  - /api/*
  - /apis
  - /apis/*
  - /healthz
  - /swaggerapi
  - /swaggerapi/*
  - /version
  verbs:
  - get
```

ClusterRoleBinding for system:discover role

```
$ kubectl get clusterrolebinding system:discovery -o yaml
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRoleBinding
metadata:
  name: system:discovery
  ...
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:discovery
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:authenticated
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:unauthenticated
```

Cluster Role view

```
$ kubectl get clusterrole view -o yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: view
  ...
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - persistentvolumeclaims
  - pods
  - replicationcontrollers
  - replicationcontrollers/scale
  - serviceaccounts
  - services
  verbs:
  - get
  - list
  - watch
...
```

- ***Cluster Role can be bound to ClusterRoleBinding or RoleBinding***

- ClusterRoleBinding bound to ClusterRole, the subjects listed in the binding can access resources across all namespaces
- If RoleBinding is bound to ClusterRole, the subjects listed in the binding can access only namespaced resources

Predefined ClusterRoles and ClusterRoleBindings

```
kubectl get clusterrolebindings
NAME                                           AGE
cluster-admin                                  1d
system:basic-user                              1d
system:controller:attachdetach-controller      1d
...
system:controller:ttl-controller               1d
system:discovery                               1d
system:kube-controller-manager                 1d
system:kube-dns                                1d
system:kube-scheduler                          1d
system:node                                    1d
system:node-proxier                            1d

kubectl get clusterroles
NAME                                           AGE
admin                                          1d
cluster-admin                                  1d
edit                                           1d
system:auth-delegator                          1d
system:basic-user                              1d
system:controller:attachdetach-controller      1d
...
system:controller:ttl-controller               1d
system:discovery                               1d
system:heapster                                1d
system:kube-aggregator                         1d
system:kube-controller-manager                 1d
system:kube-dns                                1d
system:kube-scheduler                          1d
system:node                                    1d
system:node-bootstrapper                       1d
system:node-problem-detector                   1d
system:node-proxier                            1d
system:persistent-volume-provisioner           1d
view                                           1d


```

- Important roles are `view` `edit` `admin` and `cluster-admin`

View

- Reading resources in a namespace
- Except Roles, RoleBindings and secrets

Edit

- Modify resources including reading and modifying secrets
- Forbids reading and modifying Roles and RoleBindings

Admin

- Complete control of resources in the namespace including Roles and RoleBindings
- Except ResourceQuota

Cluster-Admin

- Complete contol of resources including Resource Quota

Advance Scheduling

Taints and Tolerations to repel pods

- Node taints and Pods tolerations
- Used to restrict which pods can use a certain node
- Pod can only be scheduled to a node if it can tolerate the Node taints
- Nodes can have more than 1 taint and Pods can have more than 1 tolreations
- Can be used for Cluster partitioning to dev.preprod etc

```
kubectl describe node master.k8s
Name:          master.k8s
Role:
Labels:        beta.kubernetes.io/arch=amd64
               beta.kubernetes.io/os=linux
               kubernetes.io/hostname=master.k8s
               node-role.kubernetes.io/master=
Annotations:   node.alpha.kubernetes.io/ttl=0
               volumes.kubernetes.io/controller-managed-attach-detach=true
Taints:        node-role.kubernetes.io/master:NoSchedule
...
```

Node Taints

- Contains a Key, Value and Effect
- Represented by `<key>=<value>:<effect>`
- Three kinds of effects
  - NoSchedule - Pods would not be scheduled if they cannot tolerate the taint
  - PreferNoSchedule - Softer version of NoSchedule - Avoind scheduling pods if they cannot tolerat the taint
  - NoExecute - The running pods are also evicted if they cannot tolerate the taints

Pod Tolearation

```
kubectl describe po kube-proxy-80wqm -n kube-system
...
Tolerations:    node-role.kubernetes.io/master=:NoSchedule
```

```
                node.alpha.kubernetes.io/notReady=:Exists:NoExecute
                node.alpha.kubernetes.io/unreachable=:Exists:NoExecute
...
```

Custom Taints

Use case - Creating a Taint on production Node to avoid pre-prod pods to be scheduled on Prod Node

