Lab 5 – Manage Kubernetes Containers

Introduction

In this lab, you will understand and work with **core concepts of kubernetes**, Includes PODs, Labels, Replica Sets, Replication Controllers, Deployments, Services, Daemons and Namespaces.

1. Namespaces

1.1 Login to **pod0-master.origin.com** host as **root** user, if not already:

Copy

```
ssh root@pod0-master.origin.com
```

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces. Within the same namespace, kubernetes objects name should be unique. Different objects in different namespaces may have the same name.

Kubernetes comes with three initial namespaces

- **default:** the default namespace for objects with no other namespace
- **kube-system** the namespace for objects created by the kubernetes system
- **kube-public** The namespace is created automatically and readable by all users.

1.2 To get namespaces, Verify the default existing namespaces

Copy

```
kubectl get namespaces
```

Output:

```
NAME          STATUS    AGE

default       Active    9h
```

```
kube-public   Active    9h

kube-system   Active    9h
```

1.3 Create a new namespaces as “**kube-core**”

Copy

```
kubectl create namespace kube-core
```

Output:

```
namespace "kube-core" created
```

1.4 List all namespaces

Copy

```
kubectl get namespaces
```

Output:

```
NAME          STATUS    AGE

default       Active    9h

kube-core   Active    9h

kube-public   Active    9h

kube-system   Active    9h
```

1.5 Describe the specified namespace:

Copy

```
kubectl describe namespace kube-core
```

Output:

```
Name:         kube-core

Labels:       <none>

Annotations:  <none>

Status:       Active


No resource quota.


No resource limits.
```

1.6 Verify the labels for namespaces

Copy

```
kubectl get namespaces --show-labels
```

Output:

```
NAME          STATUS    AGE       LABELS

default       Active    34m       <none>

kube-core     Active    21s       <none>

kube-public   Active    34m       <none>

kube-system   Active    34m       <none>
```

1.7 Verify the contexts for the current namespaces

Copy

```
kubectl config view
```

Output:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://10.1.64.178:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

1.8 Set the context for **kube-core** namespaces

Copy

```
kubectl config set-context production --namespace=kube-core --cluster=
kubernetes --user=admin-user
```

Output:

```
Context "production" created.
```

1.9 List the available contexts

Copy

```
kubectl config get-contexts
```

Output:

```
CURRENT   NAME                          CLUSTER      AUTHINFO
NAMESPACE

*         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin

          production                    kubernetes   admin-user
kube-core
```

1.10 Switch to the newly created context

Copy

```
kubectl config use-context production
```

Output:

```
Switched to context "production".
```

1.11 Verify the current context in use

Copy

```
kubectl config current-context
```

Output:

```
production
```

1.12 Display merged kubeconfig settings

Copy

```
kubectl config view
```

Output:

```
apiVersion: v1

clusters:

- cluster:

    certificate-authority-data: REDACTED

    server: https://10.1.64.178:6443

  name: kubernetes

contexts:

- context:

    cluster: kubernetes

    user: kubernetes-admin

  name: kubernetes-admin@kubernetes

- context:

    cluster: kubernetes
```

```
    namespace: kube-core
    user: admin-user
  name: production
current-context: production
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

1.13 Switch back to the default namespaces:

Copy

```
kubectl config use-context kubernetes-admin@kubernetes
```

Output:

```
Switched to context "kubernetes-admin@kubernetes".
```

2. User in Kubernetes

In Kubernetes there are **two categories** of users:

- Service accounts managed by Kubernetes
- Normal users – Outside of kubernetes

An admin distributing **private keys**, a user store like Keystone or Google Accounts, even a file with a list of usernames and passwords.

Service accounts are users managed by the Kubernetes API. They are bound to **specific namespaces**, and created automatically by the API server or manually through API calls. Service accounts are tied to a set of credentials stored as Secrets, which are mounted into pods allowing in-cluster processes to talk to the Kubernetes API.

a. Create Service Account

2.1 We are creating a Service Account with the name admin-user in namespace kube-core.

Copy

```
cat > credadmin.yaml <<EOF

apiVersion: v1

kind: ServiceAccount

metadata:

  name: admin-user

  namespace: kube-core

EOF
```

2.2 Create the service account for **kube-core** namespace

Copy

```
kubectl create -f  credadmin.yaml
```

Output:

