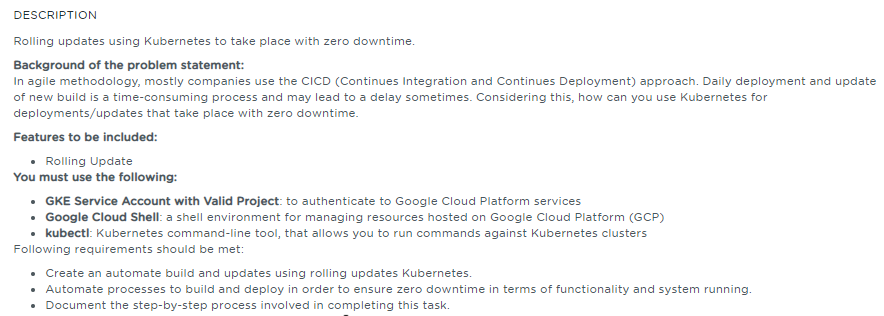
**Build Deployment using Rolling Updates - Assessment**



**Table of Contents**

[**1.** **Introduction** 1](#_Toc44791938)

[**2.** **Work Environment** 1](#_Toc44791939)

[**2.1.** **Prerequisites** 1](#_Toc44791940)

[**2.2.** **Setting up Kubernetes Cluster** 2](#_Toc44791941)

[**3.** **Performing Rolling Update** 3](#_Toc44791942)

[**3.1.** **Deployment Rolling Update** 3](#_Toc44791943)

[**3.1.1.** **Creating a Deployment** 3](#_Toc44791944)

[**3.1.2.** **Updating a Deployment** 4](#_Toc44791945)

[**3.1.3.** **Checking Rollout History of a Deployment** 6](#_Toc44791946)

[**3.1.5.** **Rolling Back to a Previous Version of Deployment** 6](#_Toc44791947)

[**3.1.5.** **Pausing and Resuming a Deployment** 7](#_Toc44791948)

[**3.2.** **StatefulSets Update** 7](#_Toc44791949)

[**3.2.1.** **Creating a StatefulSet** 7](#_Toc44791950)

[**3.2.2.** **Examining a StatefulSet** 8](#_Toc44791951)

[**3.2.3.** **Updating StatefulSet** 9](#_Toc44791952)

[**4.** **References** 9](#_Toc44791953)

# **Introduction**

The intent of this project is to perform the following activities:

* Automate the build/deployment creation and updation using Rolling updates.
* Automate the build and deploy processes to ensure zero down time in terms of functionality, and process that is running in production.

# **Work Environment**

## **Prerequisites**

Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster.

**Note:**

Not using the GKE Cluster as mentioned in the problem statement and requirement specification as am facing issue in creating the Cluster in my Trial account

Am using the SimpliLearn Practice lab session to implement this assessment.

## **Setting up Kubernetes Cluster**

Using the SimpliLearn Practice Lab session, Kubernetes Cluster is being setup with one master and two worker nodes

Run the below command on the master

|  |
| --- |
| $ sudo –i  $ kubeadm reset |

**kubeadm reset** is responsible for cleaning up a node local file system from files that were created using the kubeadm init or kubeadm join commands

Run the below command in order to set up the Kubernetes control plane

|  |
| --- |
| $ kubeadm init |

After executing the init command, Run the below commands to initialize the configuration and set the correct permissions

|  |
| --- |
| $ mkdir -p $HOME/.kube  $ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  $ sudo chown $(id -u):$(id -g) $HOME/.kube/config |

On Master, Run the below command to verify the master has all the required components running successfully

|  |
| --- |
| $ kubectl get pods –n kube-system |

Run the below command on the master to create the join-token

|  |
| --- |
| $ kubeadm token create --print-join-command |

On the slave nodes(kslave1 and kslave2), Run the command

|  |
| --- |
| $ kubeadm join 172.31.17.255:6443 --token wrfgen.elfhgxgoeqd7bnxn --discovery-token-ca-cert-hash sha256:655827d00fec5fa80d71caf58fa67b9efbc8824f1224952a4ad9fb5574fc6eaa |

On Master node, Run the following command to verify the worker nodes are joined the cluster

|  |
| --- |
| $ kubectl get nodes |

For more details on the expected output of the commands used to setup the working environment, please refer the screenshots document (Section: References)

# **Performing Rolling Update**

Users expect application to be available all the time and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done by Rolling Update.