```
kubectl taint node node1.k8s node-type=production:NoSchedule
node "node1.k8s" tainted
```

```
kubectl run test --image busybox --replicas 5 -- sleep 99999
deployment "test" created

kubectl get po -o wide
NAME                READY  STATUS    RESTARTS   AGE   IP          NODE
test-196686-46ngl   1/1    Running   0          12s   10.47.0.1   node2.k8
test-196686-73p89   1/1    Running   0          12s   10.47.0.7   node2.k8
test-196686-77280   1/1    Running   0          12s   10.47.0.6   node2.k8
test-196686-h9m8f   1/1    Running   0          12s   10.47.0.5   node2.k8
test-196686-p85ll   1/1    Running   0          12s   10.47.0.4   node2.k8
```

Adding Toleartions to Pods

Pod Manifest

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: prod
spec:
  replicas: 5
  template:
    spec:
      ...
      tolerations:
      - key: node-type
```

```
        operator: Equal
        value: production
        effect: NoSchedule
```

```
$ kubectl get po -o wide
NAME                 READY   STATUS    RESTARTS   AGE   IP          NODE
prod-350605-1ph5h    0/1     Running   0          16s   10.44.0.3   node1.k8
prod-350605-ctqcr    1/1     Running   0          16s   10.47.0.4   node2.k8
prod-350605-f7pcc    0/1     Running   0          17s   10.44.0.6   node1.k8
prod-350605-k7c8g    1/1     Running   0          17s   10.47.0.9   node2.k8
prod-350605-rp1nv    0/1     Running   0          17s   10.44.0.4   node1.k8
```

Node Affinity to attract Pods to certain Nodes

- Allows to tell K8s to schedule pods only to specific subset of nodes

Node Affinity vs Node Selectors

Node Selector	Node Affinity
Old way of affinity mechanism	Newer way of specifying Node Affinity
Node has to include all the labels specified to be eligible to become target for the pod	Can give hard requirements or preferences
Will eventually be deprecated	New standard going further

Important Node labels

- `failure-domain.beta.kubernetes.io/region` - specifies the geographical region
- `failure-domain.beta.kubernetes.io/zone` - specifies the zone
- `kubernetes.io/hostname` - specifies the hostname

Specifying the Node Affinity in the Pod manifest

More expressive than NodeSelector

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: kubia-gpu
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: gpu
            operator: In
            values:
            - "true"
```

- `requiredDuringScheduling` - Rules defined under this specify the labels the node must have for the pod to be scheduled
- `...IgnoredDuringScheduling` - Rules defined under this will be ignored during scheduling and dont affect the pods already executing on the node

Prioritizing the nodes while scheduling a Pod

- Node affinity allows to specify which node the scheduler should prefer when scheduling a pod using the `preferredDuringSchedulingIgnoredDuringExecution` field

Label the Nodes appropriately

- Based on Availability zone

```
kubectl label node node1.k8s availability-zone=zone1
node "node1.k8s" labeled

kubectl label node node1.k8s share-type=dedicated
node "node1.k8s" labeled

kubectl label node node2.k8s availability-zone=zone2
node "node2.k8s" labeled

kubectl label node node2.k8s share-type=shared
```

```
node "node2.k8s" labeled

kubectl get node -L availability-zone -L share-type
NAME         STATUS    AGE       VERSION   AVAILABILITY-ZONE   SHARE-TYPE
master.k8s   Ready     4d        v1.6.4    <none>              <none>
node1.k8s    Ready     4d        v1.6.4    zone1               dedicated
node2.k8s    Ready     4d        v1.6.4    zone2               shared
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: pref
spec:
  template:
    ...
    spec:
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 80
            preference:
              matchExpressions:
              - key: availability-zone
                operator: In
                values:
                - zone1
          - weight: 20
            preference:
              matchExpressions:
              - key: share-type
                operator: In
                values:
                - dedicated
      ...
```

