# Creating Dynamic Agents – Using Kubernetes Pods As Jenkins Agents

## Highly Recommended

- It is essential to get rid of static agents.
- However, it would help if you also remembered that it is impossible to eliminate all the static agents.
- For instance, you cannot containerize the MacOs operation system and many more.
- Running dynamic agents makes the security teams happy, and they highly recommend it.
- It means you do not have VM instances to maintain.
- This means you are not making static infrastructure costing the company money leading to wasteful resources because even when you are not using static Jenkins agents, they are still running.
- Using dynamic agents provisioned as pods by the cluster help one to create a ROBUST, SCALABLE, EFFECTIVE, EFFICIENT & SECURE CI/CD pipeline.
- The pipeline is dynamic because the engineers do not need to bother maintaining the applications' versions.
- This is because the engineers can easily define the versions they would like to work with and select the most updated ones.

In simple words, Kubernetes is going to fry the pod/s when our job/s are ready to build.

We do not need to worry about creating an agent.

An agent called Jenkins/inbound agent will be available immediately we install KUBERNETES plugin.

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.

Also, you need to be aware of Java Network Launching Protocol (JNLP).

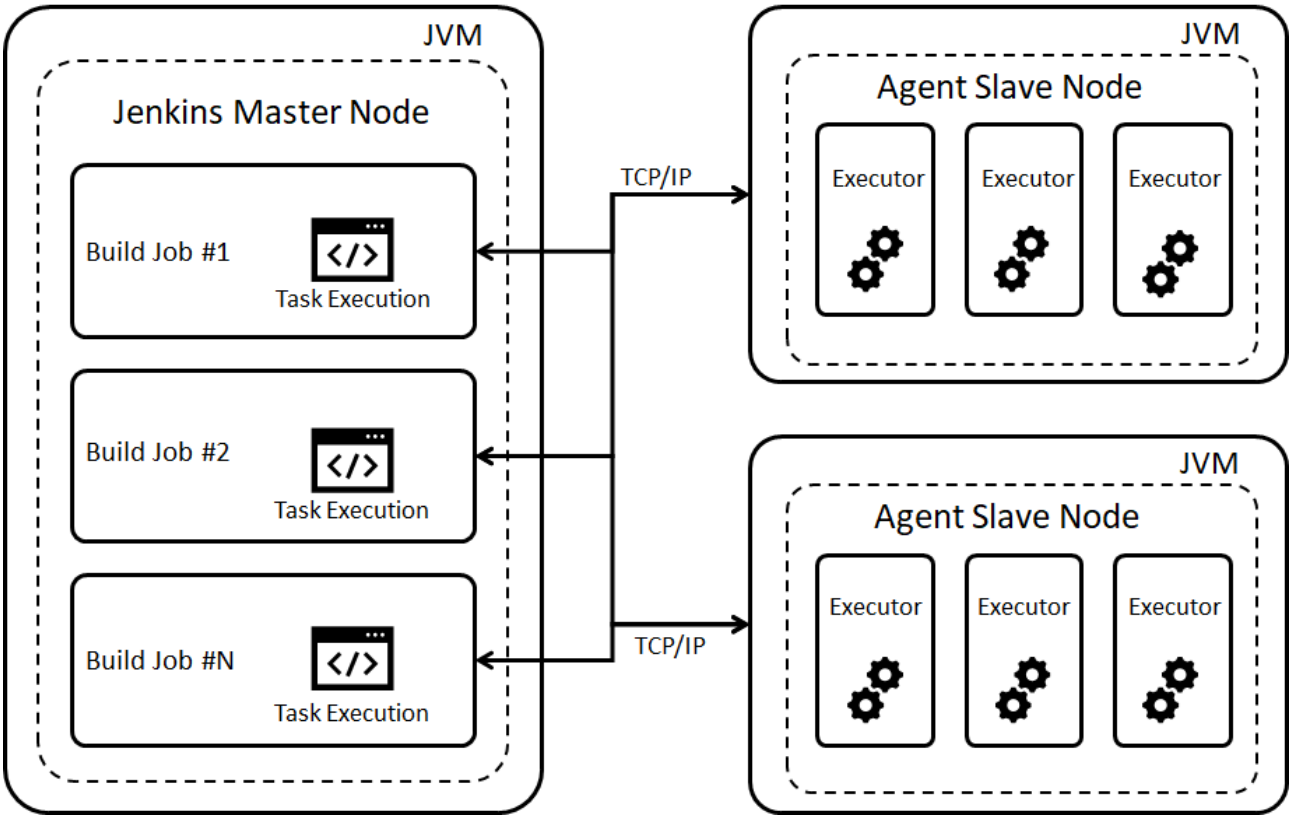
The Java Network Launch Protocol (JNLP) enables an application to be launched on a client desktop by using resources that are hosted on a remote web server.

