

CONFIGURING A JENKINS PIPELINE (USING POD AGENT TEMPLATES) TO WORK WITH A PRIVATE GITLAB REPOSITORY, HARBOR IMAGE REGISTRY AND KUBERNETES CLUSTER

By Thirumaleshwara Reddy Kamireddy, Shashank Sharma



Thirumaleshwara Reddy Kamireddy

Engineer, Cloud and Paas Development, Tata Communications Limited

Thiru is a Cloud Platform Engineer at Tata Communications, working with Kubernetes, with technical expertise in CI/CD space - Jenkins, Gitlab and Harbor, not just the usage but also providing these services to end customers. Currently he is exploring Kafka technology and the ecosystem of benchmarking services like Kafka.



Shashank Sharma

Principal, Cloud and Paas Development, Tata Communications Limited

Shashank Sharma has been following cloud native OSS for more than seven years, with a particular focus on security, observability and serverless, and other platform engineering trends. At Tata Communications, he is working as an Engineering Manager for Kubernetes and PaaS services.



Introduction

CI/CD pipelines are essential for delivering high-quality software efficiently in today's development landscape. This guide walks you through setting up a Jenkins pipeline using Kubernetes pod agents, integrating with private GitLab repositories and Harbor image registry for automated containerised deployments.

Why use pod agent templates?

When Jenkins integrates with Kubernetes, Pod Agent Templates make CI/CD pipelines more scalable and efficient. Jenkins creates temporary pods for each job, ensuring clean and isolated environments. These pods automatically scale with the demands of the number of pipelines and are deleted after job completion, saving resources while providing consistent, reusable templates across different pipelines.

Implementation

Tools needed

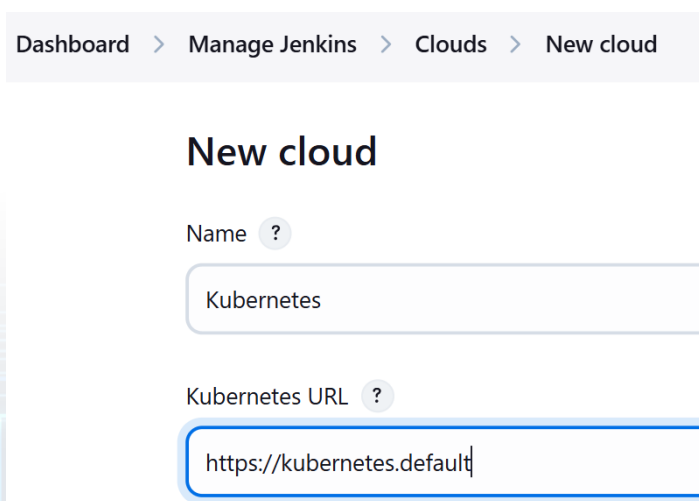
• Kubernetes Cluster | • Docker | • Jenkins | • Harbor | • GitLab

Setting up Jenkins Kubernetes integration

Step 1

Configure Kubernetes Cloud

- Open Jenkins and go to Manage Jenkins → Clouds.
- Click “New Cloud” to create a new cloud configuration.
- Select Kubernetes as the cloud type.
- Add the Kubernetes URL in the Jenkins Kubernetes Cloud configuration, as illustrated in Figure 1. For example: <https://kubernetes.default>.
- If the Jenkins controller is running inside the Kubernetes cluster, specify the internal service DNS as the Jenkins URL, as shown in Figure 2.



Dashboard > Manage Jenkins > Clouds > New cloud

New cloud

Name ?

Kubernetes

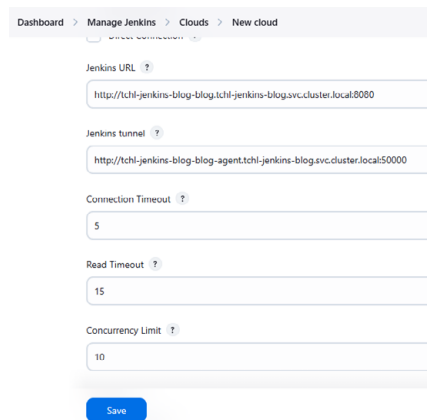
Kubernetes URL ?

