How to Use Kubernetes Pods as Jenkins Agents

- The Jenkins architecture is designed for distributed build environments.
- It allows us to use different environments for each build project balancing the workload among multiple agents running jobs in parallel.
- The Jenkins controller is the original node in the Jenkins installation.
- The Jenkins controller administers the Jenkins agents and orchestrates their work, including scheduling jobs on agents and monitoring agents.
- Agents may be connected to the Jenkins controller using either local or cloud computers.
- The agents require a Java installation and a network connection to the Jenkins controller.

Configuring agents with Docker

- Jenkins agents can be launched in physical machines, virtual machines, Kubernetes clusters, and with Docker images.
- The Jenkins agent usually connect to the control Jenkins with SSH4

Using Kubernetes as Jenkins Agents

- Launch your jenkins control through terraform or terragrunt.
- Install Kubernetes plugin for Jenkins
- Jenkins plugin to run **dynamic agents** in a Kubernetes cluster.
- JNLP - This is an image for Jenkins agents using TCP or WebSockets to establish inbound connection to the Jenkins master. This agent is powered by the Jenkins Remoting library, which version is being taken from the base Docker Agent image.

Connecting Inbound Agents

https://docs.cloudbees.com/docs/cloudbees-ci/latest/cloud-setup-guide/configure-ports-jnlp-agents

We are going to use: WebSocket for Agent Remoting Connections
Configure the Kubernetes plugin

Get the jenkins url
Get the service account authentication token
Get the cluster certificate

Commands for kind cluster

- kind create cluster --config jenkins-config.yaml
- kubectl cluster-info --context kind-kind
- kubectl create namespace jenkins
- kubectl create serviceaccount jenkins --namespace=jenkins
- kubectl describe secret $(kubectl describe serviceaccount jenkins --namespace=jenkins | grep Token | awk '{print $2}') --namespace=jenkins
- kubectl create rolebinding jenkins-admin-binding --clusterrole=admin --serviceaccount=jenkins:jenkins --namespace=jenkins

Create a new pipeline, paste this information, save and build

Notice the jenkins inbound image jnlp is already available

Pipeline version 1

```
pipeline {
  agent {
    kubernetes {
      yaml '''
        apiVersion: v1
        kind: Pod
        spec:
          containers:
          - name: maven
            image: maven:alpine
            command:
            - cat
            tty: true
      '''
    }
  }
  stages {
    stage('Run maven') {
      steps {
        container('maven') {
          sh 'mvn -version'
        }
```

```
    }
   }
  }
}
```

Pay Attention to this section

- cat
  Tty: true

It would be nice to point out the importance of "tty: true" in the yaml. Jenkins needs that to be true to allow commands to be sent to the container. Otherwise, the job just loops trying to run without any error.

You are used to see

Pipeline {

ageny any

but now, in this case, the agent is Kubernetes

- In the above example, we just defining the pod we want created.
- Within our job stages, it is vital to see how we are calling the container name maven.
- If the image does not exist, it will be pulled from dockerhub.com
- Another container that is pulled is the jenkins/inbound-agent. This is the container within the pod that is making the SSH connection back to our jenkins controller possible.
- Jenkins/inbound-agent is injected by the Kubernetes plugin.
- Within the pods, you can have multiple containers.

Pipeline version 2

```
pipeline {
  agent {
    kubernetes {
      yaml '''
        apiVersion: v1
        kind: Pod
        spec:
          containers:
```

```
          - name: maven
            image: maven:alpine
            command:
            - cat
            tty: true
          - name: node
            image: node:16-alpine3.12
            command:
            - cat
            tty: true
        '''
    }
  }
  stages {
    stage('Run maven') {
      steps {
        container('maven') {
          sh 'mvn -version'
        }
        container('node') {
          sh 'npm version'
        }
      }
    }
  }
}
```

Within the pod created, multiple containers can access the same workspace

Pipeline version 3

```
pipeline {
  agent {
    kubernetes {
      yaml '''
        apiVersion: v1
        kind: Pod
        spec:
          containers:
          - name: maven
            image: maven:alpine
            command:
            - cat
            tty: true
          - name: node
            image: node:16-alpine3.12
            command:
            - cat
```

```
          tty: true
        '''
      }
    }
    stages {
      stage('Run maven') {
        steps {
          container('maven') {
            sh 'mvn -version'
            sh ' echo Hello World > hello.txt'
            sh 'ls -last'
          }
          container('node') {
            sh 'npm version'
            sh 'cat hello.txt'
            sh 'ls -last'
          }
        }
      }
    }
}
```

1. We are creating a workspace called
   Echo Hello > hello.txt (add to the last line in the file) vs echo Hello >> hello.txt (delete and add the new content)
   When the new container is created, it is listed in
   ls -last

   The workspace is shared within the pods – this is important because if you want to pass a small file to another container like just a json or yaml fileto another container which is faster instead using other methods.

2. It is important to get rid of static agents. However, you should also remember that it is impossible to get rid of all the static agents.
   For instance, you cannot containerize McOs operation system and many more

3. Running dynamic agents make the security happy. It means you do not have vm instances to maintain. Make infrastructure happy because you are not wasting their resources by using vm instances as agents that are just seated somewhere and being used time to time.
4. It also creates a more dynamic pipeline that you as a DevOps engineer do not need to bother maintaining. This is because the engineers can easily define the versions they would like to work with.