Kubernetes

Table of Contents

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

A Kubernetes cluster adds a new automation layer to Jenkins. Kubernetes makes sure that resources are used effectively and that your servers and underlying infrastructure are not overloaded. Kubernetes' ability to orchestrate container deployment ensures that Jenkins always has the right amount of resources available.

Hosting Jenkins on a Kubernetes Cluster is beneficial for Kubernetes-based deployments and dynamic container-based scalable Jenkins agents. Here, we see a step-by-step process for setting up Jenkins on a Kubernetes Cluster.

Setup Jenkins On Kubernetes

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 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.

This guide 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 here are hosted on GitHub. Please clone the repository if you have trouble copying the manifest from the document.

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

bash Copied!

Use the GitHub files for reference and follow the steps in the next sections.

Kubernetes Jenkins Deployment

Let's get started with deploying Jenkins on Kubernetes.

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 devops-tools
```

bash Copied!

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
```

yaml 

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
```

bash 

Step 3: Create '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
```

yaml 

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
```

bash 

For volume, we are using the 'local' storage class for the purpose of demonstration. Meaning, it creates a 'PersistentVolume' volume in a specific node under the '/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.

Let’s create the volume using kubectl

```
kubectl create -f volume.yaml
```

bash 

Step 4: Create a Deployment file named 'deployment.yaml' and copy the following deployment manifest.

```
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
```

yaml 

In this Jenkins Kubernetes deployment we have used the following:

1. 'securityContext' for Jenkins pod to be able to write to the local persistent volume.
2. Liveness and readiness probe to monitor the health of the Jenkins pod.
3. Local persistent volume based on local storage class that holds the Jenkins data path '/var/jenkins_home'.

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.

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: \{}
```

yaml 

Create the deployment using kubectl.

```
kubectl apply -f deployment.yaml
```

bash Copied!

Check the deployment status.

```
kubectl get deployments -n devops-tools
```

bash Copied!

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

```
kubectl describe deployments --namespace=devops-tools
```

bash Copied!

Accessing Jenkins Using Kubernetes Service

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

Create '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
```

yaml Copied!

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, 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
```

bash Copied!

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

```
http://<node-ip>:32000
```

Copied!

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

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
```

bash Copied!

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=devops-tools
```

bash Copied!

The password can be found at the end of the log.

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

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

bash Copied!

Once you enter the password, proceed to install the suggested plugin and create an admin user. All of these steps are self-explanatory from the Jenkins dashboard.

Install Jenkins with Helm v3

A typical Jenkins deployment consists of a controller node and, optionally, one or more agents. To simplify the deployment of Jenkins, we'll use Helm to deploy Jenkins. Helm is a package manager for Kubernetes and its package format is called a chart. Many community-developed charts are available on GitHub.

Helm Charts provide "push button" deployment and deletion of apps, making adoption and development of Kubernetes apps easier for those with little container or microservices experience.

Prerequisites

Helm command line interface

> If you don't have Helm command line interface installed and configured locally, see the sections below to Install Helm and Configure Helm.

Install Helm

To install Helm CLI, follow the instructions from the Installing Helm page.

Configure Helm

Once Helm is installed and set up properly, add the Jenkins repo as follows:

```
helm repo add jenkinsci https://charts.jenkins.io
helm repo update
```

bash Copied!

The helm charts in the Jenkins repo can be listed with the command:

```
helm search repo jenkinsci
```

bash Copied!

Create a persistent volume

We want to create a persistent volume for our Jenkins controller pod. This will prevent us from losing our whole configuration of the Jenkins controller and our jobs when we reboot our minikube. This official minikube doc explains which directories we can use to mount or data. In a multi-node Kubernetes cluster, you'll need some solution like NFS to make the mount directory available in the whole cluster. But because we use minikube which is a one-node cluster we don't have to bother about it.

We choose to use the `/data` directory. This directory will contain our Jenkins controller configuration.

