Persistent storage is **any data storage device that retains data after power to that device is shut off**. It is also sometimes referred to as non-volatile storage. Magnetic media, such as hard disk drives and tape are common types of persistent storage, as are the various forms of Optical media such as DVD.

We can install and set up Jenkins on different platforms like:

- *Jenkins on On-Premise Bare Server*
- *Jenkins on On-Premise Virtual Machine*
- *Jenkins on Cloud Virtual Machine*
- *Jenkins on Docker*

In this section, we are going to learn how can we deploy Jenkins on Kubernetes. For this blog, I'm using Google Kubernetes Engine (GKE) as it's super easy to set up and manage.

We can set up GKE Cluster in few clicks or CLI commands. Please check here to deploy the GKE Cluster.

Major challenge in running Jenkins on K8S: Pods running on K8S having a major challenge related to persistent storage. If we don't define any persistent volume in storage of K8S Deployment then we'll lose our data once pods are recreated.

Jenkins stores all of it's data
in $JENKINS_HOME directory. $JENKINS_HOME is where all Jenkins-based installations store configuration, build logs, and artifacts, custom plugins etc.

To handle this problem, we need to configure **a persistent volume using GCP provided Storage Classes**.

- pd-standard — standard persistent disk
- pd-ssd — premium SSD persistent disk
- pd-balanced — standard balanced persistent disk

How To Setup Jenkins on Kubernetes Cluster as a Pod
Hosting Jenkins on a Kubernetes cluster is beneficial for Kubernetes-based deployments and dynamic container-based scalable Jenkins agents.

Setup Jenkins On Kubernetes Cluster

For setting up a jenkins cluster on Kubernetes, we will do the following.

1. Create a Namespace
2. Create a service account with Kubernetes admin permissions.
3. Create persistent volume for persistent Jenkins's data on google Kubernetes cluster.
4. Create a deployment YAML and deploy it.
5. Create a service YAML and deploy it.
6. Access the Jenkins application on a Node Port.

Step 1: Create a Namespace for Jenkins. It is good to categorize all the devops tools as a separate namespace from other applications.

kubectl create namespace jenkins-dev

Step 2: Create a serviceAccount.yaml file and copy the following admin service account manifest.

```
---
        apiVersion: rbac.authorization.k8s.io/v1
        kind: ClusterRole
        metadata:
          name: jenkins-admin
        rules:
          - apiGroups: [""]
            resources: ["*"]
            verbs: ["*"]

        ---
        apiVersion: v1
        kind: ServiceAccount
        metadata:
          name: jenkins-admin
          namespace: jenkins-dev

        ---
        apiVersion: rbac.authorization.k8s.io/v1
        kind: ClusterRoleBinding
        metadata:
          name: jenkins-admin
        roleRef:
          apiGroup: rbac.authorization.k8s.io
          kind: ClusterRole
          name: jenkins-admin
        subjects:
        - kind: ServiceAccount
          name: jenkins-admin
          namespace: jenkins-dev
```

The **serviceAccount.yaml** creates a **jenkins-admin clusterRole**, **jenkins-admin ServiceAccount** and binds the **clusterRole** to the service account.

The **jenkins-admin** cluster role has all the permissions to manage the cluster components. You can also restrict access by specifying individual resource actions.

Now create the service account using kubectl.

kubectl apply -f serviceAccount.yaml

Step 3: Create volume.yaml and copy the following persistent volume manifest.

```
---
        apiVersion: v1
        kind: PersistentVolumeClaim
        metadata:
          annotations:
          name: jenkins-pvc
          namespace: jenkins-dev
        spec:
          accessModes:
          - ReadWriteOnce
          resources:
            requests:
              storage: 80Gi
          storageClassName: pd-ssd
          volumeMode: Filesystem
```

Kubectl apply -f volume.yaml

Jenkins K8S Deployment:

We now create a K8S Deployment which would use the above created PVC to persist the data in $JENKINS_HOME

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: jenkins
namespace: jenkins-dev
spec:
replicas: 1
selector:
matchLabels:
app: jenkins
template:
metadata:
labels:
app: jenkins
spec:
containers:
- image: jenkins/jenkins:lts
imagePullPolicy: Always
name: container-0
ports:
- name: http-port
containerPort: 8080
- name: jnlp-port
containerPort: 50000
securityContext:
allowPrivilegeEscalation: true
privileged: true
readOnlyRootFilesystem: false
runAsUser: 0
volumeMounts:
- mountPath: /var/jenkins_home
name: jenkins-vol
volumes:
- name: jenkins-vol
persistentVolumeClaim:
claimName: jenkins-pvc
```

