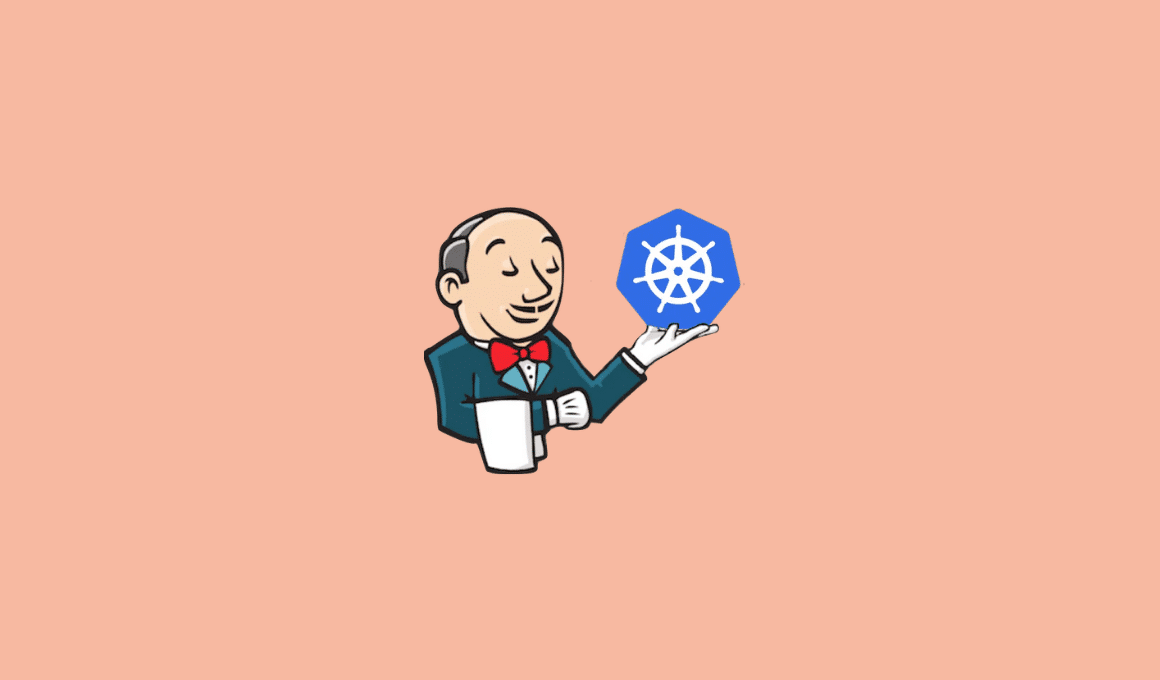
How To Setup Jenkins On Kubernetes Cluster – Beginners Guide

* by[**Bibin Wilson**](https://devopscube.com/author/bibinwilson/)
* May 9, 2021

[](https://devopscube.com/wp-content/uploads/2017/10/Kubernetes-on-Jenkins.png)

**16**

SHARES

Hosting Jenkins on a [Kubernetes cluster](https://devopscube.com/setup-kubernetes-cluster-kubeadm/) is beneficial for Kubernetes-based deployments and dynamic container-based scalable Jenkins agents.

In this guide, I have explained the step-by-step process for setting up Jenkins on a [Kubernetes cluster.](https://devopscube.com/setup-kubernetes-cluster-google-cloud/)

**Setup Jenkins On Kubernetes Cluster**

For setting up a [Jenkins](https://devopscube.com/jenkins-2-tutorials-getting-started-guide/) cluster on Kubernetes, we will do the following.

1. Create a Namespace
2. Create a service account with Kubernetes admin permissions.
3. Create local persistent volume for persistent Jenkins data on Pod restarts.
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.

**Note:** This tutorial doesn’t use local persistent volume as this is a generic guide. For using persistent volume for your Jenkins data, you need to create volumes of relevant cloud or on-prem data center and configure it.