Java Plug-in software and Java Web Start software are considered JNLP clients because they can launch remotely hosted applets and applications on a client desktop.

We can use this setup to understand the relationship between the Jenkins Master and the Slaves. Not good words to us.

We should call the Jenkins master (the control plane) and the Jenkins slaves the (Jenkins agents).

The Jenkins Control, Agent Structure




Step 1

Provision your Jenkins control using terraform/terragrunt

My jenkins control is ready

Welcome to Jenkins!

Username

Password

Sign in

Keep me signed in

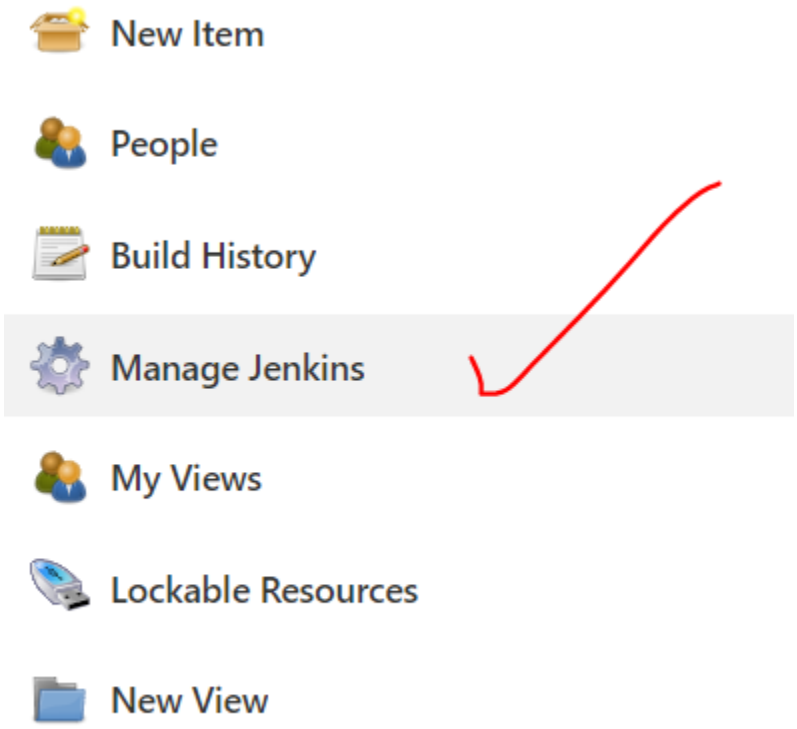
Our Jenkins Commandment

- ⇨ As DevOps engineers, we shall never build jobs in control Jenkins.
- ⇨ The work of the control Jenkins is to manage the plugins, schedule the jobs to the respective Static and Dynamic agents, manage the applications and the pipeline scripts related to the jobs, and make sure the build interfaces and logs are available to us.
- ⇨ Building with control Jenkins affects the functionality and performance of the server because it becomes overburdened with too many tasks running simultaneously.
- ⇨ It is also not secure to build with control Jenkins.

The first step before configuring anything else is to disable the control Jenkins build executors to ensure the Jenkins control will never schedule or build a job.

