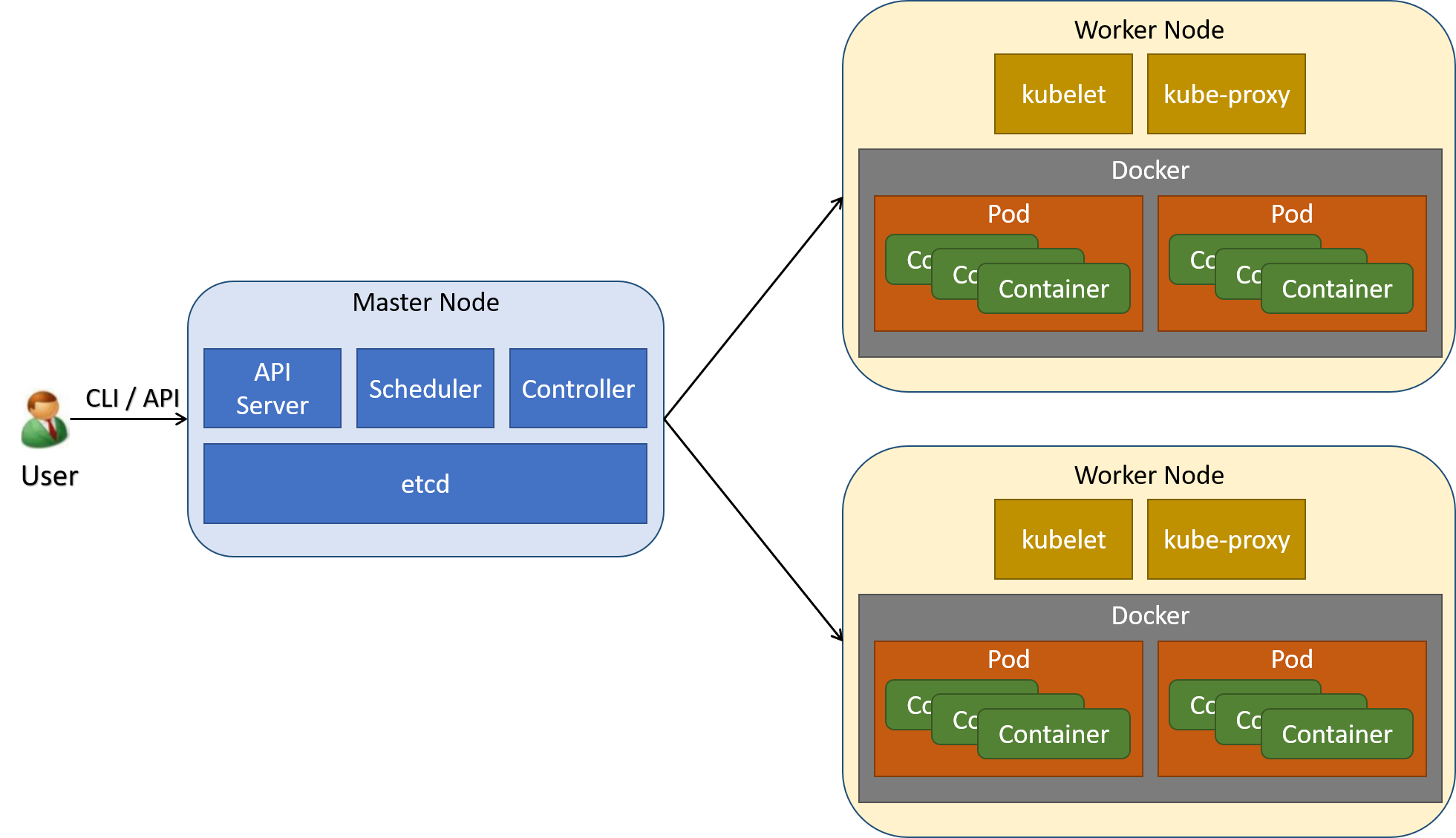


What is Kubernetes ?

Kubernetes is an **open-source container orchestration platform** that automates deployment, management and scaling of applications.