<https://kubernetes.default>

Figure 1

Configure Kubernetes URL and connection settings

- Specify the Jenkins tunnel endpoint during configuration, as illustrated in Figure 2. The Jenkins tunnel facilitates inbound communication from agent pods to the Jenkins controller. This is essential in Kubernetes setups, where agents rely on this tunnel to establish a stable connection for executing build tasks.



Dashboard > Manage Jenkins > Clouds > New cloud

Jenkins URL ?

Jenkins tunnel ?

Connection Timeout ?

Read Timeout ?

Concurrency Limit ?

Figure 2

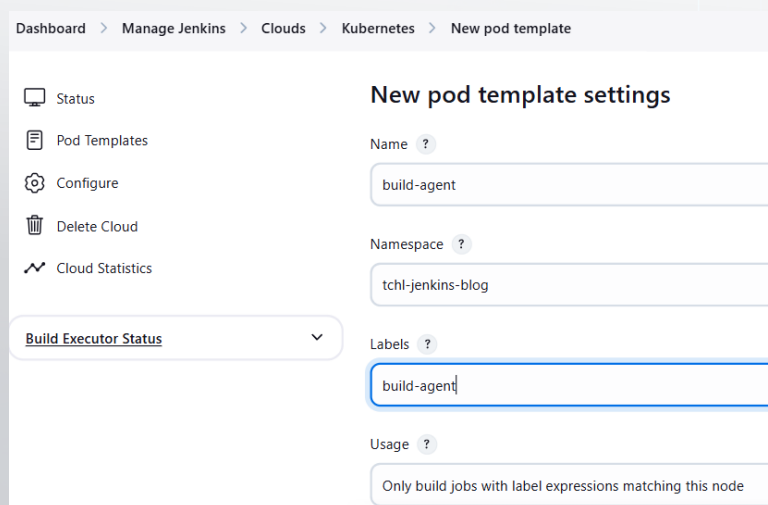
Set Jenkins tunnel endpoint and concurrency limits

- Set the Concurrency Limit to control parallel Jenkins agents. Leave empty for unlimited agents. Click Save after configuration.

Step 2

Create pod template

- Navigate to “Pod Templates” under the Kubernetes Cloud section and click “Add Pod Template”.
- Fill in the name, label and namespace for the agent pod as illustrated in Figure 3. This label connects Jenkins jobs to this specific pod template. When a pipeline uses agent `{label 'build-agent'}`, Jenkins uses that label to find and launch the appropriate environment for the job.
- The name helps identify the pod template within Jenkins and is useful when managing multiple templates.
- The namespace defines where the agent pod will be created in the Kubernetes cluster. Jenkins must have access to this namespace to run the builds successfully.
- Use the build-agent.yaml file from [this](#) GitHub repository for reference.
- Click “Create” to save the pod template.



Dashboard > Manage Jenkins > Clouds > Kubernetes > New pod template

Status
Pod Templates
Configure
Delete Cloud
Cloud Statistics

Build Executor Status

New pod template settings

Name ?

Namespace ?

Labels ?

Usage ?

Figure 3

Configure pod template name, namespace, and labels

```
spec:
  hostAliases:
  - ip: "10.8.30.9"
    hostnames:
    - "gitlab.private-example.com"
  - ip: "10.8.30.9"
    hostnames:
    - "harbor.registry.private-example.com"
```

Figure 4 HostAliases mapping for GitLab and Harbor access

- The hostAliases section in the build agent pod specification, as illustrated in Figure 4, allows the container to resolve private domain names that may not be publicly accessible. In this example:
 - gitlab.private-example.com maps to the internal IP of the private GitLab server
 - harbor.registry.private-example.com maps to the internal IP of the private Harbor image registry.