**Jenkins Kubernetes Manifest Files**

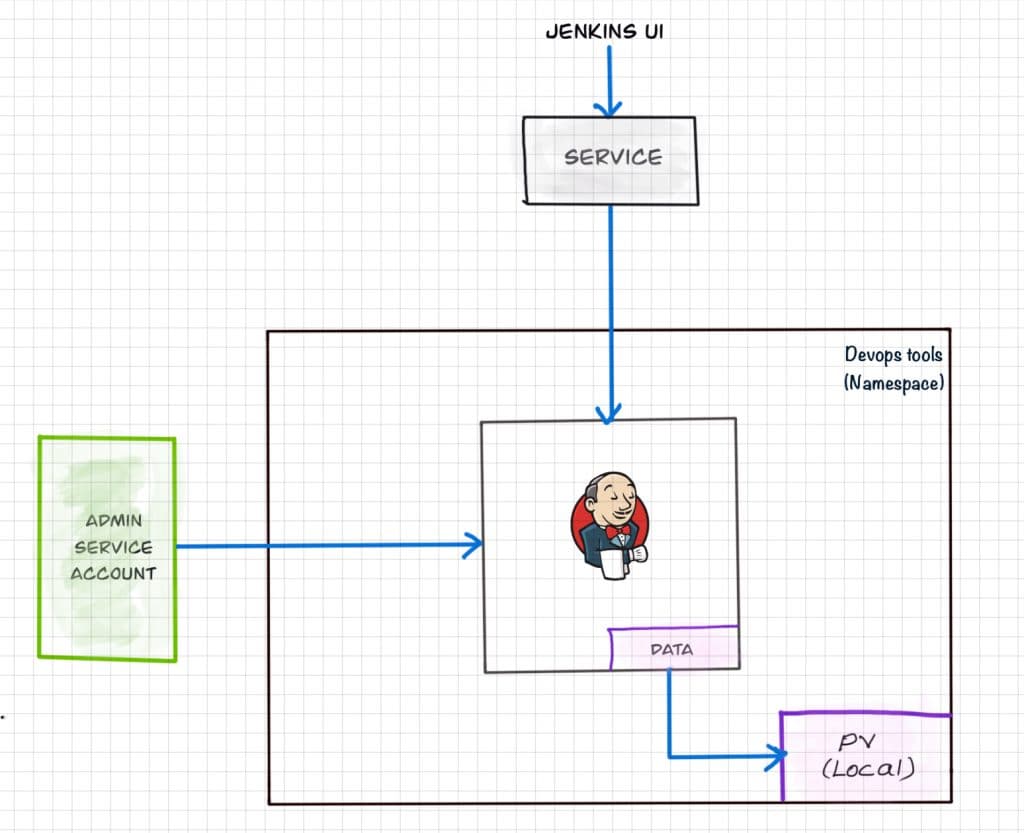
All the Jenkins Kubernetes manifest files used in this blog are [hosted on Github](https://github.com/scriptcamp/kubernetes-jenkins). Please clone the repository if you have trouble copying the manifest from the blog.

git clone https://github.com/scriptcamp/kubernetes-jenkins

Use the Github files for reference and follow the steps in the next sections

**Kubernetes Jenkins Deployment**

Here is the high level view of what we are going to do.

[](https://devopscube.com/wp-content/uploads/2021/05/IMG_0184.jpg)

Lets get started with deploying Jenkins on Kubernetes.

**Step 1**: Create a Namespace for Jenkins. It is good to categorize all the [devops tools](https://devopscube.com/devops-tools-for-infrastructure-automation/) as a separate namespace from other applications.

kubectl create namespace devops-tools

**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: devops-tools

---

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: devops-tools

The serviceAccount.yaml creates a jenkins-admin clusterRole, jenkins-admin [ServiceAccount](https://devopscube.com/kubernetes-api-access-service-account/) 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 a volume.yaml and copy the following persistent volume manifest.

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: local-storage

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: jenkins-pv-volume

labels:

type: local

spec:

storageClassName: local-storage

claimRef:

name: jenkins-pv-claim

namespace: devops-tools

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

local:

path: /mnt

nodeAffinity:

required:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

- worker-node01

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: jenkins-pv-claim

namespace: devops-tools

spec:

storageClassName: local-storage

accessModes:

- ReadWriteOnce

resources:

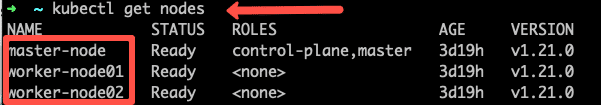
requests:

storage: 3Gi

**Important Note:** Replace worker-node01 with any one of your cluster worker nodes hostname.

You can get the worker node hostname using the kubectl.

kubectl get nodes

[](https://devopscube.com/wp-content/uploads/2021/05/nodes.png)

For volume, I have used the local storage class for the purpose of demonstration. Meaning, it creates a PersistentVolume volume in a specific node under /mnt location.