Architecture

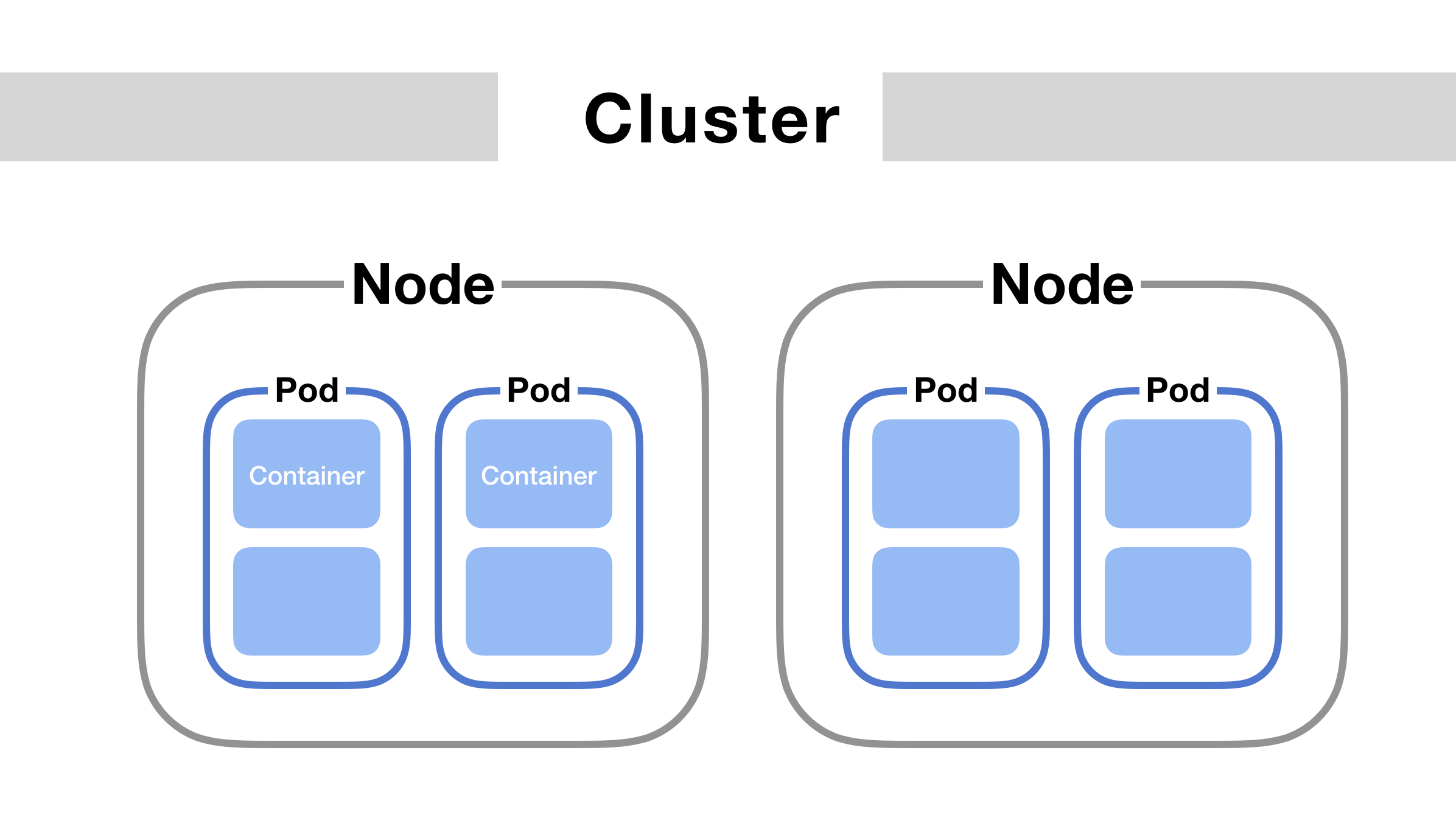


# **Most used conceptual terms :**

|  |  |
| --- | --- |
| * Pod * Volume * Replica set * Config maps * service | * Namespaces * Secrets * Deployments * Jobs * Daemon sets |

# **Kubernetes - PODs**

Pods Pods Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.



Multi-container pod

Multi-container pods are the pods that contain two or more related containers that share resources like network space, shared volumes, etc and work together as a single unit. One of the reasons to use a multi-container pod is simpler communication between containers.

* We can create pods and other components using two ways:

**1. Declarative way:** we can declare what we need in the yaml script and can deploy them, it is reusable again and again we can use them.

**2. Imperative way:** we can deploy applications using cli commands, this is useful in pre-production not recommendable for production.

**Some basics we need to know about pods:**

* If pods deleted automatically associated resources also deleted.
* If the node fails automatically the pod will be deleted.
* Once the resources being terminated there is no chance to restart them same resources again, we can start like same specs other resources from our scripts and commands.
* The volumes are mapped with pods are available until pod available , if pod or node dies those will be deleted.

**Get Worker Nodes Status**

* Verify if kubernetes worker nodes are ready.

# Get Worker Node Status  
kubectl get nodes  
  
# Get Worker Node Status with wide option  
kubectl get nodes -o wide

### **Create a Pod**

* Create a Pod

# Template  
kubectl run <desired-pod-name> --image <Container-Image> --generator=run-pod/v1  
  
# Replace Pod Name, Container Image  
kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0 --generator=run-pod/v1

* **Important Note:** Without **--generator=run-pod/v1** it will create a pod with a deployment which is another core kubernetes concept which we will learn in next few minutes.
* **Important Note:**
  + With **Kubernetes 1.18 version**, there is lot clean-up to **kubectl run** command.
  + The below will suffice to create a Pod as a pod without creating deployment. We dont need to add **--generator=run-pod/v1**

kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0

### **List Pods**

* Get the list of pods

# List Pods  
kubectl get pods  
  
# Alias name for pods is po  
kubectl get po

### **List Pods with wide option**

* List pods with wide option which also provide Node information on which Pod is running

kubectl get pods -o wide

### **What happened in the backgroup when above command is run?**

1. Kubernetes created a pod
2. Pulled the docker image from docker hub
3. Created the container in the pod
4. Started the container present in the pod

### **Describe Pod**

* Describe the POD, primarily required during troubleshooting.
* Events shown will be of a great help during troubleshooting.

# To get list of pod names  
kubectl get pods  
  
# Describe the Pod  
kubectl describe pod <Pod-Name>  
kubectl describe pod my-first-pod

### **Access Application**

* Currently we can access this application only inside worker nodes.
* To access it externally, we need to create a **NodePort Service**.
* **Services** is one very very important concept in Kubernetes.

### **Delete Pod**

# To get list of pod names  
kubectl get pods  
  
# Delete Pod  
kubectl delete pod <Pod-Name>  
kubectl delete pod my-first-pod

## **Step-03: NodePort Service Introduction**

* What are Services in k8s?
* What is a NodePort Service?
* How it works?

## **Step-04: Demo - Expose Pod with a Service**

* Expose pod with a service (NodePort Service) to access the application externally (from internet)
* **Ports**
  + **port:** Port on which node port service listens in Kubernetes cluster internally
  + **targetPort:** We define container port here on which our application is running.
  + **NodePort:** Worker Node port on which we can access our application.

# Create a Pod  
kubectl run <desired-pod-name> --image <Container-Image> --generator=run-pod/v1  
kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0 --generator=run-pod/v1  
  
# Expose Pod as a Service  
kubectl expose pod <Pod-Name> --type=NodePort --port=80 --name=<Service-Name>  
kubectl expose pod my-first-pod --type=NodePort --port=80 --name=my-first-service  
  
# Get Service Info  
kubectl get service  
kubectl get svc  
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide

* **Access the Application using Public IP**

http://<node1-public-ip>:<Node-Port>

* **Important Note about: target-port**
  + If target-port is not defined, by default and for convenience, the **targetPort** is set to the same value as the **port** field.