Similarly, for the build agent pod to interact with a Kubernetes cluster, a container must be added to interact with the API Server and do the deployment, as illustrated in Figure 5.

```
- name: kubectl
  image: joshendriks/alpine-k8s
  resources:
    requests:
      cpu: "150m"
      memory: "100Mi"
    limits:
      cpu: "300m"
      memory: "200Mi"
  command:
  - /bin/cat
  tty: true
  securityContext:
    privileged: true
```

Figure 5 Kubectl Container configuration for cluster interaction

Step 3 Configuring Jenkins Credentials

- Go to Manage Jenkins → Credentials.
- Open “Global credentials”.
- Click “Add Credentials” to store GitLab or Harbor access credentials as illustrated in Figure 6.

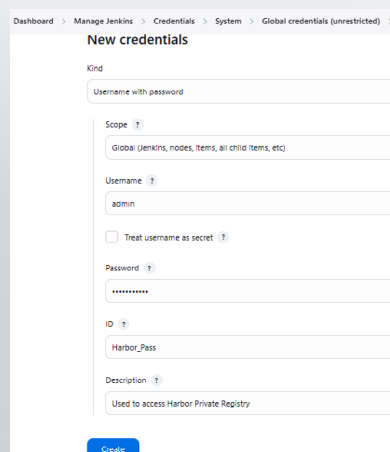


Figure 6 Configure Harbor registry credentials

- Click “Create” to save credentials.
- These credentials can be then referenced in the pipelines.

Creating the Jenkins pipeline

Step 1 Create new pipeline project

- In the Jenkins dashboard, click on New Item.
- Enter a name for the pipeline (e.g., harbor-kubernetes-pipeline).
- Select pipeline as the project type and click OK.

Step 2 Configure pipeline script

- Go to the Pipeline section in the Jenkins job configuration.
- Select Pipeline Script as the definition type.
- Paste the Jenkinsfile content into the script area as shown in Figure 7.
- Refer to the [Jenkinsfile](#) from the GitHub repository - it automates the workflow by checking out the code from Gitlab, building a Docker image, pushing it to Harbor, and deploying a pod in the Kubernetes cluster.
- Jenkins executes each stage in sequence and replaces variables like \${BUILD_NUMBER} and \${BUILD_TAG} with actual build values. These dynamic tags uniquely identify Docker images, making them easier to track in Harbor and the Kubernetes Cluster.

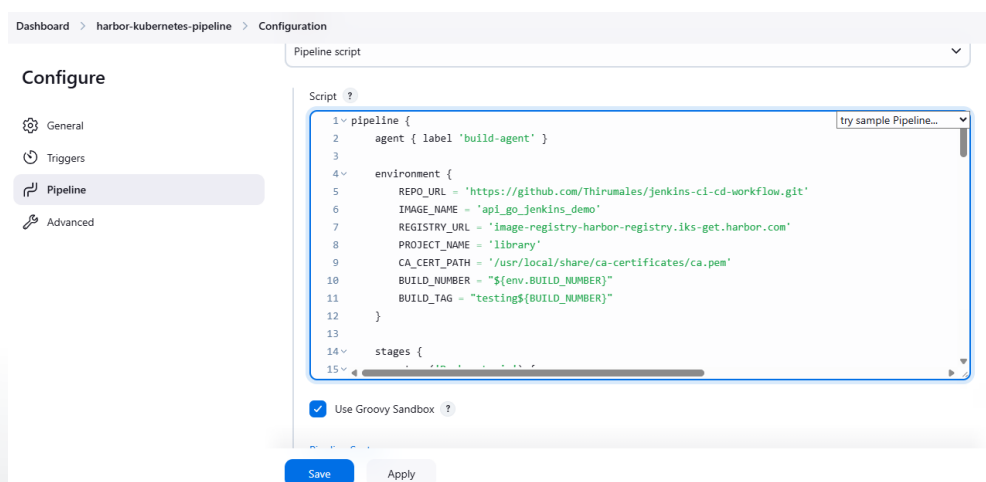


Figure 7 Configure pipeline script with Jenkinsfile content

Step 3 Running the pipeline

- Click "Build Now" to start the pipeline execution.
- Monitor the progress using "Console Output" to view detailed logs.