```
serviceaccount "admin-user" created
```

b. Create ClusterRoleBinding

2.3 In most cases after provisioning our cluster using **kops** or **kubeadm** or any other popular tool admin Role already exists in the cluster. We can use it and create only **RoleBinding** for our **ServiceAccount**.

Copy

```
cat > clusterrolebinding.yaml <<EOF

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

  name: admin-user

roleRef:

  apiGroup: rbac.authorization.k8s.io

  kind: ClusterRole

  name: cluster-admin

subjects:

- kind: ServiceAccount

  name: admin-user

  namespace: kube-core

EOF
```

2.4 Create a **ClusterRoleBinding** for service account

Copy

```
kubectl create -f clusterrolebinding.yaml
```

Output:

```
clusterrolebinding.rbac.authorization.k8s.io "admin-user" created
```

c. Cleanup Namespaces

2.5 Delete the context

Copy

```
kubectl config delete-context production
```

Output:

```
deleted context production from /root/.kube/config
```

2.6 Delete the namespaces

Copy

```
kubectl delete namespaces kube-core
```

Output:

```
namespace "kube-core" deleted
```

3. PODs

In Kubernetes, a group of one or more containers is called a **pod**. Containers in a pod are deployed together, and are **started**, **stopped**, and **replicated** as a group.

3.1 Create a nginx pod, name as "**pod-nginx.yaml**" file.

Copy

```
cat > pod-nginx.yaml <<EOF

apiVersion: v1

kind: Pod
```

```
metadata:

  name: mynginx

  namespace: default

  labels:

    run: nginx

spec:

  containers:

  - name: mynginx

    image: nginx:latest

    ports:

    - containerPort: 80

EOF
```

A pod definition is a declaration of a **desired state**.

- **Desired state** is a very important concept in the Kubernetes model. Many things present a desired state to the system, and it is Kubernetes' responsibility to make sure that the current state matches the desired state.

3.2 Create a pod containing an nginx server from the **pod-nginx.yaml** file

Copy

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

Output:

```
pod "mynginx" created
```

3.3 List all pods are running state:

Copy

```
kubectl get pods -o wide
```

Output:

```
NAME      READY     STATUS    RESTARTS   AGE       IP           NODE

mynginx   1/1       Running   0          2m        20.1.1.3   pod0-nod
e.onecloud.com
```

Note: Wait till the pod status changes to **"running"** status. **Re-run** the above command to verify of it.

A pod can be in one of the following phases:

- **Pending:** the API Server has created a pod resource and stored it in etcd, but the pod has not been scheduled yet, nor have container images been pulled from the registry.
- **Running:** the pod has been scheduled to a node and all containers have been created by the kubelet.
- **Succeeded:** all containers in the pod have terminated successfully and will not be restarted.
- **Failed:** all containers in the pod have terminated and, at least one container has terminated in failure.
- **Unknown:** The API Server was unable to query the state of the pod, typically due to an error in communicating with the kubelet.

3.4 Describe the created **nginx** pod in detailed:

Copy

```
kubectl describe pod mynginx
```

Output:

```
Name:         mynginx

Namespace:    default

Node:         pod30-node.onecloud.com/10.1.64.178
```

```
Start Time:   Sun, 22 Apr 2018 08:33:21 +0000

Labels:       run=nginx

Annotations:  <none>

Status:       Running

IP:           20.1.1.3

Containers:

....

....

Events:

  Type    Reason                 Age   From
Message

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

  Normal  Scheduled              1m    default-scheduler
Successfully assigned mynginx to pod30-node.onecloud.com

  Normal  SuccessfulMountVolume  1m    kubelet, pod30-node.onecloud.co
m  MountVolume.SetUp succeeded for volume "default-token-nf7p8"

  Normal  Pulling                1m    kubelet, pod30-node.onecloud.co
m  pulling image "nginx:latest"

  Normal  Pulled                 48s   kubelet, pod30-node.onecloud.co
m  Successfully pulled image "nginx:latest"

  Normal  Created                48s   kubelet, pod30-node.onecloud.co
m  Created container

  Normal  Started                48s   kubelet, pod30-node.onecloud.co
m  Started container
```

4. Labels

In Kubernetes, labels are a system to organize **objects** into **groups**. Labels are **key-value pairs** that are attached to each object.

Label selectors can be passed along with a request to the apiserver to retrieve a list of objects which match that label selector.

4.1 Create a label to a pod, add a labels section under metadata in the pod definition:

Copy