# Below command will fail when accessing the application, as service port (81) and container port (80) are different  
kubectl expose pod my-first-pod --type=NodePort --port=81 --name=my-first-service2   
  
# Expose Pod as a Service with Container Port (--taret-port)  
kubectl expose pod my-first-pod --type=NodePort --port=81 --target-port=80 --name=my-first-service3  
  
# Get Service Info  
kubectl get service  
kubectl get svc  
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide

|  |
| --- |
| **If external ip not allocated we can allocate ip**  **kubectl patch svc nginx -p '{"spec":{"externalIPs":["54.90.135.121"]}}'** |

* **Access the Application using Public IP**

http://<node1-public-ip>:<Node-Port>

## **Step-05: Interact with a Pod**

### **Verify Pod Logs**

# Get Pod Name  
kubectl get po  
  
# Dump Pod logs  
kubectl logs <pod-name>  
kubectl logs my-first-pod  
  
# Stream pod logs with -f option and access application to see logs  
kubectl logs <pod-name>  
kubectl logs -f my-first-pod

* **Important Notes**
  + Refer below link and search for **Interacting with running Pods** for additional log options
  + Troubleshooting skills are very important. So please go through all logging options available and master them.
  + **Reference:** <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

### **Connect to Container in a POD**

* **Connect to a Container in POD and execute commands**

# Connect to Nginx Container in a POD  
kubectl exec -it <pod-name> -- /bin/bash  
kubectl exec -it my-first-pod -- /bin/bash  
  
# Execute some commands in Nginx container  
ls  
cd /usr/share/nginx/html  
cat index.html  
exit

* **Running individual commands in a Container**

kubectl exec -it <pod-name> env  
  
# Sample Commands  
kubectl exec -it my-first-pod env  
kubectl exec -it my-first-pod ls  
kubectl exec -it my-first-pod cat /usr/share/nginx/html/index.html

## **Step-06: Get YAML Output of Pod & Service**

### **Get YAML Output**

# Get pod definition YAML output  
kubectl get pod my-first-pod -o yaml   
  
# Get service definition YAML output  
kubectl get service my-first-service -o yaml

## **Step-07: Clean-Up**

# Get all Objects in default namespace  
kubectl get all  
  
# Delete Services  
kubectl delete svc my-first-service  
kubectl delete svc my-first-service2  
kubectl delete svc my-first-service3  
  
# Delete Pod  
kubectl delete pod my-first-pod  
  
# Get all Objects in default namespace  
kubectl get all

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

kubectl run nginx-podtest --image=nginx:latest

kubectl expose pod/nginx-podtest --type=NodePort --port=80

Kubectl describe pod nginx-podtest

Kubectl get svc –o wide

kubectl get svc

kubectl exec -it nginx-podtest /bin/bash

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

# **Kubernetes - ReplicaSets**

## **Step-01: Introduction to ReplicaSets**

* What are ReplicaSets?

A ReplicaSet is a process that runs multiple instances of a Pod and keeps the specified number of Pods constant. Its purpose is to maintain the specified number of Pod instances running in a cluster at any given time to prevent users from losing access to their application when a Pod fails or is inaccessible.

* What is the advantage of using ReplicaSets?

A ReplicaSet ensures that a specified number of Pod replicas are running continuously and helps with load-balancing in case of an increase in resource usage. Creating a Kubernetes ReplicaSet We will create an example ReplicaSet using the below configuration, just like we created a Pod in part 3 of this series.

## **Step-02: Create ReplicaSet**

### **Create ReplicaSet**

* Create ReplicaSet

kubectl create -f replicaset-demo.yml

* **replicaset-demo.yml**

apiVersion: apps/v1  
kind: ReplicaSet  
metadata:  
 name: my-helloworld-rs  
 labels:  
 app: my-helloworld  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: my-helloworld  
 template:  
 metadata:  
 labels:  
 app: my-helloworld  
 spec:  
 containers:  
 - name: my-helloworld-app  
 image: stacksimplify/kube-helloworld:1.0.0

### **List ReplicaSets**

* Get list of ReplicaSets

kubectl get replicaset  
kubectl get rs

### **Describe ReplicaSet**

* Describe the newly created ReplicaSet

kubectl describe rs/<replicaset-name>  
  
kubectl describe rs/my-helloworld-rs  
[or]  
kubectl describe rs my-helloworld-rs

### **List of Pods**

* Get list of Pods

#Get list of Pods  
kubectl get pods  
kubectl describe pod <pod-name>  
  
# Get list of Pods with Pod IP and Node in which it is running  
kubectl get pods -o wide

### **Verify the Owner of the Pod**

* Verify the owner reference of the pod.
* Verify under **"name"** tag under **"ownerReferences"**. We will find the replicaset name to which this pod belongs to.

kubectl get pods <pod-name> -o yaml  
kubectl get pods my-helloworld-rs-c8rrj -o yaml

## **Step-03: Expose ReplicaSet as a Service**

* Expose ReplicaSet with a service (NodePort Service) to access the application externally (from internet)

# Expose ReplicaSet as a Service  
kubectl expose rs <ReplicaSet-Name> --type=NodePort --port=80 --target-port=8080 --name=<Service-Name-To-Be-Created>  
kubectl expose rs my-helloworld-rs --type=NodePort --port=80 --target-port=8080 --name=my-helloworld-rs-service  
  
# Get Service Info  
kubectl get service  
kubectl get svc  
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide

* **Access the Application using Public IP**

http://<node1-public-ip>:<Node-Port>/hello

## **Step-04: Test Replicaset Reliability or High Availability**

* Test how the high availability or reliability concept is achieved automatically in Kubernetes
* Whenever a POD is accidentally terminated due to some application issue, ReplicaSet should auto-create that Pod to maintain desired number of Replicas configured to achive High Availability.

# To get Pod Name  
kubectl get pods  
  
# Delete the Pod  
kubectl delete pod <Pod-Name>  
  
# Verify the new pod got created automatically  
kubectl get pods (Verify Age and name of new pod)

## **Step-05: Test ReplicaSet Scalability feature**

* Test how scalability is going to seamless & quick
* Update the **replicas** field in **replicaset-demo.yml** from 3 to 6.