We will create a volume which is called jenkins-pv:

1. Paste the content from https://raw.githubusercontent.com/installing-jenkins-on-kubernetes/jenkins-volume.yaml into a YAML formatted file called `jenkins-volume.yaml`.
2. Run the following command to apply the spec:

   ```
   kubectl apply -f jenkins-volume.yaml
   ```

   bash 

It's worth noting that, in the above spec, hostPath uses the /data/jenkins-volume/ of your node to emulate network-attached storage. This approach is only suited for development and testing purposes. For production, you should provide a network resource like a Google Compute Engine persistent disk, or an Amazon Elastic Block Store volume.

Minikube configured for hostPath sets the permissions on /data to the root account only. Once the volume is created you will need to manually change the permissions to allow the jenkins account to write its data.

```
minikube ssh
sudo chown -R 1000:1000 /data/jenkins-volume
```

bash 

Create a service account

In Kubernetes, service accounts are used to provide an identity for pods. Pods that want to interact with the API server will authenticate with a particular service account. By default, applications will authenticate as the `default` service account in the namespace they are running in. This means, for example, that an application running in the `test` namespace will use the default service account of the `test` namespace.

We will create a service account called jenkins:

A ClusterRole is a set of permissions that can be assigned to resources within a given cluster. Kubernetes APIs are categorized into API groups, based on the API objects that they relate to. While creating a ClusterRole, you can specify the operations that can be performed by the ClusterRole on one or more API objects in one or more API groups, just as we have done above. ClusterRoles have several uses. You can use a ClusterRole to:

- define permissions on namespaced resources and be granted within individual namespace(s)
- define permissions on namespaced resources and be granted across all namespaces
- define permissions on cluster-scoped resources

If you want to define a role cluster-wide, use a ClusterRole; if you want to define a role within a namespace, use a Role.

A role binding grants the permissions defined in a role to a user or set of users. It holds a list of subjects (users, groups, or service accounts), and a reference to the role being granted.

A RoleBinding may reference any Role in the same namespace. Alternatively, a RoleBinding can reference a ClusterRole and bind that ClusterRole to the namespace of the RoleBinding. To bind a ClusterRole to all the namespaces in our cluster, we use a ClusterRoleBinding.

1. Paste the content from https://raw.githubusercontent.com/installing-jenkins-on-kubernetes/jenkins-sa.yaml into a YAML formatted file called `jenkins-sa.yaml`.

2. Run the following command to apply the spec:

   ```
   kubectl apply -f jenkins-sa.yaml
   ```

   bash 

Install Jenkins

We will deploy Jenkins including the Jenkins Kubernetes plugin. See the official chart for more details.

1. To enable persistence, we will create an override file and pass it as an argument to the Helm CLI. Paste the content from raw.githubusercontent.com/jenkinsci/helm-charts/main/charts/jenkins/values.yaml into a YAML formatted file called `jenkins-values.yaml`.

   The `jenkins-values.yaml` is used as a template to provide values that are necessary for setup.

2. Open the `jenkins-values.yaml` file in your favorite text editor and modify the following:

   - nodePort: Because we are using minikube we need to use NodePort as service type. Only cloud providers offer load balancers. We define port 32000 as port.

   - storageClass:

     ```
     storageClass: jenkins-pv
     ```

     yaml 

   - serviceAccount: the serviceAccount section of the jenkins-values.yaml file should look like this:

     ```
     serviceAccount:
       create: false
     # Service account name is autogenerated by default
     name: jenkins
     annotations: {}
     ```

     yaml 


     ```
     Where `name: jenkins` refers to the serviceAccount created for jenkins.
     ```

   - We can also define which plugins we want to install on our Jenkins. We use some default plugins like git and the pipeline plugin.