```
# kubectl create -f deployment.yaml
```

Expose Deployment using K8S Services:

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: jenkins
  namespace: jenkins-dev
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30000
  selector:
    app: jenkins---apiVersion: v1
kind: Service
metadata:
  name: jenkins-jnlp
  namespace: jenkins-dev
spec:
  type: ClusterIP
  ports:
    - port: 50000
      targetPort: 50000
  selector:
    app: jenkins
```

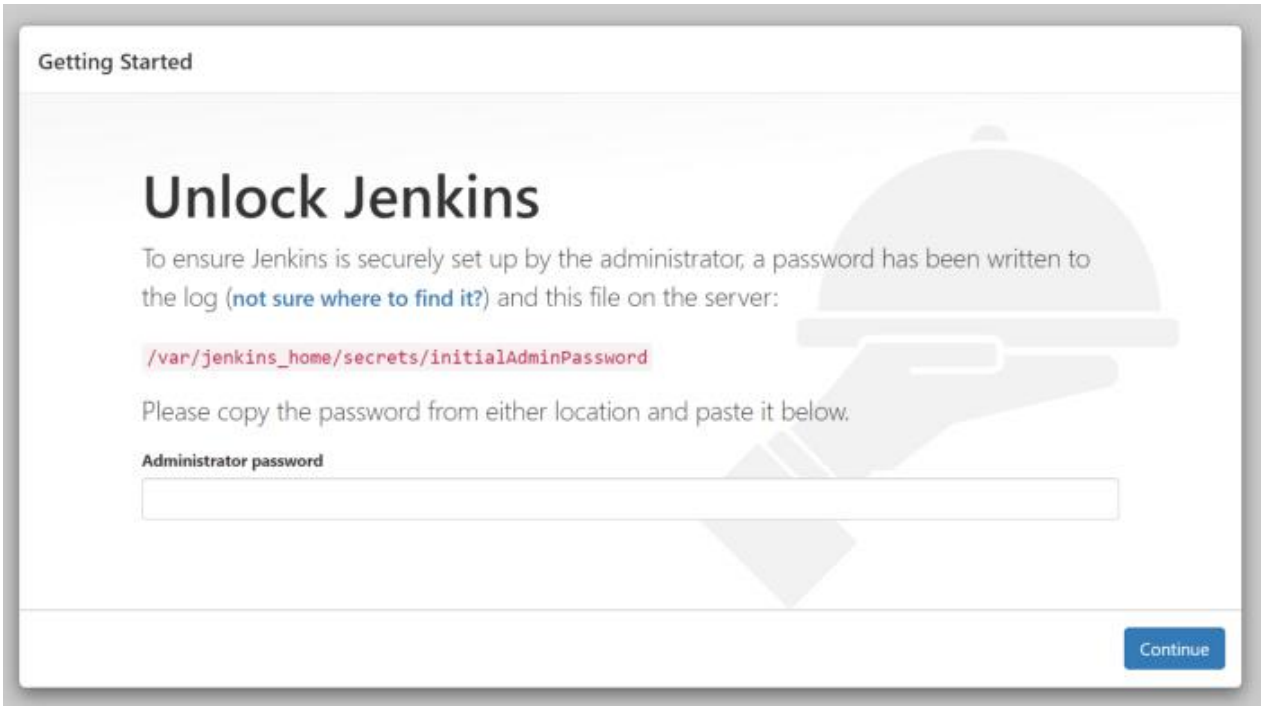
Please check all the deployed config using:

```
kubectl get all -n jenkins-dev
```

To get the NodePort details from kubectl:

```
kubectl get svc -n jenkins-dev
```

Browse the Jenkins using NodePort (http://nodeIPaddress:nodeport), we will get this screen to provide initialadminpassword



We can get the initialadminpassword in the logs using following commands:

```
kubectl get po -n jenkins-dev
kubectl logs <pod-name> -n jenkins-dev
```

We need to setup username, password and domain etc. Then, we are good to log in to Jenkins.