# Before change  
spec:  
 replicas: 3  
  
# After change  
spec:  
 replicas: 6

* Update the ReplicaSet

# Apply latest changes to ReplicaSet  
kubectl replace -f replicaset-demo.yml  
  
# Verify if new pods got created  
kubectl get pods -o wide

## **Step-06: Delete ReplicaSet & Service**

### **Delete ReplicaSet**

# Delete ReplicaSet  
kubectl delete rs <ReplicaSet-Name>  
  
# Sample Commands  
kubectl delete rs/my-helloworld-rs  
[or]  
kubectl delete rs my-helloworld-rs  
  
# Verify if ReplicaSet got deleted  
kubectl get rs

### **Delete Service created for ReplicaSet**

# Delete Service  
kubectl delete svc <service-name>  
  
# Sample Commands  
kubectl delete svc my-helloworld-rs-service  
[or]  
kubectl delete svc/my-helloworld-rs-service  
  
# Verify if Service got deleted  
kubectl get svc

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

**replicaset-demo.yml**

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: my-helloworld-rs

labels:

app: my-helloworld

spec:

replicas: 3

selector:

matchLabels:

app: my-helloworld

template:

metadata:

labels:

app: my-helloworld

spec:

containers:

- name: my-helloworld-app

image: gudditi/webapp:v9

**Kubectl apply –f replicaset-demo.yml**

**Kubectl expose rs my-helloworld-rs –-type=NodePort –port=80**

**Kubectl get svc**

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

# **Kubernetes - Deployment**

## **Topics**

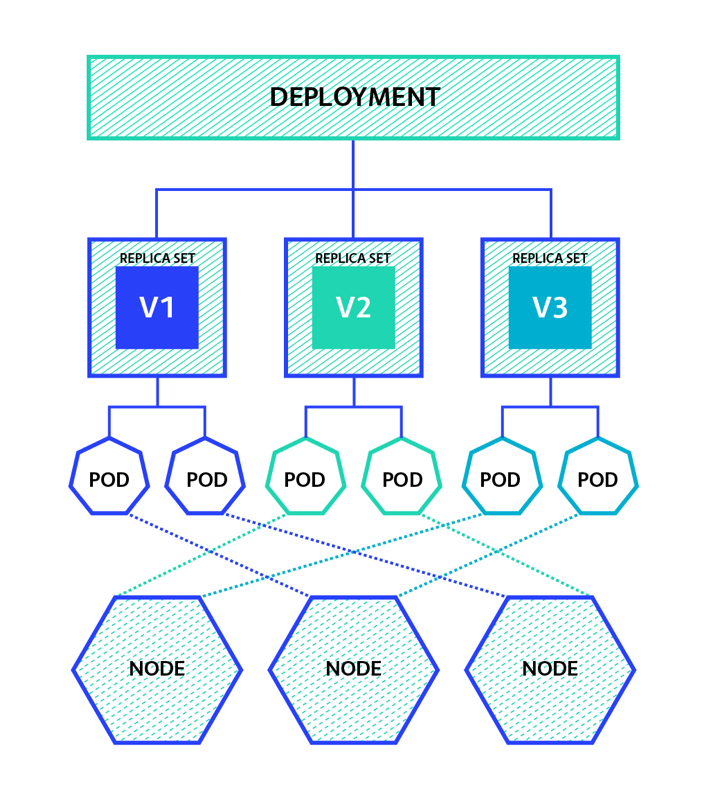
* Create Deployment
* Scale the Deployment
* Expose Deployment as a Service
* Update Deployment
* Rollback Deployment
* Rolling Restarts
* Pause & Resume Deployments
* Canary Deployments (Will be covered at Declarative section of Deployments)

# **Kubernetes - Deployment**

## **Step-01: Introduction to Deployments**

* What is a Deployment?

deployment is a resource object in Kubernetes that provides declarative updates to applications. A deployment allows you to describe an application’s life cycle, such as which images to use for the app, the number of pods there should be, and the way in which they should be updated.



* What all we can do using Deployment?
* Create a Deployment
* Scale the Deployment
* Expose the Deployment as a Service

## **Step-02: Create Deployment**

* Create Deployment to rollout a ReplicaSet
* Verify Deployment, ReplicaSet & Pods
* **Docker Image Location:** <https://hub.docker.com/repository/docker/stacksimplify/kubenginx>

# Create Deployment  
kubectl create deployment <Deplyment-Name> --image=<Container-Image>  
kubectl create deployment my-first-deployment --image=stacksimplify/kubenginx:1.0.0   
  
# Verify Deployment  
kubectl get deployments  
kubectl get deploy   
  
# Describe Deployment  
kubectl describe deployment <deployment-name>  
kubectl describe deployment my-first-deployment  
  
# Verify ReplicaSet  
kubectl get rs  
  
# Verify Pod  
kubectl get po

## **Step-03: Scaling a Deployment**

* Scale the deployment to increase the number of replicas (pods)

# Scale Up the Deployment  
kubectl scale --replicas=20 deployment/<Deployment-Name>  
kubectl scale --replicas=20 deployment/my-first-deployment   
  
# Verify Deployment  
kubectl get deploy  
  
# Verify ReplicaSet  
kubectl get rs  
  
# Verify Pods  
kubectl get po  
  
# Scale Down the Deployment  
kubectl scale --replicas=10 deployment/my-first-deployment   
kubectl get deploy

## **Step-04: Expose Deployment as a Service**

* Expose **Deployment** with a service (NodePort Service) to access the application externally (from internet)

# Expose Deployment as a Service  
kubectl expose deployment <Deployment-Name> --type=NodePort --port=80 --target-port=80 --name=<Service-Name-To-Be-Created>  
kubectl expose deployment my-first-deployment --type=NodePort --port=80 --target-port=80 --name=my-first-deployment-service  
  
# Get Service Info  
kubectl get svc  
Observation: Make a note of port which starts with 3 (Example: 80:3xxxx/TCP). Capture the port 3xxxx and use it in application URL below.   
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide  
Observation: Make a note of "EXTERNAL-IP" if your Kubernetes cluster is setup on AWS EKS.