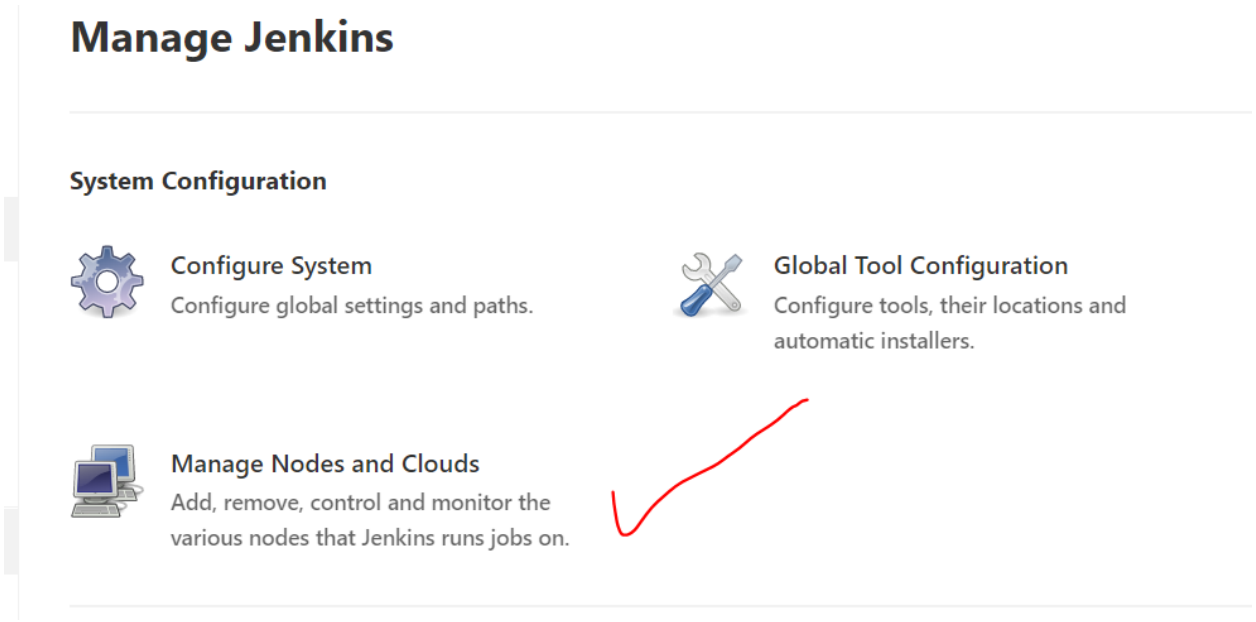
Steps

Click on

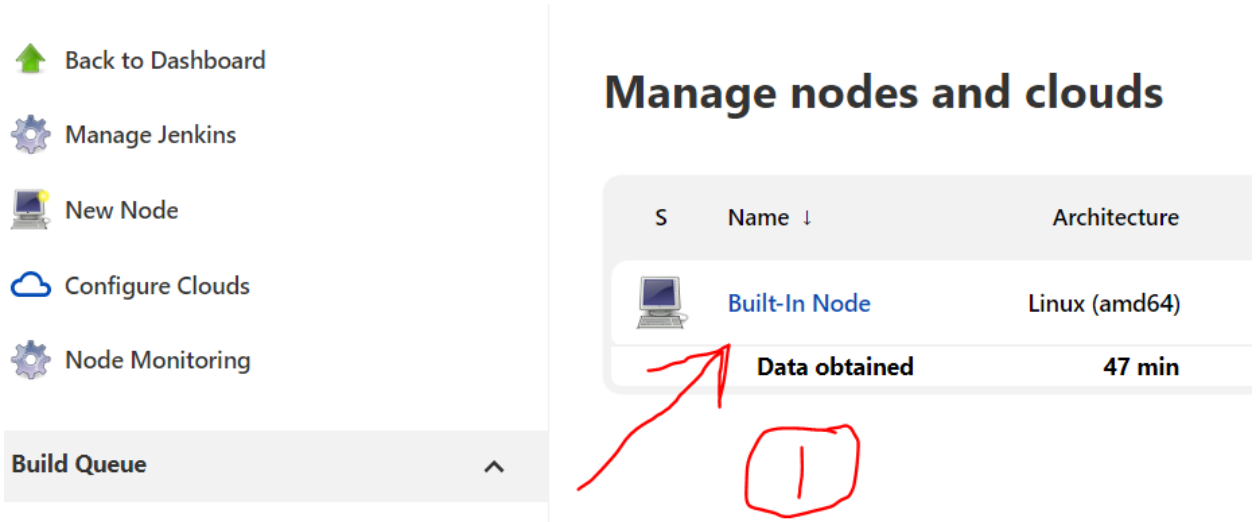



Next

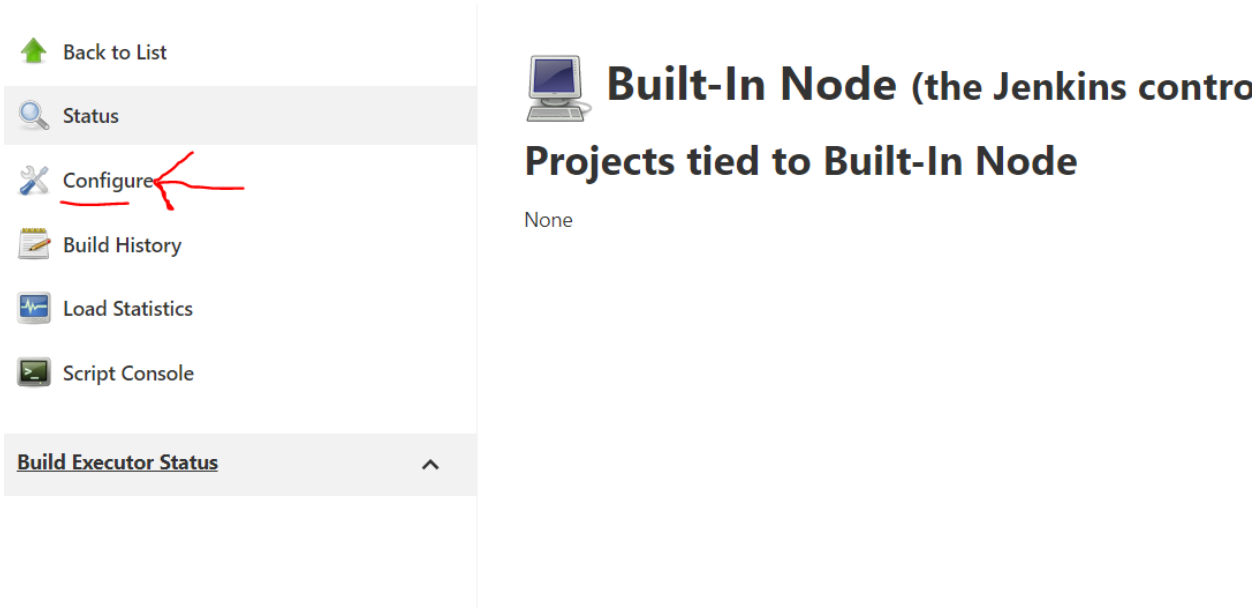
Click on


Manage Jenkins
System Configuration
Configure System
Configure global settings and paths.
Global Tool Configuration
Configure tools, their locations and automatic installers.
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Next