```
kubectl get po -o wide
NAME                READY   STATUS    RESTARTS  AGE   IP          NODE
pref-607515-1rnwv   1/1     Running   0         4m    10.47.0.1   node2.k8
pref-607515-27wp0   1/1     Running   0         4m    10.44.0.8   node1.k8
pref-607515-5xd0z   1/1     Running   0         4m    10.44.0.5   node1.k8
pref-607515-jx9wt   1/1     Running   0         4m    10.44.0.4   node1.k8
pref-607515-mlgqm   1/1     Running   0         4m    10.44.0.6   node1.k8
```

Colocating Pods with affinity and Anti affinity

- To definit the affinity between pods

Deploy one pod

```
$ kubectl run backend -l app=backend --image busybox -- sleep 999999
deployment "backend" created
```

Front end pod definition

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 5
  template:
    ...
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchLabels:
                app: backend
      ...
```

Deploying pods with Affinity

```
$ kubectl get po -o wide
NAME                   READY  STATUS   RESTARTS  AGE  IP         NODE
backend-257820-qhqj6   1/1    Running  0         8m   10.47.0.1  node2.k8s
```

```
kubectl create -f frontend-podaffinity-host.yaml
```

```
deployment "frontend" created

kubectl get po -o wide
NAME                    READY   STATUS    RESTARTS   AGE   IP          NODE
backend-257820-qhqj6    1/1     Running   0          8m    10.47.0.1   node2.k8
frontend-121895-2c1ts   1/1     Running   0          13s   10.47.0.6   node2.k8
frontend-121895-776m7   1/1     Running   0          13s   10.47.0.4   node2.k8
frontend-121895-7ffsm   1/1     Running   0          13s   10.47.0.8   node2.k8
frontend-121895-fpgm6   1/1     Running   0          13s   10.47.0.7   node2.k8
frontend-121895-vb9ll   1/1     Running   0          13s   10.47.0.5   node2.k8
```

Co-locating pods in the same Availability Zone / Region

- Set the toplogyKey to `failure-domain.beta.kubernetes.io/zone`
- Set the toplogyKey to `failure-domain.beta.kubernetes.io/region`

Expressing Affinity rather than hard requirements

- Expressing preferences instead of hard requirements
- The scheduler will honor if there is provision else will select a different node

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 5
  template:
    ...
    spec:
      affinity:
        podAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 80
            podAffinityTerm:
              topologyKey: kubernetes.io/hostname
              labelSelector:
                matchLabels:
                  app: backend
      containers: ...
```

Scheduling the pods with Anti-affinity

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 5
  template:
    metadata:
      labels:
        app: frontend
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchLabels:
                app: frontend
      containers: ...
```

```
$ kubectl get po -l app=frontend -o wide
NAME                    READY  STATUS   RESTARTS  AGE  IP         NODE
frontend-286632-0lffz   0/1    Pending  0         1m   <none>
frontend-286632-2rkcz   1/1    Running  0         1m   10.47.0.1  node2.k8
frontend-286632-4nwhp   0/1    Pending  0         1m   <none>
frontend-286632-h4686   0/1    Pending  0         1m   <none>
frontend-286632-st222   1/1    Running  0         1m   10.44.0.4  node1.k8
```

Appendix-A

Ingress Controller Setup

1. Instructions for setup - https://docs.nginx.com/nginx-ingress-controller/installation/installation-with-manifests/
2.

a117bf1ffc495407cb9dbc76bed4ef73-79 2800871.ap-south-1.elb.amazonaws.com 13.234.250.206 13.126.122.135

```
curl –resolve cafe.example.com:$IC_HTTPS_PORT:$IC_IP https://cafe.example.com:$IC_HTTPS_PORT/tea –insecure
```

```
curl –resolve hostnameservice.classpath.com:443:13.126.122.135 https://hostnameservice.classpath.com:443/ –insecure
```

```
 curl –resolve cafe.example.com:443:13.126.122.135 https://cafe.example.com:443/coffee –insecure
```

```
 curl –resolve hostnameservice.classpath.com:80:13.126.101.254 http://hostnameservice.classpath.com:80/ –insecure
```