* **Access the Application using Public IP**

http://<worker-node-public-ip>:<Node-Port>

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

kubectl create deployment my-nginx-deployment --image=nginx:latest

kubectl get deploy

kubectl describe deploy my-nginx-deployment

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

kubectl expose deployment/my-nginx-deployment --type=NodePort --port=80 --target-port=80

kubectl get svc

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

# **Kubernetes - Update Deployments**

## **Step-00: Introduction**

* We can update deployments using two options
  + Set Image
  + Edit Deployment

## **Step-01: Updating Application version V1 to V2 using "Set Image" Option**

### **Update Deployment**

* **Observation:** Please Check the container name in spec.container.name yaml output and make a note of it and replace in kubectl set image command

# Get Container Name from current deployment  
kubectl get deployment my-first-deployment -o yaml  
  
# Update Deployment - SHOULD WORK NOW  
kubectl set image deployment/<Deployment-Name> <Container-Name>=<Container-Image> --record=true  
kubectl set image deployment/my-first-deployment kubenginx=stacksimplify/kubenginx:2.0.0 --record=true

### **Verify Rollout Status (Deployment Status)**

* **Observation:** By default, rollout happens in a rolling update model, so no downtime.

# Verify Rollout Status   
kubectl rollout status deployment/my-first-deployment  
  
# Verify Deployment  
kubectl get deploy

### **Describe Deployment**

* **Observation:**
  + Verify the Events and understand that Kubernetes by default do "Rolling Update" for new application releases.
  + With that said, we will not have downtime for our application.

# Descibe Deployment  
kubectl describe deployment my-first-deployment

### **Verify ReplicaSet**

* **Observation:** New ReplicaSet will be created for new version

# Verify ReplicaSet  
kubectl get rs

### **Verify Pods**

* **Observation:** Pod template hash label of new replicaset should be present for PODs letting us know these pods belong to new ReplicaSet.

# List Pods  
kubectl get po

### **Verify Rollout History of a Deployment**

* **Observation:** We have the rollout history, so we can switch back to older revisions using revision history available to us.

# Check the Rollout History of a Deployment  
kubectl rollout history deployment/<Deployment-Name>  
kubectl rollout history deployment/my-first-deployment

### **Access the Application using Public IP**

* We should see Application Version:V2 whenever we access the application in browser

# Get NodePort  
kubectl get svc  
Observation: Make a note of port which starts with 3 (Example: 80:3xxxx/TCP). Capture the port 3xxxx and use it in application URL below.   
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide  
Observation: Make a note of "EXTERNAL-IP" if your Kubernetes cluster is setup on AWS EKS.  
  
# Application URL  
http://<worker-node-public-ip>:<Node-Port>

## **Step-02: Update the Application from V2 to V3 using "Edit Deployment" Option**

### **Edit Deployment**

# Edit Deployment  
kubectl edit deployment/<Deployment-Name> --record=true  
kubectl edit deployment/my-first-deployment --record=true

# Change From 2.0.0  
 spec:  
 containers:  
 - image: stacksimplify/kubenginx:2.0.0  
  
# Change To 3.0.0  
 spec:  
 containers:  
 - image: stacksimplify/kubenginx:3.0.0

### **Verify Rollout Status**

* **Observation:** Rollout happens in a rolling update model, so no downtime.

# Verify Rollout Status   
kubectl rollout status deployment/my-first-deployment

### **Verify Replicasets**

* **Observation:** We should see 3 ReplicaSets now, as we have updated our application to 3rd version 3.0.0

# Verify ReplicaSet and Pods  
kubectl get rs  
kubectl get po

### **Verify Rollout History**

# Check the Rollout History of a Deployment  
kubectl rollout history deployment/<Deployment-Name>  
kubectl rollout history deployment/my-first-deployment

### **Access the Application using Public IP**

* We should see Application Version:V3 whenever we access the application in browser

# Get NodePort  
kubectl get svc  
Observation: Make a note of port which starts with 3 (Example: 80:3xxxx/TCP). Capture the port 3xxxx and use it in application URL below.   
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide  
Observation: Make a note of "EXTERNAL-IP" if your Kubernetes cluster is setup on AWS EKS.  
  
# Application URL  
http://<worker-node-public-ip>:<Node-Port>

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

no

---

Kubectl get deploy my-nginx-deployment -o yaml

**kubectl set image deployment/<Deployment-Name> <Container-Name>=<Container-Image> --record=true**

kubectl set image deployment/my-nginx-deployment nginx=gudditi/kube-nginx:v1 --record=true

# Verify Rollout Status   
kubectl rollout status deployment/my-nginx-deployment  
# Verify Deployment  
kubectl get deploy

--

Edit deployment yaml file

kubectl edit deployment/my-nginx-deployment

**Error – ImagePullBackOff**

**Open any deployment or any service**

kubectl edit deployment/my-nginx-deployment

**Update the section mentioned**

* if you omit the imagePullPolicy field, and the tag for the container image is :latest, imagePullPolicy is automatically set to Always;
* if you omit the imagePullPolicy field, and you don't specify the tag for the container image, imagePullPolicy is automatically set to Always;
* if you omit the imagePullPolicy field, and you specify the tag for the container image that isn't :latest, the imagePullPolicy is automatically set to IfNotPresent.

|  |
| --- |
| spec:  containers:  - image: gudditi/gudditi/kube-nginx:v3  imagePullPolicy: IfNotPresent { change to Always }  name: kube-nginx  resources: {}  terminationMessagePath: /dev/termination-log  terminationMessagePolicy: File  dnsPolicy: ClusterFirst |

Kubectl edit deployment/my-nginx-deployment

Kubectl rollout history deploy/my-nginx-deployment

**Remove all stopeed images&conatiners using**

**docker image prune –a**

**docker container prune**

# ------------------------------------------------------------------------------------- **Rollback Deployment**

## **Step-00: Introduction**

* We can rollback a deployment in two ways.
  + Previous Version
  + Specific Version

## **Step-01: Rollback a Deployment to previous version**

### **Check the Rollout History of a Deployment**

# List Deployment Rollout History  
kubectl rollout history deployment/<Deployment-Name>  
kubectl rollout history deployment/my-first-deployment

### **Verify changes in each revision**

* **Observation:** Review the "Annotations" and "Image" tags for clear understanding about changes.