```
cat pod-nginx.yaml
```

4.2 Label the **mynginx** pod

Copy

```
kubectl label pod mynginx type=webserver
```

Output:

```
pod "mynginx" labeled
```

4.3 List the pods based on specified label

Copy

```
kubectl get pods -l type=webserver
```

Output:

```
NAME      READY     STATUS    RESTARTS   AGE

mynginx   1/1       Running   0          4m
```

4.4 Labels can be applied not only to pods but also to other Kubernetes objects like nodes.

Copy

```
kubectl get nodes
```

Output:

```
NAME                         STATUS    ROLES     AGE       VERSION
pod0-master.onecloud.com   Ready     master    8h        v1.10.0
pod0-node.onecloud.com     Ready     node      8h        v1.10.0
```

4.5 Apply the label to **pod0-master.onecloud.com** node

Copy

```
kubectl label node pod0-master.onecloud.com  rack=rack01
```

Output:

```
node "pod0-master.onecloud.com" labeled
```

4.6 Verify that the **master** node is labeled

Copy

```
kubectl get nodes -l rack=rack01
```

Output:

```
NAME                         STATUS    ROLES     AGE       VERSION
pod0-master.onecloud.com   Ready     master    8h        v1.10.0
```

4.7 Apply label to **pod0-master.onecloud.com** node

Copy

```
kubectl label node pod0-master.onecloud.com rack=rack01
```

Output:

```
node "pod0-master.onecloud.com" labeled
```

4.8 Verify that the worker node is labeled

Copy

```
kubectl get nodes -l rack=rack01
```

Output:

```
NAME                         STATUS    ROLES     AGE       VERSION

pod38-node.onecloud.com   Ready     master    8h        v1.10.0

pod38-node.onecloud.com     Ready     node      8h        v1.10.0
```

Labels are also used as selector for services and deployments.

5. Replica Controllers

A Replica Controller ensures that a specified number of pod replicas are running at any one time. In other words, a Replica Controller makes sure that a pod or homogeneous set of pods are always up and available. If there are too many pods, it will kill some. If there are too few, it will start more. Unlike manually created pods, the pods maintained by a Replica Controller are automatically replaced if they fail, get deleted, or are terminated.

A Replica Controller configuration consists of:

- The number of replicas desired
- The pod definition
- The selector to bind the managed pod

A selector is a label assigned to the pods that are managed by the replica controller. Labels are included in the pod definition that the replica controller instantiates. The replica controller uses the selector to determine how many instances of the pod are already running in order to adjust as needed.

5.1 Create a replica controller with **replica** of 3 for our **nginx** pod, and name as "**nginx-rc.yaml**" file:

Copy

```
cat > nginx-rc.yaml <<EOF

---

apiVersion: v1

kind: ReplicationController

metadata:

  name: nginx-rc

  namespace: default

spec:

  replicas: 3

  selector:

    run: nginx

  template:

    metadata:

      name: nginx-rc

      labels:

        run: nginx

    spec:

      containers:

      - name: nginx-rc

        image: nginx:latest
```

```
        ports:
        - containerPort: 80
EOF
```

5.2 Create a replica controller using **nginx-rc.yaml** file:

Copy

```
kubectl create -f nginx-rc.yaml
```

Output:

```
replicationcontroller "nginx-rc" created
```

5.3 List and describe a replica controller

Copy

```
kubectl get rc
```

Output:

```
NAME       DESIRED   CURRENT   READY     AGE
nginx-rc     3         3         3         1m
```

Note: Wait till READY count changes to "**3**".

5.4 Describe the replica controller which you just created:

Copy

```
kubectl describe rc nginx-rc
```

Output:

```
Name:         nginx-rc
```

```
Namespace:    default

Selector:     run=nginx

Labels:       run=nginx

Annotations:  <none>

Replicas:     3 current / 3 desired

Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed

...

...