3. Now you can install Jenkins by running the `helm install` command and passing it the following arguments:

   - The name of the release `jenkins`
   - The `-f` flag with the YAML file with overrides `jenkins-values.yaml`
   - The name of the chart `jenkinsci/jenkins`
   - The `-n` flag with the name of your namespace `jenkins`

     ```
     chart=jenkinsci/jenkins
     helm install jenkins -n jenkins -f jenkins-values.yaml $chart
     ```

     bash 


     ```
     This outputs something similar to the following:
     ```

```
NAME: jenkins
LAST DEPLOYED: Wed Sep 16 11:13:10 2020
NAMESPACE: jenkins
STATUS: deployed
REVISION: 1
```

bash Copied!

1. Get your 'admin' user password by running:

```
jsonpath="{.data.jenkins-admin-password}"
secret=$(kubectl get secret -n jenkins jenkins -o jsonpath=$jsonpath)
echo $(echo $secret | base64 --decode)
```

bash Copied!

2. Get the Jenkins URL to visit by running these commands in the same shell:

```
jsonpath="{.spec.ports[0].nodePort}"
NODE_PORT=$(kubectl get -n jenkins -o jsonpath=$jsonpath services jenkins)
jsonpath="{.items[0].status.addresses[0].address}"
NODE_IP=$(kubectl get nodes -n jenkins -o jsonpath=$jsonpath)
echo http://$NODE_IP:$NODE_PORT/login
```

bash Copied!

3. Login with the password from step 1 and the username: admin

4. Use Jenkins Configuration as Code by specifying configScripts in your values.yaml file. See the configuration as code documentation and examples.

Visit the Jenkins on Kubernetes solutions page for more information on running Jenkins on Kubernetes. Visit the Jenkins Configuration as Code project for more information on configuration as code. . Depending on your environment, it can take a bit of time for Jenkins to start up. Enter the following command to inspect the status of your Pod:

```
kubectl get pods -n jenkins
```

bash Copied!

Once Jenkins is installed, the status should be set to Running as in the following output:

```
kubectl get pods -n jenkins
NAME                       READY   STATUS    RESTARTS   AGE
jenkins-645fbf58d6-6xfvj   1/1     Running   0          2m
```

bash Copied!

1. To access your Jenkins server, you must retrieve the password. You can retrieve your password using either of the two options below.

   Option 1

   Run the following command:

```
jsonpath="{.data.jenkins-admin-password}"
secret=$(kubectl get secret -n jenkins jenkins -o jsonpath=$jsonpath)
echo $(echo $secret | base64 --decode)
```

   bash Copied!

   The output should look like this:

```
Um1kJLOWQY
```

Copied!

Note that your password will be different.

Option 2

Run the following command:

```
jsonpath="{.data.jenkins-admin-password}"
kubectl get secret -n jenkins jenkins -o jsonpath=$jsonpath
```

bash Copied!

The output should be a **base64 encoded string** like this:

```
WkIwRkdnbDZYZg==
```

Copied!

Decode the base64 string and you have your password. You can use this website to decode your output.

2. Get the name of the Pod running that is running Jenkins using the following command:

```
kubectl get pods -n jenkins
```

bash Copied!

3. Use the kubectl command to set up port forwarding:

```
kubectl -n jenkins port-forward <pod_name> 8080:8080
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

bash Copied!

Visit 127.0.0.1:8080/ and log in using `admin` as the username and the password you retrieved earlier.

Install Jenkins with YAML files

This section describes how to use a set of YAML (Yet Another Markup Language) files to install Jenkins on a Kubernetes cluster. The YAML files are easily tracked, edited, and can be reused indefinitely.

Create Jenkins deployment file

Copy the contents here into your preferred text editor and create a jenkins-deployment.yaml file in the “jenkins” namespace we created in this section above.

- This deployment file is defining a Deployment as indicated by the `kind` field.
- The Deployment specifies a single replica. This ensures one and only one instance will be maintained by the Replication Controller in the event of failure.
- The container image name is jenkins and version is 2.32.2
- The list of ports specified within the spec are a list of ports to expose from the container on the Pods IP address.

    - Jenkins running on (http) port 8080.
    - The Pod exposes the port 8080 of the jenkins container.

- The volumeMounts section of the file creates a Persistent Volume. This volume is mounted within the container at the path /var/jenkins_home and so modifications to data within /var/jenkins_home are written to the volume. The role of a persistent volume is to store basic Jenkins data and preserve it beyond the lifetime of a pod.

Exit and save the changes once you add the content to the Jenkins deployment file.

Deploy Jenkins

To create the deployment execute:

```
kubectl create -f jenkins-deployment.yaml -n jenkins
```

bash Copied!

The command also instructs the system to install Jenkins within the jenkins namespace.

To validate that creating the deployment was successful you can invoke:

```
kubectl get deployments -n jenkins
```

bash Copied!

Grant access to Jenkins service

We have a Jenkins instance deployed but it is still not accessible. The Jenkins Pod has been assigned an IP address that is internal to the Kubernetes cluster. It's possible to log into the Kubernetes Node and access Jenkins from there but that's not a very useful way to access the service.

To make Jenkins accessible outside the Kubernetes cluster the Pod needs to be exposed as a Service. A Service is an abstraction that exposes Jenkins to the wider network. It allows us to maintain a persistent connection to the pod regardless of the changes in the cluster. With a local deployment, this means creating a NodePort service type. A NodePort service type exposes a service on a port on each node in the cluster. The service is accessed through the Node IP address and the service nodePort. A simple service is defined here:

- This service file is defining a Service as indicated by the `kind` field.
- The Service is of type NodePort. Other options are ClusterIP (only accessible within the cluster) and LoadBalancer (IP address assigned by a cloud provider e.g. AWS Elastic IP).
- The list of ports specified within the spec is a list of ports exposed by this service.
    - The port is the port that will be exposed by the service.
    - The target port is the port to access the Pods targeted by this service. A port name may also be specified.
- The selector specifies the selection criteria for the Pods targeted by this service.

To create the service execute:

```
kubectl create -f jenkins-service.yaml -n jenkins
```

bash Copied!

To validate that creating the service was successful you can run:

```
kubectl get services -n jenkins
NAME       TYPE        CLUSTER-IP       EXTERNAL-IP    PORT(S)          AGE
jenkins    NodePort    10.103.31.217    <none>         8080:32664/TCP    59s
```

bash Copied!

Access Jenkins dashboard

So now we have created a deployment and service, how do we access Jenkins?

From the output above we can see that the service has been exposed on port 32664. We also know that because the service is of type NodeType the service will route requests made to any node on this port to the Jenkins pod. All that's left for us is to determine the IP address of the minikube VM. Minikube have made this really simple by including a specific command that outputs the IP address of the running cluster:

```
minikube ip
192.168.99.100
```

bash Copied!

Now we can access the Jenkins instance at 192.168.99.100:32664/

To access Jenkins, you initially need to enter your credentials. The default username for new installations is admin. The password can be obtained in several ways. This example uses the Jenkins deployment pod name.

To find the name of the pod, enter the following command:

```
kubectl get pods -n jenkins
```

bash Copied!

Once you locate the name of the pod, use it to access the pod's logs.

```
kubectl logs <pod_name> -n jenkins
```

bash Copied!

The password is at the end of the log formatted as a long alphanumeric string:

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required.
An admin user has been created and a password generated.
Please use the following password to proceed to installation:

94b73ef6578c4b4692a157f768b2cfef

This may also be found at:
/var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
```

Copied!

You have successfully installed Jenkins on your Kubernetes cluster and can use it to create new and efficient development pipelines.

Install Jenkins with Jenkins Operator

The Jenkins Operator is a Kubernetes native Operator which manages operations for Jenkins on Kubernetes.

It was built with immutability and declarative configuration as code in mind, to automate many of the manual tasks required to deploy and run Jenkins on Kubernetes.

Jenkins Operator is easy to install with applying just a few yaml manifests or with the use of Helm.

For instructions on installing Jenkins Operator on your Kubernetes cluster and deploying and configuring Jenkins there, see official documentation of Jenkins Operator.