As the local storage class requires the node selector, you need to specify the worker node name correctly for the Jenkins pod to get scheduled in the specific node.

If the pod gets deleted or restarted, the data will get persisted in the node volume. However, if the node gets deleted, you will lose all the data.

Ideally, you should use a persistent volume using the available storage class with the cloud provider or the one provided by the cluster administrator to persist data on node failures.

Lets create the volume using kubectl

kubectl create -f volume.yaml

**Step 2:** Create a Deployment file named deployment.yaml and copy the following deployment manifest.

Here we are using the latest [Jenkins LTS docker image](https://hub.docker.com/r/jenkins/jenkins) from the Docker hub.

apiVersion: apps/v1

kind: Deployment

metadata:

name: jenkins

namespace: devops-tools

spec:

replicas: 1

selector:

matchLabels:

app: jenkins-server

template:

metadata:

labels:

app: jenkins-server

spec:

securityContext:

fsGroup: 1000

runAsUser: 1000

serviceAccountName: jenkins-admin

containers:

- name: jenkins

image: jenkins/jenkins:lts

resources:

limits:

memory: "2Gi"

cpu: "1000m"

requests:

memory: "500Mi"

cpu: "500m"

ports:

- name: httpport

containerPort: 8080

- name: jnlpport

containerPort: 50000

livenessProbe:

httpGet:

path: "/login"

port: 8080

initialDelaySeconds: 90

periodSeconds: 10

timeoutSeconds: 5

failureThreshold: 5

readinessProbe:

httpGet:

path: "/login"

port: 8080

initialDelaySeconds: 60

periodSeconds: 10

timeoutSeconds: 5

failureThreshold: 3

volumeMounts:

- name: jenkins-data

mountPath: /var/jenkins\_home

volumes:

- name: jenkins-data

persistentVolumeClaim:

claimName: jenkins-pv-claim

In this Jenkins [Kubernetes deployment](https://devopscube.com/kubernetes-deployment-tutorial/) we have used the following.

1. securityContext for Jenkins pod to be able to write to the local persistent volume.
2. Liveliness and readiness probe.
3. Local persistent volume based on local storage class that holds the Jenkins data path /var/jenkins\_home

**Note**: The deployment file uses local storage class persistent volume for Jenkins data. For production use cases, you should add a cloud-specific storage class persistent volume for your Jenkins data. See the sample implementation of [persistent volume for Jenkins in Google Kubernetes Engine](https://devopscube.com/persistent-volume-google-kubernetes-engine/)

If you don’t want the local storage persistent volume, you can replace the volume definition in the deployment with the host directory as shown below.

volumes:

- name: jenkins-data

emptyDir: {}

Create the deployment using kubectl.

kubectl apply -f deployment.yaml

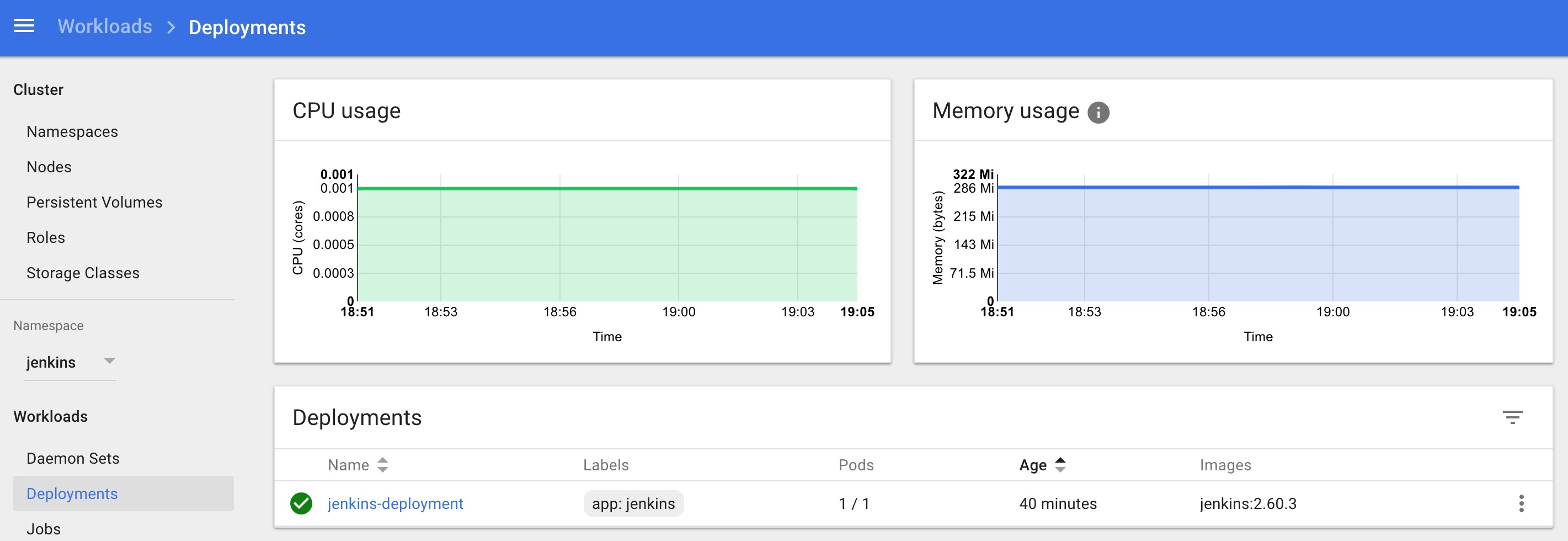
Check the deployment status.