Events:

  Type    Reason            Age   From                    Message

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

  Normal  SuccessfulCreate  1m    replication-controller  Created pod:
nginx-rc-zpgwx

  Normal  SuccessfulCreate  1m    replication-controller  Created pod:
nginx-rc-zc6g4
```

5.5 Verify the pods are in running state:

Copy

```
watch kubectl get pods
```

Output:

```
NAME             READY     STATUS    RESTARTS   AGE

mynginx          1/1       Running   0          7m

nginx-rc-zc6g4   1/1       Running   0          2m
```

```
nginx-rc-zpgwx   1/1       Running   0          2m
```

Note: Wait for couple of minutes to get nginx-rc-*** pods to create and READY as **Running** and then press **<ctrl+c>** to interrupt.

5.6 The Replica Controller makes it easy to scale the number of replicas up or down, either manually or by an auto-scaling control agent, by simply updating the replicas field.

Copy

```
kubectl scale rc nginx-rc --replicas=0
```

Output:

```
replicationcontroller "nginx-rc" scaled
```

5.7 Verify that replica controller is scaled **down to 0**

Copy

```
kubectl get rc nginx-rc
```

Output:

```
NAME       DESIRED   CURRENT   READY     AGE

nginx-rc     0         0         0         7m
```

5.8 Scale up the replica controller to create new pods

Copy

```
kubectl scale rc nginx-rc --replicas=4
```

Output:

```
replicationcontroller "nginx-rc" scaled
```

5.9 Verify that **nginx-rc** pods **scaled** to **4**.

Copy

```
kubectl get rc nginx-rc
```

Output:

```
NAME       DESIRED   CURRENT   READY     AGE

nginx-rc     4         4         4         9m
```

Note: Wait till READY count changes to "**4**".

Also in case of failure of a node, the replica controller takes care of keep the same number of pods by scheduling the containers running on the failed node to the remaining nodes in the cluster.

5.10 Let us delete one of the pod from the list of running, and you notice the new pod will be created immediately.

Copy

```
kubectl get pods
```

Output:

```
NAME             READY     STATUS    RESTARTS   AGE

nginx-rc-fxq2j   1/1       Running   0          2m

nginx-rc-hfgvr   1/1       Running   0          2m

nginx-rc-rdkrq   1/1       Running   0          2m

nginx-rc-rn8rf   1/1       Running   0          2m
```

Note: Wait for couple of minutes to get pods to create and **Running** in READY state.

Copy

```
kubectl delete pod `kubectl get pods | grep nginx-rc | awk '{print $1}
' | head -1`
```

Output:

```
pod "nginx-rc-fxq2j" deleted
```

5.11 Verify that "**nginx-rc-******", pod is deleted and getting recreated

Copy

```
kubectl get pods
```

Output:

```
NAME             READY     STATUS              RESTARTS   AGE

nginx-rc-786x7   0/1       ContainerCreating   0          17s

nginx-rc-hfgvr   1/1       Running             0          3m

nginx-rc-rdkrq   1/1       Running             0          3m

nginx-rc-rn8rf   1/1       Running             0          3m
```

5.12 Run the below command to delete a replica controller, which deletes all of the pods created.

Copy

```
kubectl delete rc/nginx-rc
```

Output:

```
replicationcontroller "nginx-rc" deleted
```

5.13 Verify the pods are deleted

Copy

```
kubectl get pods
```

Output:

```
No resources found.
```

5.14 Deleting a replica controller deletes all pods managed by that replica. But, because pods created by a replication controller are not actually an integral part of the replication controller, but only managed by it, we can delete only the replication controller and leave the pods running.

Copy

```
kubectl create -f nginx-rc.yaml
```

Output:

```
replicationcontroller "nginx-rc" created
```

5.15 Verify the pod status

Copy

```
kubectl get pods
```

Output:

```
NAME             READY     STATUS    RESTARTS   AGE

nginx-rc-hgqv9   1/1       Running   0          1m

nginx-rc-jqh8f   1/1       Running   0          1m

nginx-rc-tk7nc   1/1       Running   0          1m
```

Note: Wait till the pod ready changes to **“running”**. **Re-run** the above command to verify of it.

5.16 Delete the replication controller with the option **cascade**, which retains the pods by just deleting replication controller.

Copy

```
kubectl delete rc/nginx-rc --cascade=false
```

Output:

```
replicationcontroller "nginx-rc" deleted
```

5.17 Verify the rc pod status

Copy

```
kubectl get pods
```

Output:

```
NAME             READY     STATUS    RESTARTS   AGE

nginx-rc-hgqv9   1/1       Running   0          1m

nginx-rc-jqh8f   1/1       Running   0          1m

nginx-rc-tk7nc   1/1       Running   0          1m
```

5.18 Verify the replication controllers are deleted.

Copy