Post-installation setup wizard

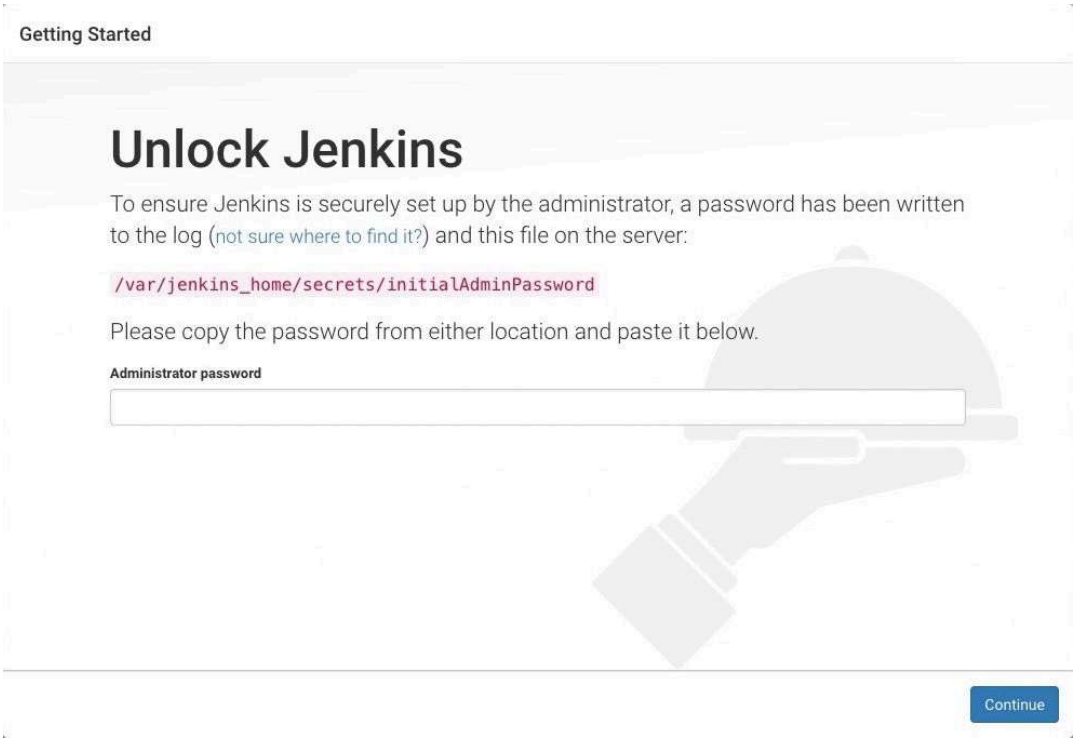
After downloading, installing and running Jenkins using one of the procedures above (except for installation with Jenkins Operator), the post-installation setup wizard begins.

This setup wizard takes you through a few quick "one-off" steps to unlock Jenkins, customize it with plugins and create the first administrator user through which you can continue accessing Jenkins.