# List Deployment History with revision information  
kubectl rollout history deployment/my-first-deployment --revision=1  
kubectl rollout history deployment/my-first-deployment --revision=2  
kubectl rollout history deployment/my-first-deployment --revision=3

### **Rollback to previous version**

* **Observation:** If we rollback, it will go back to revision-2 and its number increases to revision-4

# Undo Deployment  
kubectl rollout undo deployment/my-first-deployment  
  
# List Deployment Rollout History  
kubectl rollout history deployment/my-first-deployment

### **Verify Deployment, Pods, ReplicaSets**

kubectl get deploy  
kubectl get rs  
kubectl get po  
kubectl describe deploy my-first-deployment

### **Access the Application using Public IP**

* We should see Application Version:V2 whenever we access the application in browser

# Get NodePort  
kubectl get svc  
Observation: Make a note of port which starts with 3 (Example: 80:3xxxx/TCP). Capture the port 3xxxx and use it in application URL below.   
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide  
Observation: Make a note of "EXTERNAL-IP" if your Kubernetes cluster is setup on AWS EKS.  
  
# Application URL  
http://<worker-node-public-ip>:<Node-Port>

## **Step-02: Rollback to specific revision**

### **Check the Rollout History of a Deployment**

# List Deployment Rollout History  
kubectl rollout history deployment/<Deployment-Name>  
kubectl rollout history deployment/my-first-deployment

### **Rollback to specific revision**

# Rollback Deployment to Specific Revision  
kubectl rollout undo deployment/my-first-deployment --to-revision=3

### **List Deployment History**

* **Observation:** If we rollback to revision 3, it will go back to revision-3 and its number increases to revision-5 in rollout history

# List Deployment Rollout History  
kubectl rollout history deployment/my-first-deployment

### **Access the Application using Public IP**

* We should see Application Version:V3 whenever we access the application in browser

# Get NodePort  
kubectl get svc  
Observation: Make a note of port which starts with 3 (Example: 80:3xxxx/TCP). Capture the port 3xxxx and use it in application URL below.   
  
# Get Public IP of Worker Nodes  
kubectl get nodes -o wide  
Observation: Make a note of "EXTERNAL-IP" if your Kubernetes cluster is setup on AWS EKS.  
  
# Application URL  
http://<worker-node-public-ip>:<Node-Port>

## **Step-03: Rolling Restarts of Application**

* Rolling restarts will kill the existing pods and recreate new pods in a rolling fashion.

# Rolling Restarts  
kubectl rollout restart deployment/<Deployment-Name>  
kubectl rollout restart deployment/my-first-deployment  
  
# Get list of Pods  
kubectl get po

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

Kubectl rollout undo deployment/my-nginx-deployment

Kubectl rollout history deploy/my-nginx-deployment

kubectl rollout undo deploy/my-nginx-deployment --to-revision=3

kubectl rollout restart deploy/my-nginx-deployment

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

# **Pause & Resume Deployments**

## **Step-00: Introduction**

* Why do we need Pausing & Resuming Deployments?
  + If we want to make multiple changes to our Deployment, we can pause the deployment make all changes and resume it.
* We are going to update our Application Version from **V3 to V4** as part of learning "Pause and Resume Deployments"

## **Step-01: Pausing & Resuming Deployments**

### **Check current State of Deployment & Application**

# Check the Rollout History of a Deployment  
kubectl rollout history deployment/my-first-deployment   
Observation: Make a note of last version number  
  
# Get list of ReplicaSets  
kubectl get rs  
Observation: Make a note of number of replicaSets present.  
  
# Access the Application   
http://<worker-node-ip>:<Node-Port>  
Observation: Make a note of application version

### **Pause Deployment and Two Changes**

# Pause the Deployment  
kubectl rollout pause deployment/<Deployment-Name>  
kubectl rollout pause deployment/my-first-deployment  
  
# Update Deployment - Application Version from V3 to V4  
kubectl set image deployment/my-first-deployment kubenginx=stacksimplify/kubenginx:4.0.0 --record=true  
  
# Check the Rollout History of a Deployment  
kubectl rollout history deployment/my-first-deployment   
Observation: No new rollout should start, we should see same number of versions as we check earlier with last version number matches which we have noted earlier.  
  
# Get list of ReplicaSets  
kubectl get rs  
Observation: No new replicaSet created. We should have same number of replicaSets as earlier when we took note.   
  
# Make one more change: set limits to our container  
kubectl set resources deployment/my-first-deployment -c=kubenginx --limits=cpu=20m,memory=30Mi

### **Resume Deployment**

# Resume the Deployment  
kubectl rollout resume deployment/my-first-deployment  
  
# Check the Rollout History of a Deployment  
kubectl rollout history deployment/my-first-deployment   
Observation: You should see a new version got created  
  
# Get list of ReplicaSets  
kubectl get rs  
Observation: You should see new ReplicaSet.

### **Access Application**

# Access the Application   
http://<node1-public-ip>:<Node-Port>  
Observation: You should see Application V4 version

## **Step-02: Clean-Up**

# Delete Deployment  
kubectl delete deployment my-first-deployment  
  
# Delete Service  
kubectl delete svc my-first-deployment-service  
  
# Get all Objects from Kubernetes default namespace  
kubectl get all

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

Kubectl get pods

Kubetcl get deployments

Kubectl get svc

---

Kubectl rollout pause deploy/my-nginx-deployment

Kubectl rollout resume deploy/my-nginx-deployment

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

# **Kubernetes - Services**

## **Step-01: Introduction to Services**

* **Service Types**
  1. ClusterIp
  2. NodePort
  3. LoadBalancer
  4. ExternalName
* We are going to look in to ClusterIP and NodePort in this section with a detailed example.
* LoadBalancer Type is primarily for cloud providers and it will differ cloud to cloud, so we will do it accordingly (per cloud basis)
* ExternalName doesn't have Imperative commands and we need to write YAML definition for the same, so we will look in to it as and when it is required in our course.