kubectl get deployments -n devops-tools

Now, you can get the deployment details using the following command.

kubectl describe deployments --namespace=jenkins

Also, You can get the details from the kubernetes dashboard as shown below.

[](https://devopscube.com/wp-content/uploads/2017/10/Screen-Shot-2017-10-08-at-7.05.25-PM.jpg)

**Accessing Jenkins Using Kubernetes Service**

We have created a deployment. However, it is not accessible to the outside world. For accessing the Jenkins deployment from the outside world, we should create a service and map it to the deployment.

**Step 1:** Create a service.yaml and copy the following service manifest.

apiVersion: v1

kind: Service

metadata:

name: jenkins-service

namespace: devops-tools

annotations:

prometheus.io/scrape: 'true'

prometheus.io/path: /

prometheus.io/port: '8080'

spec:

selector:

app: jenkins-server

type: NodePort

ports:

- port: 8080

targetPort: 8080

nodePort: 32000

**Note:** Here, we are using the type as NodePort which will expose Jenkins on all kubernetes node IPs on port 32000. If you have an [ingress setup](https://devopscube.com/setup-ingress-kubernetes-nginx-controller/), you can create an ingress rule to access Jenkins. Also, you can expose the Jenkins service as a Loadbalancer if you are running the cluster on AWS, Google, or Azure cloud.

