# Kube comansds azure

===================

Created Friday 31 December 2021

## Create a Pod

kubectl run <desired-pod-name> --image <Container-Image>

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

## # Expose Pod as a Service

kubectl expose pod <Pod-Name> --type=LoadBalancer --port=80 --name=<Service-Name>

kubectl expose pod my-first-pod --type=LoadBalancer --port=80 --name=my-first-service

## # Get Service Info

kubectl get service

kubectl get svc

## # Describe Service

kubectl describe service my-first-service

# Access Application

http://<External-IP-from-get-service-output>

Verify the following after LB Service creation

Azure Standard Load Balancer created for Azure AKS Cluster

Frontend IP Configuration

Load Balancing Rules

Azure Public IP

View the resources in Azure AKS Cluster - Resources section from Azure Portal Management Console

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

# Delete Pod

kubectl delete pod my-first-pod

# Get all Objects in default namespace

kubectl get all

© 2021 GitHub, Inc.

Terms

Privacy

Security

Status

Docs

Contac

Kubernetes - ReplicaSets

Step-01: Introduction to ReplicaSets

What are ReplicaSets?

What is the advantage of using ReplicaSets?

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 (Load Balancer Service) to access the application externally (from internet)

## # Expose ReplicaSet as a Service

kubectl expose rs <ReplicaSet-Name> --type=LoadBalancer --port=80 --target-port=8080 --name=<Service-Name-To-Be-Created>

kubectl expose rs my-helloworld-rs --type=LoadBalancer --port=80 --target-port=8080 --name=my-helloworld-rs-service

## # Get Service Info

kubectl get service

kubectl get svc

Access the Application using External or Public IP

http://<External-IP-from-get-service-output>/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]

# Kubernetes - Deployment

## Step-01: Introduction to Deployments

* What is a Deployment?
* 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=10 deployment/<Deployment-Name>

kubectl scale --replicas=10 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=2 deployment/my-first-deployment

kubectl get deploy

## Step-04: Expose Deployment as a Service

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

# Expose Deployment as a Service

kubectl expose deployment <Deployment-Name> --type=LoadBalancer --port=80 --target-port=80 --name=<Service-Name-To-Be-Created>

kubectl expose deployment my-first-deployment --type=LoadBalancer --port=80 --target-port=80 --name=my-first-deployment-service

# Get Service Info

kubectl get svc

* **Access the Application using Public IP**

http://<External-IP-from-get-service-output>

## !!!!!!!!!!!!!!!!!!!!!!#######################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 Load Balancer IP

kubectl get svc

# Application URL

http://<External-IP-from-get-service-output>

## 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 Load Balancer IP

kubectl get svc

# Application URL

http://<External-IP-from-get-service-output>

# 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 Load Balancer IP

kubectl get svc

# Application URL

http://<External-IP-from-get-service-output>

## 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 Load Balancer IP

kubectl get svc

# Application URL

http://<External-IP-from-get-service-output>

# !!##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 Load Balancer IP

kubectl get svc

# Application URL

http://<External-IP-from-get-service-output>

## 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 Load Balancer IP

kubectl get svc

# Application URL

http://<Load-Balancer-IP>

## 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

# AKS Storage - Storage Classes, Persistent Volume Claims

## Step-01: Introduction

* We are going to create a MySQL Database with persistence storage using **Azure Disks**

| **Kubernetes Object** | **YAML File** |
| --- | --- |
| Storage Class | 01-storage-class.yml |
| Persistent Volume Claim | 02-persistent-volume-claim.yml |
| Config Map | 03-UserManagement-ConfigMap.yml |
| Deployment, Environment Variables, Volumes, VolumeMounts | 04-mysql-deployment.yml |
| ClusterIP Service | 05-mysql-clusterip-service.yml |

## Step-02: Create following Kubernetes manifests

### Create Storage Class manifest

* <https://kubernetes.io/docs/concepts/storage/storage-classes/#volume-binding-mode>
* <https://kubernetes.io/docs/concepts/storage/storage-classes/#azure-disk>

### Create Persistent Volume Claims manifest

# Create Storage Class & PVC

kubectl apply -f kube-manifests/01-storage-class.yml

kubectl apply -f kube-manifests/02-persistent-volume-claim.yml

# List Storage Classes

kubectl get sc

# List PVC

kubectl get pvc

# List PV

kubectl get pv

### Create ConfigMap manifest

* We are going to create a usermgmt database schema during the mysql pod creation time which we will leverage when we deploy User Management Microservice.

### Create MySQL Deployment manifest

* Environment Variables
* Volumes
* Volume Mounts

### Create MySQL ClusterIP Service manifest

* At any point of time we are going to have only one mysql pod in this design so ClusterIP: None will use the Pod IP Address instead of creating or allocating a separate IP for MySQL Cluster IP service.

## Step-03: Create MySQL Database with all above manifests

# Create MySQL Database

kubectl apply -f kube-manifests/

# List Storage Classes

kubectl get sc

# List PVC

kubectl get pvc

# List PV

kubectl get pv

# List pods

kubectl get pods

# List pods based on label name

kubectl get pods -l app=mysql

## Step-04: Connect to MySQL Database

# Connect to MYSQL Database

kubectl run -it --rm --image=mysql:5.6 --restart=Never mysql-client -- mysql -h mysql -pdbpassword11

# Verify usermgmt schema got created which we provided in ConfigMap

mysql> show schemas;

## Step-05: Clean-Up

# Delete All

kubectl delete -f kube-manifests/

## Step-06: Delete PV exclusively - It exists due to retain policy

# List PV

kubect get pv

# Delete PV exclusively

kubectl get pv

kubectl delete pv <PV-NAME>

# Delete Azure Disks

Go to All Services -> Disks -> Select and Delete the Disk