## **Step-02: ClusterIP Service - Backend Application Setup**

* Create a deployment for Backend Application (Spring Boot REST Application)
* Create a ClusterIP service for load balancing backend application.

# Create Deployment for Backend Rest App  
kubectl create deployment my-backend-rest-app --image=stacksimplify/kube-helloworld:1.0.0   
kubectl get deploy  
  
# Create ClusterIp Service for Backend Rest App  
kubectl expose deployment my-backend-rest-app --port=8080 --target-port=8080 --name=my-backend-service  
kubectl get svc  
Observation: We don't need to specify "--type=ClusterIp" because default setting is to create ClusterIp Service.

* **Important Note:** If backend application port (Container Port: 8080) and Service Port (8080) are same we don't need to use **--target-port=8080** but for avoiding the confusion i have added it. Same case applies to frontend application and service.
* **Backend HelloWorld Application Source** [kube-helloworld](https://github.com/stacksimplify/kubernetes-fundamentals/blob/master/00-Docker-Images/02-kube-backend-helloworld-springboot/kube-helloworld)

## **Step-03: NodePort Service - Frontend Application Setup**

* We have implemented **NodePort Service** multiple times so far (in pods, replicasets and deployments), even then we are going to implement one more time to get a full architectural view in relation with ClusterIp service.
* Create a deployment for Frontend Application (Nginx acting as Reverse Proxy)
* Create a NodePort service for load balancing frontend application.
* **Important Note:** In Nginx reverse proxy, ensure backend service name my-backend-service is updated when you are building the frontend container. We already built it and put ready for this demo (stacksimplify/kube-frontend-nginx:1.0.0)
* **Nginx Conf File**

server {  
 listen 80;  
 server\_name localhost;  
 location / {  
 # Update your backend application Kubernetes Cluster-IP Service name and port below   
 # proxy\_pass http://<Backend-ClusterIp-Service-Name>:<Port>;   
 proxy\_pass <http://my-backend-service:8080>;  
 }  
 error\_page 500 502 503 504 /50x.html;  
 location = /50x.html {  
 root /usr/share/nginx/html;  
 }  
}

* **Docker Image Location:** <https://hub.docker.com/repository/docker/stacksimplify/kube-frontend-nginx>
* **Frontend Nginx Reverse Proxy Application Source** [kube-frontend-nginx](https://github.com/stacksimplify/kubernetes-fundamentals/blob/master/00-Docker-Images/03-kube-frontend-nginx)

# Create Deployment for Frontend Nginx Proxy  
kubectl create deployment my-frontend-nginx-app --image=stacksimplify/kube-frontend-nginx:1.0.0   
kubectl get deploy  
  
# Create ClusterIp Service for Frontend Nginx Proxy  
kubectl expose deployment my-frontend-nginx-app --type=NodePort --port=80 --target-port=80 --name=my-frontend-service  
kubectl get svc  
  
# Capture IP and Port to Access Application  
kubectl get svc  
kubectl get nodes -o wide  
http://<node1-public-ip>:<Node-Port>/hello  
  
# Scale backend with 10 replicas  
kubectl scale --replicas=10 deployment/my-backend-rest-app  
  
# Test again to view the backend service Load Balancing  
http://<node1-public-ip>:<Node-Port>/hello

## **Pending Topics**

* We will look in tho these items when we progress in to course on that respective cloud provider
  + LoadBalancer
  + ExternalName

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

# **YAML Basics**

## **Step-01: Comments & Key Value Pairs**

* Space after colon is mandatory to differentiate key and value

# Defining simple key value pairs  
name: Anjaneyulu  
age: 23  
city: Hyderabad

## **Step-02: Dictionary / Map**

* Set of properties grouped together after an item
* Equal amount of blank space required for all the items under a dictionary

person:  
 name: Anjaneyulu  
 age: 23  
 city: Hyderabad

## **Step-03: Array / Lists**

* Dash indicates an element of an array

person: # Dictionary  
 name: Anjaneyulu  
 age: 23  
 city: Hyderabad  
 hobbies: # List   
 - cycling  
 - cookines  
 hobbies: [cycling, cooking] # List with a differnt notation

## **Step-04: Multiple Lists**

* Dash indicates an element of an array

person: # Dictionary  
 name: Anjaneyulu  
 age: 23  
 city: Hyderabad  
 hobbies: # List   
 - cycling  
 - cooking  
 hobbies: [cycling, cooking] # List with a differnt notation   
 friends: #   
 - name: friend1  
 age: 22  
 - name: friend2  
 age: 25

## **Step-05: Sample Pod Tempalte for Reference**

apiVersion: v1 # String  
kind: Pod # String  
metadata: # Dictionary  
 name: myapp-pod  
 labels: # Dictionary   
 app: myapp   
spec:  
 containers: # List  
 - name: myapp  
 image: stacksimplify/kubenginx:1.0.0  
 ports:  
 - containerPort: 80  
 protocol: "TCP"  
 - containerPort: 81  
 protocol: "TCP"

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

# **PODs with YAML**

## **Step-01: Kubernetes YAML Top level Objects**

* Discuss about the k8s YAML top level objects
* **01-kube-base-definition.yml**

apiVersion:  
kind:  
metadata:  
   
spec:

* **Pod API Objects Reference:** <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#pod-v1-core>

## **Step-02: Create Simple Pod Definition using YAML**

* We are going to create a very basic pod definition
* **02-pod-definition.yml**

apiVersion: v1 # String  
kind: Pod # String  
metadata: # Dictionary  
 name: myapp-pod  
 labels: # Dictionary   
 app: myapp   
spec:  
 containers: # List  
 - name: myapp  
 image: stacksimplify/kubenginx:1.0.0  
 ports:  
 - containerPort: 80