Monitor pipeline activities

- Ensure that the Jenkins instance you are using has enabled Prometheus and Grafana integration.
- Open Grafana to track the build pod's metrics like CPU and Memory while the pipeline is running.



Figure 8 CPU Usage of Build Agent containers during pipeline execution



Figure 9 Real-time Jenkins Metrics - jobs, memory, CPU and health

How the pipeline works

The CI/CD pipeline follows the workflow illustrated in Figure 10:

- **Code Development:** Developers create a Dockerfile defining application packaging requirements.
- **GitLab Integration:** Jenkins retrieves the latest source code from a private GitLab repository during the checkout stage.
- **Testing Phase:** Pipeline executes unit tests using Golang containers, ensuring code quality before proceeding.

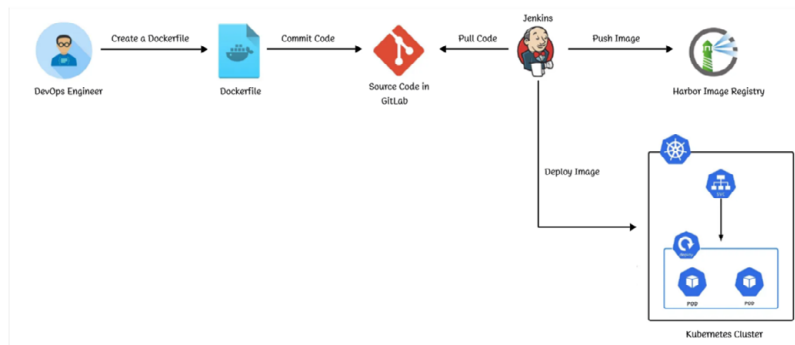


Figure 10 Complete CI/CD workflow from development to deployment

- **Image Building:** Jenkins builds container images using checked-out source code, tagging them with unique build numbers for version tracking.
- **Harbor Authentication:** Jenkins securely logs into Harbor registry using stored credentials, ensuring private image repository access.
- **Image Publishing:** Built images are pushed to Harbor registry with a proper CA certificate mounted for secure communication.
- **Kubernetes Deployment:** The pipeline updates the Kubernetes deployment manifest by replacing the image tag placeholder with the newly built version. Then, using the kubectl container defined in the pod template, Jenkins applies the updated manifest to deploy the application to the Kubernetes cluster.

Conclusion

This Jenkins pipeline configuration with Kubernetes pod agents streamlines the entire CI/CD process, from source code management to automated deployment.

Using pod agents in Jenkins ensures clean and consistent build environments while efficiently using resources. With secure credential handling and a streamlined pipeline setup, this approach enables reliable and scalable CI/CD workflows. By following this configuration, teams can accelerate deployments and achieve smoother integration from development to production.

How can Tata Communications help you get started on this?

We understand that setting up the pipelines and the associated integrations can be overwhelming (at least the first few times). It is something organisations across the digital front have adopted for a few years. Still, when we hear customer journeys and their problem statements, the discussions always segue into one single conclusion and that is - even though setting up pipelines is something our customers would like to do because of the deep association with the actual application, what they would like to delegate to us as a service provider is - **helping them with things like Jenkins, Gitlab and Container Registry as a service!** And we have done precisely that.

We can help you quickly start your pipelines and complete CICD setup in Vayu Cloud using our [Managed Kubernetes Service](#) and the dedicated instances of Jenkins, Gitlab and a Container Registry. This means we manage these services' patches, security, and observability aspects. At the same time, you quickly get started with your application deployment. To know more, get in touch with us [here!](#)