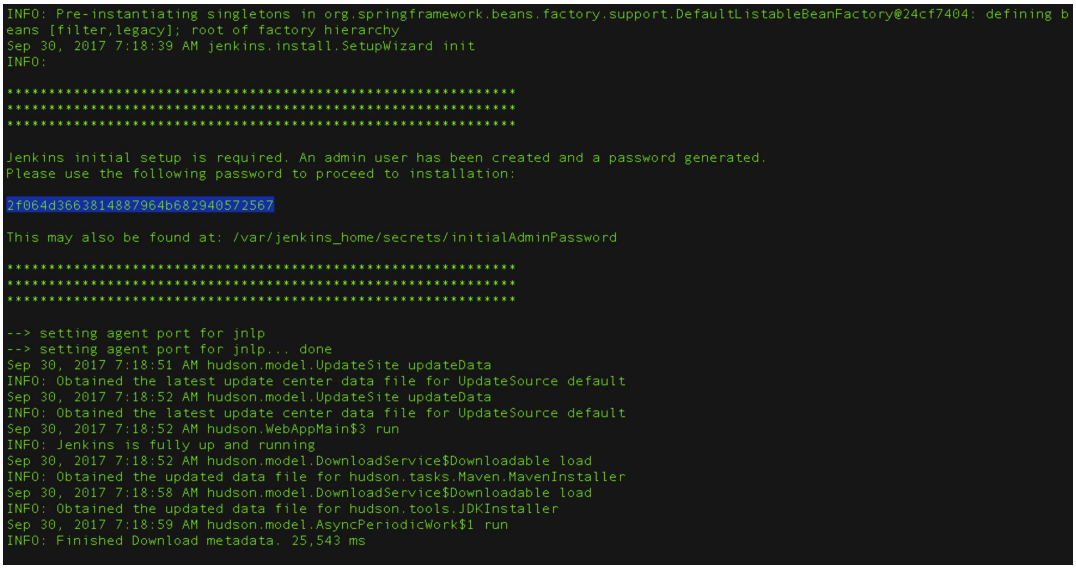
Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically-generated password.

1. Browse to `http://localhost:8080` (or whichever port you configured for Jenkins when installing it) and wait until the **Unlock Jenkins** page appears.



2. From the Jenkins console log output, copy the automatically-generated alphanumeric password (between the 2 sets of asterisks).



Note:

- The command: `sudo cat /var/lib/jenkins/secrets/initialAdminPassword` will print the password at console.
- If you are running Jenkins in Docker using the official `jenkins/jenkins` image you can use `sudo docker exec ${CONTAINER_ID or CONTAINER_NAME} cat /var/jenkins_home/secrets/initialAdminPassword` to print the password in the console without having to exec into the container.

3. On the **Unlock Jenkins** page, paste this password into the **Administrator password** field and click **Continue**.
   Note:
   - The Jenkins console log indicates the location (in the Jenkins home directory) where this password can also be obtained. This password must be entered in the setup wizard on new Jenkins installations before you can access Jenkins's main UI. This password also serves as the default administrator account's password (with username "admin") if you happen to skip the subsequent user-creation step in the setup wizard.

Customizing Jenkins with plugins

After unlocking Jenkins, the **Customize Jenkins** page appears. Here you can install any number of useful plugins as part of your initial setup.

Click one of the two options shown:

- **Install suggested plugins -** to install the recommended set of plugins, which are based on most common use cases.
- **Select plugins to install -** to choose which set of plugins to initially install. When you first access the plugin selection page, the suggested plugins are selected by default.

If you are not sure what plugins you need, choose **Install suggested plugins**. You can install (or remove) additional Jenkins plugins at a later point in time via the **Manage Jenkins** > **Plugins** page in Jenkins.

The setup wizard shows the progression of Jenkins being configured and your chosen set of Jenkins plugins being installed. This process may take a few minutes.

Creating the first administrator user

Finally, after customizing Jenkins with plugins, Jenkins asks you to create your first administrator user.

1. When the **Create First Admin User** page appears, specify the details for your administrator user in the respective fields and click **Save and Finish**.
2. When the **Jenkins is ready** page appears, click **Start using Jenkins**.
   Notes:

- This page may indicate **Jenkins is almost ready!** instead and if so, click **Restart**.
- If the page does not automatically refresh after a minute, use your web browser to refresh the page manually.

3. If required, log in to Jenkins with the credentials of the user you just created and you are ready to start using Jenkins!

Conclusion

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

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

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

⇐ Docker
⇑ Installing Jenkins Index
Linux ⇒

Project

Structure and governance
Issue tracker
Roadmap
GitHub
Jenkins on Jenkins

Community

Forum
Events
Mailing lists
Chats
Special Interest Groups
𝕏 (formerly Twitter)
Reddit

Other

Code of Conduct
Press information
Merchandise
Artwork
Awards