* **Create Pod**

# Create Pod  
kubectl create -f 02-pod-definition.yml  
[or]  
kubectl apply -f 02-pod-definition.yml  
  
# List Pods  
kubectl get pods

## **Step-03: Create a NodePort Service**

* **03-pod-nodeport-service.yml**

apiVersion: v1  
kind: Service  
metadata:  
 name: myapp-pod-nodeport-service # Name of the Service  
spec:  
 type: NodePort  
 selector:  
 # Loadbalance traffic across Pods matching this label selector  
 app: myapp  
 # Accept traffic sent to port 80   
 ports:   
 - name: http  
 port: 80 # Service Port  
 targetPort: 80 # Container Port  
 nodePort: 31231 # NodePort

* **Create NodePort Service for Pod**

# Create Service  
kubectl apply -f 03-pod-nodeport-service.yml  
  
# List Service  
kubectl get svc  
  
# Get Public IP  
kubectl get nodes -o wide  
  
# Access Application  
http://<WorkerNode-Public-IP>:<NodePort>  
http://<WorkerNode-Public-IP>:31231

## **API Object References**

* **Pod**: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#pod-v1-core>
* **Service**: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#service-v1-core>

## **Updated API Object References**

* **Pod**: <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/>
* **Service**: <https://kubernetes.io/docs/reference/kubernetes-api/service-resources/service-v1/>
* **Kubernetes API Reference:** <https://kubernetes.io/docs/reference/kubernetes-api/>

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

Vi nginx.yml

**apiVersion: v1 # String**

**kind: Pod # String**

**metadata: # Dictionary**

**name: myapp-pod**

**labels: # Dictionary**

**app: myapp**

**spec:**

**containers: # List**

**- name: myapp**

**image: gudditi/kube-nginx:v6**

**ports:**

**- containerPort: 80**

**---**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: myapp-pod-nodeport-service**

**spec:**

**type: NodePort**

**selector:**

**app: myapp**

**ports:**

**- name: http**

**port: 80 # Service Port**

**targetPort: 80 # Container Port**

**nodePort: 31231 # NodePort**

kubectl create -f nginx.yml

kubectl edit -f nginx.yml

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

# **ReplicaSets with YAML**

## **Step-01: Create ReplicaSet Definition**

* **replicaset-definition.yml**

apiVersion: apps/v1  
kind: ReplicaSet  
metadata:  
 name: myapp2-rs  
spec:  
 replicas: 3 # 3 Pods should exist at all times.  
 selector: # Pods label should be defined in ReplicaSet label selector  
 matchLabels:  
 app: myapp2  
 template:  
 metadata:  
 name: myapp2-pod  
 labels:  
 app: myapp2 # Atleast 1 Pod label should match with ReplicaSet Label Selector  
 spec:  
 containers:  
 - name: myapp2  
 image: stacksimplify/kubenginx:2.0.0  
 ports:  
 - containerPort: 80

## **Step-02: Create ReplicaSet**

* Create ReplicaSet with 3 Replicas

# Create ReplicaSet  
kubectl apply -f 02-replicaset-definition.yml  
  
# List Replicasets  
kubectl get rs

* Delete a pod
* ReplicaSet immediately creates the pod.

# List Pods  
kubectl get pods  
  
# Delete Pod  
kubectl delete pod <Pod-Name>

## **Step-03: Create NodePort Service for ReplicaSet**

apiVersion: v1  
kind: Service  
metadata:  
 name: replicaset-nodeport-service  
spec:  
 type: NodePort  
 selector:  
 app: myapp2  
 ports:  
 - name: http  
 port: 80  
 targetPort: 80  
 nodePort: 31232

* Create NodePort Service for ReplicaSet & Test

# Create NodePort Service  
kubectl apply -f 03-replicaset-nodeport-servie.yml  
  
# List NodePort Service  
kubectl get svc  
  
# Get Public IP  
kubectl get nodes -o wide  
  
# Access Application  
http://<Worker-Node-Public-IP>:<NodePort>  
http://<Worker-Node-Public-IP>:31232

## **API References**

* **ReplicaSet:** <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#replicaset-v1-apps>

-------------------------------------------------------------------------------------- kubectl delete -f /root/yml/nginx.yml

--

Vi replicaset-nginx.yml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: myapp2-rs

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata: # Dictionary

name: myapp-pod

labels: # Dictionary

app: myapp

spec:

containers: # List

- name: myapp

image: gudditi/kube-nginx:v6

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: myapp-pod-nodeport-service

spec:

type: NodePort

selector:

app: myapp

ports:

- name: http

port: 80 # Service Port

targetPort: 80 # Container Port

nodePort: 31231 # NodePort

Kubectl create –f replicaset-nginx.yml

Kubectl edit –f replicaset-nginx.yml

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

# **Deployments with YAML**

## **Step-01: Copy templates from ReplicaSet**

* Copy templates from ReplicaSet and change the kind: Deployment
* Update Container Image version to 3.0.0
* Update NodePort service nodePort: 31233
* Change all names to Deployment
* Change all labels and selectors to myapp3

# Create Deployment  
kubectl apply -f 02-deployment-definition.yml  
kubectl get deploy  
kubectl get rs  
kubectl get po  
  
# Create NodePort Service  
kubectl apply -f 03-deployment-nodeport-service.yml  
  
# List Service  
kubectl get svc  
  
# Get Public IP  
kubectl get nodes -o wide  
  
# Access Application  
http://<Worker-Node-Public-IP>:31233

## **API References**

* **Deployment:** <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#deployment-v1-apps>

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

vi deployment-nginx.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp2-rs

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata: # Dictionary

name: myapp-pod

labels: # Dictionary

app: myapp

spec:

containers: # List

- name: myapp

image: gudditi/kube-nginx:v1

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: myapp-pod-nodeport-service

spec:

type: NodePort

selector:

app: myapp

ports:

- name: http

port: 80 # Service Port

targetPort: 80 # Container Port

nodePort: 31231 # NodePort

kubectl create -f deployment-nginx.yml

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