```
kubectl get rc
```

Output:

```
No resources found.
```

Now there is nothing managing pods, but we can always create a new replication controller with the proper label selector and make them managed again.

5.19 Delete all the pods

Copy

```
kubectl delete pod --all
```

Output:

```
pod "nginx-rc-hgqv9" deleted

pod "nginx-rc-jqh8f" deleted

pod "nginx-rc-tk7nc" deleted
```

6. Replica Sets

A Replica Set object is very similar to the replica controller object we practiced before.

6.1 Create a file "**nginx-rs.yaml**" configuration file defines a replica set for the nginx pod

Copy

```
cat > nginx-rs.yaml <<EOF

---

apiVersion: extensions/v1beta1

kind: ReplicaSet

metadata:

  labels:

    run: nginx

  namespace:

  name: nginx-rs

spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      run: nginx
  template:
    metadata:
      labels:
        run: nginx
    spec:
      containers:
      - image: nginx:1.12
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
EOF
```

6.2 Create a replica set using **nginx-rs.yaml** file

Copy

```
kubectl create -f nginx-rs.yaml
```

Output:

```
replicaset.extensions "nginx-rs" created
```

6.3 Verify the replicaset status

Copy

```
kubectl get rs -o wide
```

Output:

```
NAME       DESIRED   CURRENT   READY     AGE       CONTAINERS   IMAGES
SELECTOR

nginx-rs   3         3         3         1m        nginx        nginx:
1.12   run=nginx
```

Note: Wait till READY count changes to "**3**".

A replica set behaves exactly like a replication controller, but it has more powerful pod selectors. Whereas a replication controller label selector only allows matching pods that include a certain label, the replica set pod selector also allows matching pods that lack a certain label or pods that include a certain label key, regardless of its value.

6.4 Verify that the status for nginx replicas

Copy

```
kubectl get pods
```

Sample output:

```
NAME             READY     STATUS    RESTARTS   AGE

nginx-rs-26kw5   1/1       Running   0          6m

nginx-rs-dszmn   1/1       Running   0          6m

nginx-rs-s5cwd   1/1       Running   0          6m
```

6.5 Delete all the pods

Copy

```
kubectl delete pod --all
```

Sample output:

```
pod "nginx-rs-26kw5" deleted

pod "nginx-rs-dszmn" deleted

pod "nginx-rs-s5cwd" deleted
```

6.6 Delete the replicaset

Copy

```
kubectl delete rs nginx-rs
```

Output:

```
replicaset.extensions "nginx-rs" deleted
```

7. Deployments

A Deployment provides declarative updates for pods and replicas. The Deployment object defines the strategy for transitioning between deployments of the same application.

There are basically two ways of updating an application:

- delete all existing pods first and then start the new ones or
- start new ones and once they are up, delete the old ones

The latter, can be done with two different approach:

- add all the new pods and then deleting all the old ones at once
- add new pods at time and then removing old ones one by one

7.1 Create a deployment for our **nginx webserver**, name as **nginx-deploy.yaml** file .

Copy

```
cat > nginx-deploy.yaml <<EOF

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

  generation: 1

  labels:

    run: nginx

  name: nginx

  namespace: default

spec:

  replicas: 2

  selector:

    matchLabels:

      run: nginx

  strategy:

    rollingUpdate:

      maxSurge: 1

      maxUnavailable: 1

    type: RollingUpdate

  template:
```

```
    metadata:
      labels:
        run: nginx
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
EOF
```

7.2 Create a deployment for **nginx-webserver** using **nginx-deploy.yaml** file

Copy

```
kubectl create -f nginx-deploy.yaml
```

Output:

```
deployment.extensions "nginx" created
```

7.3 The deployment creates the following objects

Copy

```
kubectl get all -l run=nginx -o wide
```

Output:

```
NAME                     READY     STATUS              RESTARTS   AGE
IP        NODE

nginx-6bfb654d7c-kglvd   0/1       ContainerCreating   0          16s
<none>    pod0-master.onecloud.com

nginx-6bfb654d7c-s8k5f   0/1       ContainerCreating   0          16s
<none>    pod0-master.onecloud.com



NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE       CONTA
INERS   IMAGES         SELECTOR

nginx     2         2         2            0           16s       nginx
nginx:latest   run=nginx



NAME               DESIRED   CURRENT   READY     AGE       CONTAINERS
IMAGES         SELECTOR

nginx-6bfb654d7c   2         2         0         16s       nginx
nginx:latest   pod-template-hash=2696210837,run=nginx
```

7.4 A deployment, can be scaled up or down

Copy

```
kubectl scale deploy nginx --replicas=3
```

Output:

```
deployment.extensions "nginx" scaled
```

7.5 Verify the deploy nginx status

Copy

```
kubectl get deploy nginx
```

Output:

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE

nginx     3         3         3            3           8m
```

In a deploy, pods are always controlled by the replica set. However, because the replica set is controlled by the deploy, if we try to scale the replica set instead of the deploy, the deploy will take priority and the number of pods will be reported to the number requested by the deploy.

7.6 Try to scale up the replica set from the previous example to have 5 replicas

Copy

```
replica_set=`kubectl get rs | grep nginx | awk '{print $1}'`

kubectl scale rs $replica_set --replicas=5
```

Output:

```
replicaset.extensions "nginx-6bfb654d7c" scaled
```

we see the number of pod scaled to 5, according to the request to scale the replica set to 5 pod. After few seconds, the deploy will take priority and remove all new pod created by the scaling the replica set because the desired stae, as specified by the deploy is to 3 pods.

7.7 Verify that created pods status

Copy

```
watch kubectl get pods
```

Output:

```
NAME                     READY     STATUS    RESTARTS   AGE
nginx-6bfb654d7c-kglvd   1/1       Running   0          8m
nginx-6bfb654d7c-s8k5f   1/1       Running   0          8m
nginx-6bfb654d7c-x5k72   1/1       Running   0          8m
```

Note: Wait for couple of minutes to get pods to create and running on **READY** state and then press **<ctrl+c>** to interrupt.

7.8 A deployment also defines the strategy for updates pods

```
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
```

In the snippet above, we set the update strategy as rolling update.

7.9 Update the pods with a different version of **nginx** image

Copy

```
kubectl set image deploy nginx nginx=nginx:1.13
```

Output:

```
deployment.apps "nginx" image updated
```

7.10 Rollout the deployed nginx status

Copy

```
kubectl rollout status deploy nginx
```

Output:

```
Waiting for rollout to finish: 2 out of 3 new replicas have been updat
ed...

Waiting for rollout to finish: 2 out of 3 new replicas have been updat
ed...

...

...

Waiting for rollout to finish: 1 old replicas are pending termination.
..

...

Waiting for rollout to finish: 2 of 3 updated replicas are available..
.

deployment "nginx" successfully rolled out
```

Note: Wait for couple of minutes to rolled out successfully.

7.11 Check the status of the deploy

Copy

```
kubectl get all -l run=nginx -o wide
```

Output:

```
NAME                     READY     STATUS    RESTARTS   AGE       IP
NODE

nginx-59dc74b9-bw5t9   1/1       Running   0          3m        10.244
.1.16   pod0-node.onecloud.com
```

```
nginx-59dc74b9-sv29f   1/1       Running   0          2m        10.244
.1.17   pod0-node.onecloud.com

nginx-59dc74b9-xbc96   1/1       Running   0          3m        10.244
.0.10   pod0-node.onecloud.com



NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE       CONTA
INERS   IMAGES       SELECTOR

nginx     3         3         3            3           12m       nginx
nginx:1.13   run=nginx



NAME               DESIRED   CURRENT   READY     AGE       CONTAINERS
IMAGES         SELECTOR

nginx-59dc74b9     3         3         3         3m        nginx
nginx:1.13     pod-template-hash=15873065,run=nginx

nginx-6bfb654d7c   0         0         0         12m       nginx
nginx:latest   pod-template-hash=2696210837,run=nginx
```

Now there is a new replica set now taking control of the pods. This replica set control new pods having image nginx:1.13. The old replica set is still there and can be used in case of downgrade.