Rolling Update allows Kubernetes workloads like Deployment to update itself with zero downtime by incrementally updating Pods instances with the new ones. The new pods will be scheduled on Nodes with available resources.

The following objects represent Kubernetes workloads. You can trigger a rolling update on these workloads by updating their pod template:

* Deployment
* StatefulSets
* DaemonSets

Here we shall see how to perform the Rolling updates for the Deployment and StatefulSets workloads.

## **Deployment Rolling Update**

## **Creating a Deployment**

The following is an example of a Deployment definition file. It creates a ReplicaSet to bring up 3 nginx Pods:

|  |
| --- |
| **apiVersion: apps/v1**  **kind: Deployment**  **metadata:**  **name: nginx-deployment**  **labels:**  **app: nginx**  **spec:**  **replicas: 3**  **selector:**  **matchLabels:**  **app: nginx**  **template:**  **metadata:**  **labels:**  **app: nginx**  **spec:**  **containers:**  **- name: nginx**  **image: nginx:1.14.2**  **ports:**  **- containerPort: 80** |

Create the Deployment by running the following command:

|  |
| --- |
| **$ kubectl apply -f nginx-deployment.yaml** |

Run kubectl get deployments to check if the Deployment was created.

|  |
| --- |
| **$ kubectl get deployments** |

The output of the get deployments command looks like this:

**NAME READY UP-TO-DATE AVAILABLE AGE**

**nginx-deployment 0/3 0 0 1s**

Run the below command to view the deployment roll out status

|  |
| --- |
| **$ kubectl rollout status deployment.v1.apps/nginx-deployment** |

The output is similar to:

**deployment "nginx-deployment" successfully rolled out**

Notice that the Deployment has created all three replicas, and all replicas are up-to-date (they contain the latest Pod template) and available.

Run the below command to view the ReplicaSets created by the deployment

|  |
| --- |
| **$ kubectl get rs** |

The output is similar this:

**NAME DESIRED CURRENT READY AGE**

**nginx-deployment-7fd6966748 3 3 3 4m8s**

Run the below command to ensure the created ReplicaSet has three nginx Pods.

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

The output is similar this:

**NAME READY STATUS RESTARTS AGE LABELS**

**nginx-deployment-7fd6966748-8fkq2 1/1 Running 0 8m1s app=nginx,pod-template-hash=7fd6966748**

**nginx-deployment-7fd6966748-pwzwk 1/1 Running 0 8m1s app=nginx,pod-template-hash=7fd6966748**

**nginx-deployment-7fd6966748-r5l8l 1/1 Running 0 8m1s app=nginx,pod-template-hash=7fd6966748**

## **Updating a Deployment**

Follow the steps below to update the deployment:

Let's update the nginx Pods to use the nginx:1.16.1 image instead of the nginx:1.14.2 image.

Run the below command the update the nginx image

|  |
| --- |
| **$ kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1 --record** |

The output is similar to this:

**deployment.extensions/nginx-deployment image updated**

Run the below command to view the rollout status of an updated deployment

|  |
| --- |
| **$ kubectl rollout status deployment.v1.apps/nginx-deployment** |

The output is similar to this:

**deployment "nginx-deployment" successfully rolled out**

Run ***kubectl get rs*** to see that the Deployment updated the Pods by creating a new ReplicaSet and scaling it up to 3 replicas, as well as scaling down the old ReplicaSet to 0 replicas

|  |
| --- |
| **$ kubectl get rs** |

The output is similar to this:

**NAME DESIRED CURRENT READY AGE**

**nginx-deployment-6f9d665859 3 3 3 8m45s**

**nginx-deployment-7fd6966748 0 0 0 30m**

Run the below command to view the details of the deployment

|  |
| --- |
| **$ kubectl deployment describe nginx-depoyment** |

The output is similar to this:

**Name: nginx-deployment**

**Namespace: default**

**CreationTimestamp: Wed, 01 Jul 2020 17:32:53 +0000**

**Labels: app=nginx**

**Annotations: deployment.kubernetes.io/revision: 2**

**kubectl.kubernetes.io/last-applied-configuration:**

**{"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"app":"nginx"},"name":"nginx-deployment","namespace":"d...**

**kubernetes.io/change-cause: kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1 --record=true**

**Selector: app=nginx**

**Replicas: 3 desired | 3 updated | 3 total | 3 available | 0 unavailable**

**StrategyType: RollingUpdate**

**MinReadySeconds: 0**

**RollingUpdateStrategy: 25% max unavailable, 25% max surge**

**Pod Template:**

**Labels: app=nginx**

**Containers:**

**nginx:**

**Image: nginx:1.16.1**

**Port: 80/TCP**

**Host Port: 0/TCP**

**Environment: <none>**

**Mounts: <none>**

**Volumes: <none>**

**Conditions:**

**Type Status Reason**

**---- ------ ------**

**Available True MinimumReplicasAvailable**

**Progressing True NewReplicaSetAvailable**

**OldReplicaSets: <none>**

**NewReplicaSet: nginx-deployment-6f9d665859 (3/3 replicas created)**

**Events:**

**Type Reason Age From Message**

**---- ------ ---- ---- -------**

**Normal ScalingReplicaSet 35m deployment-controller Scaled up replica set nginx-deployment-7fd6966748 to 3**

**Normal ScalingReplicaSet 13m deployment-controller Scaled up replica set nginx-deployment-6f9d665859 to 1**

**Normal ScalingReplicaSet 13m deployment-controller Scaled down replica set nginx-deployment-7fd6966748 to 2**

**Normal ScalingReplicaSet 13m deployment-controller Scaled up replica set nginx-deployment-6f9d665859 to 2**

**Normal ScalingReplicaSet 13m deployment-controller Scaled down replica set nginx-deployment-7fd6966748 to 1**

**Normal ScalingReplicaSet 13m deployment-controller Scaled up replica set nginx-deployment-6f9d665859 to 3**

**Normal ScalingReplicaSet 13m deployment-controller Scaled down replica set nginx-deployment-7fd6966748 to 0**

## **Checking Rollout History of a Deployment**

Run the below command to view the deployment rollout history

|  |
| --- |
| **$ kubectl rollout history deployment nginx-depoyment** |

The output is similar to this:

**deployment.apps/nginx-deployment**

**REVISION CHANGE-CAUSE**

**2 kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1 --record=true**

**3 kubectl apply --filename=nginx-deployment.yml --record=true**

## **Rolling Back to a Previous Version of Deployment**

Run the below command to view the deployment rollout history

|  |
| --- |
| **$ kubectl rollout undo deployment.v1.apps/nginx-deployment –to-revision=2** |

The output is similar to this:

**deployment.apps/nginx-deployment rolled back**

**root@kmaster:~/k8s\_rollingupdate# kubectl rollout history deployment.v1.apps/nginx-deployment**

**deployment.apps/nginx-deployment**

**REVISION CHANGE-CAUSE**

**3 kubectl apply --filename=nginx-deployment.yml --record=true**

**4 kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1 --record=true**

## **Pausing and Resuming a Deployment**

You can pause a Deployment before triggering one or more updates and then resume it. This allows you to apply multiple fixes in between pausing and resuming without triggering unnecessary rollouts.