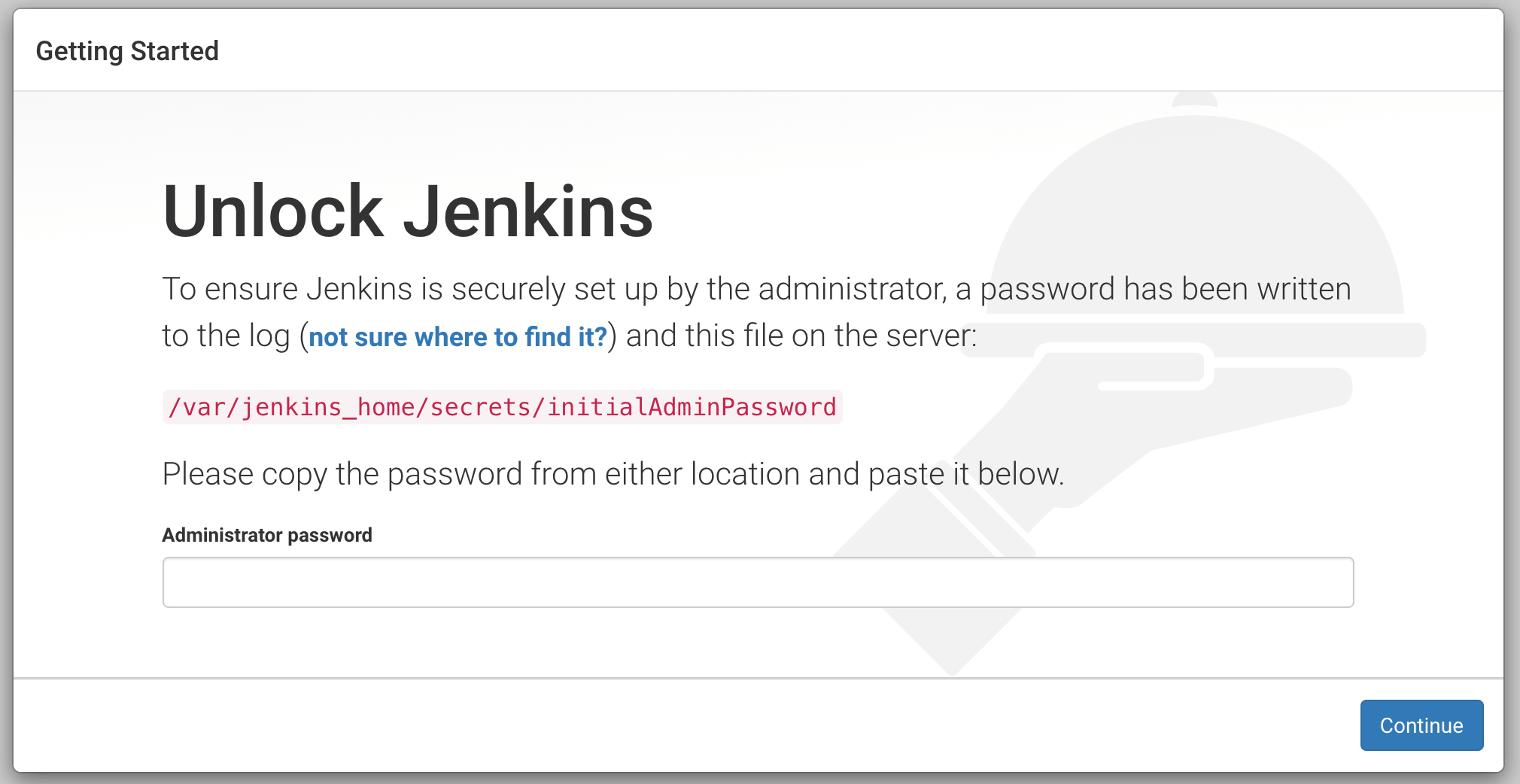
Create the Jenkins service using kubectl.

kubectl apply -f service.yaml

Now if you browse to any one of the Node IPs on port 32000, you will be able to access the Jenkins dashboard.

http://<node-ip>:32000

Jenkins will ask for the initial Admin password when you access the dashbaord for the first time.

[](https://devopscube.com/wp-content/uploads/2021/05/Jenkins-dashbaord.png)

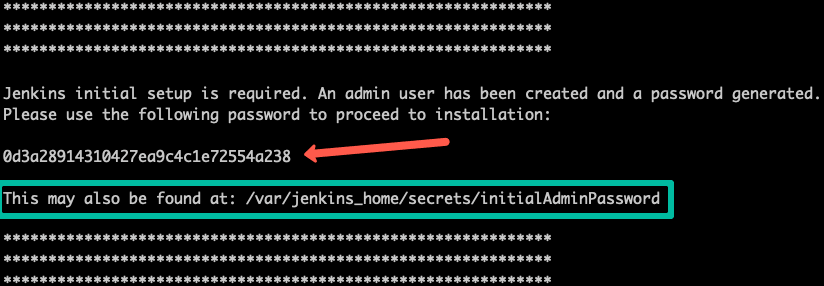
You can get that from the pod logs either from the kubernetes dashboard or  CLI. You can get the pod details using the following CLI command.

kubectl get pods --namespace=devops-tools

And with the pod name, you can get the logs as shown below. replace the pod name with your pod name.

kubectl logs jenkins-deployment-2539456353-j00w5 --namespace=jenkins

The password can be found at the end of the log as shown below.

[](https://devopscube.com/wp-content/uploads/2021/05/jenkins-password-min.png)

Alternatively, you can run the exec command to get the password directly from the location as show below.

kubectl exec -it jenkins-559d8cd85c-cfcgk cat /var/jenkins\_home/secrets/initialAdminPassword -n devops-tools

Once you enter the password you can proceed installing the suggested plugin and creating a admin user. All these steps are self-explanatory from the Jenkins dashboard.

**Conclusion**

When you host Jenkins on Kubernetes for production workloads, you need to con sider setting up a highly available persistent volume to avoid data loss during pod or or node deletetion.

A pod or node deletion could happen anytime in Kubernetes environments. It could be a patching activity or a downscaling activity.

Hope this step by step guide helps you to learn and understand the components involved in setting up a Jenkins server on a Kubernetes cluster.