7.12 Verify the replicasets status

Copy

```
kubectl get rs
```

Output:

```
NAME               DESIRED   CURRENT   READY     AGE

nginx-59dc74b9     3         3         3         3m
```

```
nginx-6bfb654d7c   0         0         0         13m
```

7.13 Verify the pods status

Copy

```
kubectl get pods
```

Output:

```
NAME                    READY     STATUS    RESTARTS   AGE

nginx-59dc74b9-bw5t9    1/1       Running   0          4m

nginx-59dc74b9-sv29f    1/1       Running   0          3m

nginx-59dc74b9-xbc96    1/1       Running   0          4m
```

8. Services

A Kubernetes Service is an abstraction which defines a logical set of pods and a policy by which to access them. The set of pods targeted by a Service is usually determined by a label selector. Kubernetes offers a simple Endpoints API that is updated whenever the set of pods in a service changes.

8.1 Create a service for our **nginx webserver**, name as "**nginx-service.yaml**" file

Copy

```
cat > nginx-service.yaml <<EOF

apiVersion: v1

kind: Service

metadata:

  name: nginx

  labels:
```

```
    run: nginx
spec:
  selector:
    run: nginx
  ports:
  - protocol: TCP
    port: 8000
    targetPort: 80
  type: ClusterIP
EOF
```

8.2 Create a service for **nginx webserver** service, using **nginx-service.yaml**

Copy

```
kubectl create -f nginx-service.yaml
```

Output:

```
service "nginx" created
```

8.3 Verify the status for nginx service

Copy

```
kubectl get service -l run=nginx
```

Output:

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

```
nginx     ClusterIP   10.103.148.246   <none>        8000/TCP   7s
```

8.4 Describe the nginx service

Copy

```
kubectl describe service nginx
```

Output:

```
Name:              nginx
Namespace:         default
Labels:            run=nginx
Annotations:       <none>
Selector:          run=nginx
Type:              ClusterIP
IP:                10.103.148.246
Port:              <unset>  8000/TCP
TargetPort:        80/TCP
Endpoints:         10.244.0.10:80,10.244.1.16:80,10.244.1.17:80
Session Affinity:  None
Events:            <none>
```

The above service is associated to our previous nginx pods. Pay attention to the service selector run=nginx field. It tells Kubernetes that all pods with the label run=nginx are associated to this service, and should have traffic distributed amongest them. In other words, the service provides an abstraction layer, and it is the input point to reach all of the associated pods.

Pods can be added to the service arbitrarily. Make sure that the label run=nginx is associated to any pod we would to bind to the service.

8.5 Create a new pod from the following file without (intentionally) any label

Copy

```
cat > nginx-pod.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

  name: mynginx

  namespace: default

  labels:

spec:

  containers:

  - name: mynginx

    image: nginx:latest

    ports:

    - containerPort: 80

EOF
```

8.6 Create a new pod, using **nginx-pod.yaml**

Copy

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

Output:

```
pod "mynginx" created
```

8.7 Verify the created pods status

Copy

```
kubectl get pods
```

Output:

```
NAME                     READY     STATUS    RESTARTS   AGE
mynginx                  1/1       Running   0          1m
nginx-59dc74b9-bw5t9     1/1       Running   0          6m
nginx-59dc74b9-sv29f     1/1       Running   0          5m
nginx-59dc74b9-xbc96     1/1       Running   0          6m
```

Note: Wait till pod status changes to **running** state.

8.8 The just created new pod is not still associated to the nginx service, verify the **nginx endpoints**

Copy

```
kubectl get endpoints | grep nginx
```

Output:

```
nginx        10.244.0.10:80,10.244.1.16:80,10.244.1.17:80   2m
```

8.9 Lets label the new pod with run=nginx label

Copy

```
kubectl label pod mynginx run=nginx
```

Output:

```
pod "mynginx" labeled
```

8.10 We can see a new endpoint is added to the service

Copy

```
kubectl get endpoints | grep nginx
```

Output:

```
nginx        10.244.0.10:80,10.244.1.16:80,10.244.1.17:80 + 1 more...
3m
```

Any pod in the cluster need for the nginx service will be able to talk with this service by the service address no matter which IP address will be assigned to the nginx pod. Also, in case of multiple nginx pods, the service abstraction acts as load balancer between the nginx pods.