Pause by running the following command:

|  |
| --- |
| **$ kubectl rollout pause deployment.v1.apps/nginx-deployment** |

The output is similar to this:

**deployment.apps/nginx-deployment paused**

Eventually, resume the Deployment by running the below command and observe a new ReplicaSet coming up with all the new updates which has been made pausing

|  |
| --- |
| **$ kubectl rollout resume deployment.v1.apps/nginx-deployment** |

## **StatefulSets Update**

* StatefulSets is a Kubernetes resource used to manages stateful applications.
* It manages the deployment and scaling of set of Pods, and provides guarantee about the ordering and uniqueness of these Pods.
* StatefulSets is also a Controller but unlike Deployment, it doesn’t create ReplicaSet rather itself creates Pods with a unique naming convention.
* Every replica of a StatefulSet will have its own state, and each of the pods will be creating its own PVC (Persistent Volume Claim). So a StatefulSet with 3 replicas will create 3 pods, each having its own Volume, so total 3 PVCs.

We can begin by creating a StatefulSet using the example below. It creates a [headless Service](https://kubernetes.io/docs/concepts/services-networking/service/#headless-services), nginx, to publish the IP addresses of Pods in the StatefulSet, web.

## **Creating a StatefulSet**

Save the content to a file called **nginx-statefulset.yml**

|  |
| --- |
| **apiVersion: v1**  **kind: Service**  **metadata:**  **name: nginx**  **labels:**  **app: nginx**  **spec:**  **ports:**  **- port: 80**  **name: web**  **clusterIP: None**  **selector:**  **app: nginx**  **---**  **apiVersion: apps/v1**  **kind: StatefulSet**  **metadata:**  **name: web**  **spec:**  **serviceName: "nginx"**  **replicas: 2**  **selector:**  **matchLabels:**  **app: nginx**  **template:**  **metadata:**  **labels:**  **app: nginx**  **spec:**  **containers:**  **- name: nginx**  **image: k8s.gcr.io/nginx-slim:0.8**  **ports:**  **- containerPort: 80**  **name: web**  **volumeMounts:**  **- name: www**  **mountPath: /usr/share/nginx/html**  **volumeClaimTemplates:**  **- metadata:**  **name: www**  **spec:**  **accessModes: [ "ReadWriteOnce" ]**  **resources:**  **requests:**  **storage: 1Gi** |

Run the below command [kubectl apply](https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands/#apply) to create the headless Service and StatefulSet defined in **nginx-statefulset.yml**.

|  |
| --- |
| **$ kubectl apply –f nginx-statefulset.yml** |

The output is similar to this:

**service/nginx created**

**statefulset.apps/web created**

## **Examining a StatefulSet**

Pods in a StatefulSet have a sticky, unique identity. This identity is based on a unique ordinal index that is assigned to each Pod by the StatefulSet [controller](https://kubernetes.io/docs/concepts/architecture/controller/).

Run the below command to get the StatefulSet’s Pods

|  |
| --- |
| **$ kubectl get pods –l app=nginx** |

The output is similar to this:

**NAME READY STATUS RESTARTS AGE**

**web-0 1/1 Running 0 1m**

**web-1 1/1 Running 0 1m**

## **Updating StatefulSet**

Updating StatefulSets feature can be used to upgrade the container images, resource requests and/or limits, labels, and annotations of the Pods in a StatefulSet. There are two valid update strategies, **RollingUpdate** and **OnDelete**

The *RollingUpdate* update strategy will update all Pods in a StatefulSet, in reverse ordinal order, while respecting the StatefulSet guarantees.

The *OnDelete* update strategy will not automatically update the Pods in a StatefulSet. Users must manually delete Pods to cause the controller to create new Pods that reflect modifications made to a StatefulSet's .spec.template

Patch the web StatefulSet to apply the RollingUpdate update strategy:

|  |
| --- |
| **$ kubectl patch statefulset web -p '{"spec":{"updateStrategy":{"type":"RollingUpdate"}}}'** |

The output is similar to this:

**statefulset.apps/web patched**

# **References**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Components** | **Reference** |
| 1 | Screenshots document | 5.CI\_CD\_Deployment\_using\_Rolling\_updates\_screenshots.docx |
| 2 | Sources document | 5.CI\_CD\_Deployment\_using\_Rolling\_updates\_sources.docx |
| 3 | Kubernetes.io | <https://kubernetes.io/docs/> |