8.11 Create a new pod, name as **busybox.yaml** file

Copy

```
cat > busybox.yaml <<EOF

apiVersion: v1

kind: Pod

metadata:

  name: busybox

  namespace: default

spec:

  containers:

  - image: busybox
```

```
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
  restartPolicy: Always
EOF
```

8.12 We'll use this pod to address the nginx service

Copy

```
kubectl create -f busybox.yaml
pod "busybox" created
```

8.13 Verify **busybox** pod is in running state:

Copy

```
kubectl get pods
```

Output:

```
NAME                    READY     STATUS    RESTARTS   AGE
busybox                 1/1       Running   0          1m
mynginx                 1/1       Running   0          3m
nginx-59dc74b9-bw5t9    1/1       Running   0          9m
nginx-59dc74b9-sv29f    1/1       Running   0          8m
```

```
nginx-59dc74b9-xbc96   1/1       Running   0          9m
```

Note: Wait till pod status changes to **running** state.

8.14 Capture the **service_ip** from the **nginx** service

Copy

```
service_ip=`kubectl get svc | grep nginx | awk '{print $3}'`

echo $service_ip
```

Output:

```
10.103.148.246
```

8.15 Download the busybox application with assigned service_ip

Copy

```
kubectl exec -it busybox -- wget -O - $service_ip:8000

Connecting to 10.254.105.187:8000 (10.254.105.187:8000)

<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx!</title>

</head>

<body>

<h1>Welcome to nginx!</h1>

<p>If you see this page, the nginx web server is successfully installe
d and
```

```
working. Further configuration is required.</p>

...

</body>

</html>

-                    100% |*******************************|   612   0:
00:00 ETA
```

8.16 Delete the deployed nginx pod

Copy

```
kubectl delete  deploy nginx
```

Output:

```
deployment.extensions "nginx" deleted
```

8.17 Delete the service nginx

Copy

```
kubectl delete svc nginx
```

Output:

```
service "nginx" deleted
```

8.18 Cleanup the other remaining pods by running the below command:

Copy

```
kubectl delete pods --all
```

Output:

```
pod "busybox" deleted

pod "mynginx" deleted
```

In kubernetes, the service abstraction acts as stable entrypoint for any application, no matter wich is the IP address of the pod(s) running that application. We can destroy all nginx pods and recreate but service will be always the same IP address and port.

9. Daemons

A Daemon Set is a controller type ensuring each node in the cluster runs a pod. As new node is added to the cluster, a new pod is added to the node. As the node is removed from the cluster, the pod running on it is removed and not scheduled on another node. Deleting a Daemon Set will clean up all the pods it created.

9.1 The configuration file **nginx-daemon-set.yaml** defines a daemon set for the nginx application

Copy

```
cat > nginx-daemon-set.yaml << EOF

apiVersion: extensions/v1beta1

kind: DaemonSet

metadata:

  labels:

    run: nginx

  name: nginx-ds

  namespace:

spec:

  selector:

    matchLabels:
```

```
      run: nginx-ds
  template:
    metadata:
      labels:
        run: nginx-ds
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
EOF
```

9.2 Create a daemon set using **nginx-daemon-set.yaml** file

Copy

```
kubectl create -f nginx-daemon-set.yaml
```

Output:

```
daemonset.extensions "nginx-ds" created
```

9.3 Verify the daemon-set status

Copy

```
kubectl get ds nginx-ds -o wide

NAME       DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE       CONTAINERS   IMAGES         SELECTOR

nginx-ds   2         2         2         2            2
1m        nginx        nginx:latest   run=nginx-ds
```

Note: Wait till the ready state changes to "**2** .

9.4 There are exactly two pods since we have two nodes

Copy

```
kubectl get pods -o wide
```

Output:

```
NAME             READY     STATUS    RESTARTS   AGE       IP
NODE

nginx-ds-kts84   1/1       Running   0          1m        10.244.1.20
pod38-node.onecloud.com

nginx-ds-mtt9k   1/1       Running   0          1m        10.244.0.11
pod38-master.onecloud.com
```

Note: Wait till pod status changes to **running** state.

and each pod is running on a different node.

If you delete a node from the cluster, the running pod on it is removed and not scheduled on other nodes as happens with other types of controllers

Adding back the node to the cluster, you will see a new pod scheduled on that node.

9.5 It is possible to exclude some nodes from the daemon set by forcing the node selector.

```
...
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
...
      nodeSelector:
        kubernetes.io/hostname: spare.onecloud.com
```

9.6 Cleanup the daemonset nginx

Copy

```
kubectl delete ds nginx-ds
```

Output:

```
daemonset.extensions "nginx-ds" deleted
```