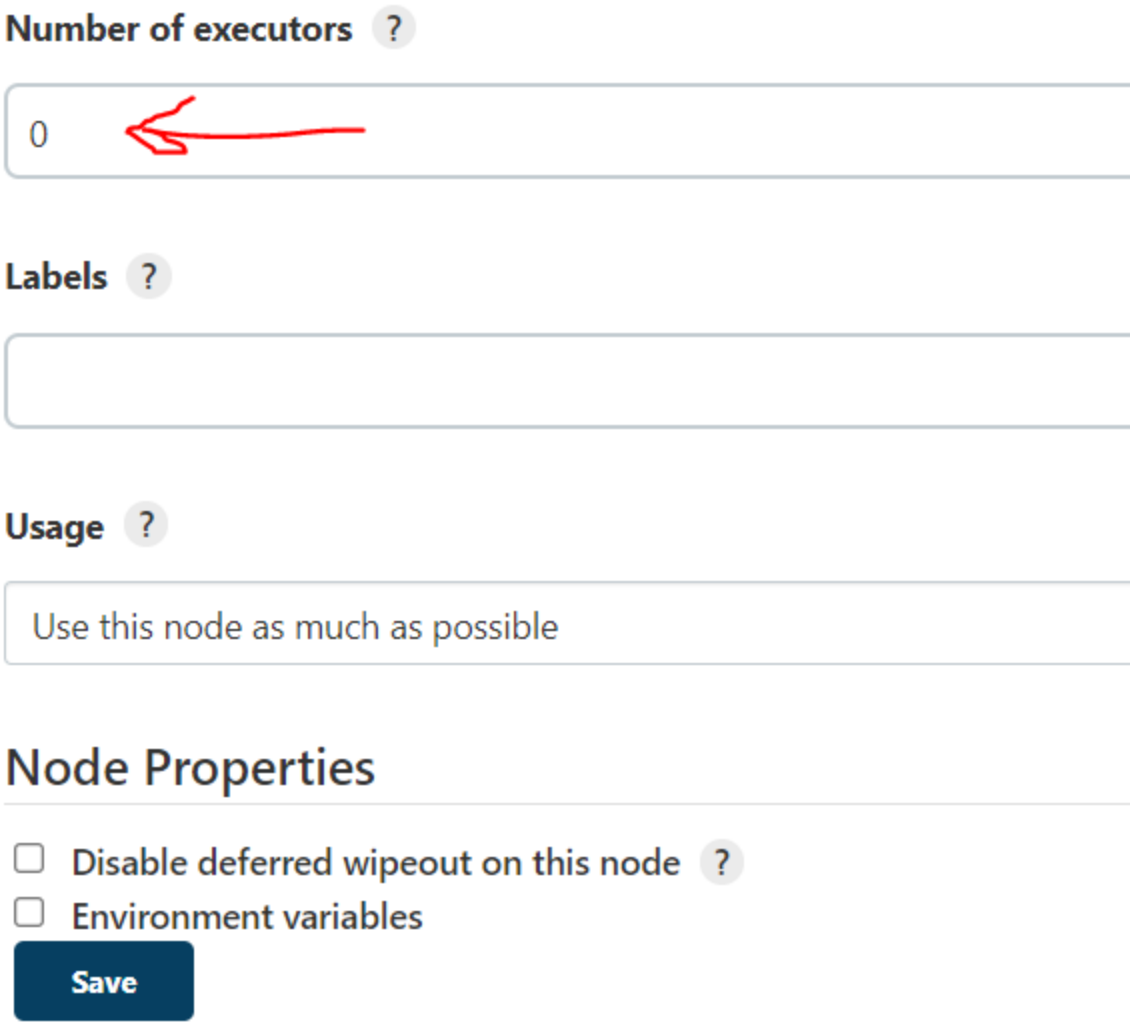

Back to Dashboard
Manage Jenkins
New Node
Configure Clouds
Node Monitoring
Build Queue
No builds in the queue.
Build Executor Status
Manage nodes and clouds
S
Name ↓
Architecture
Built-In Node
Linux (amd64)
Data obtained
47 min
1

Next

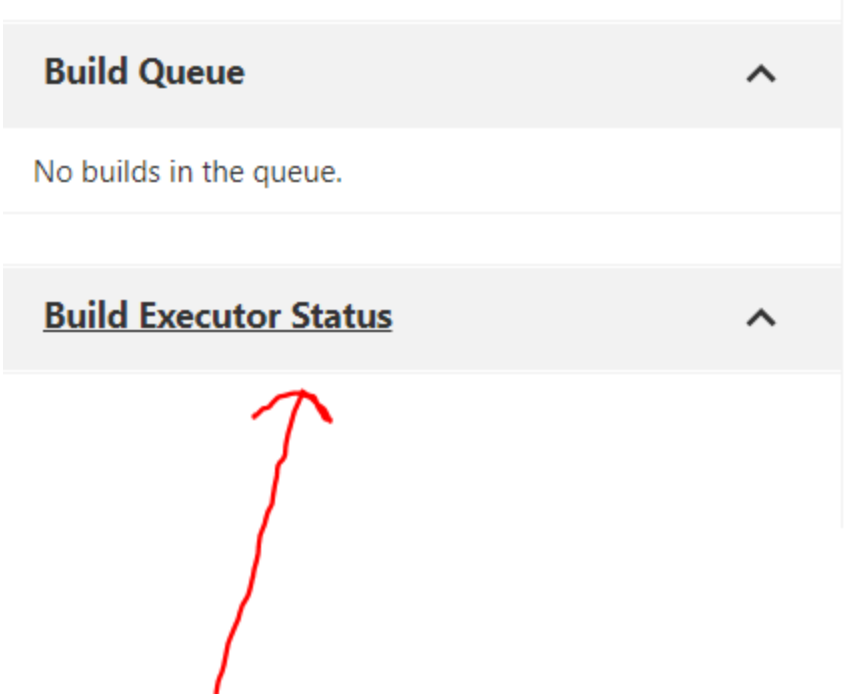



Next

Change the number of executors to zero and save




Confirm no executor exists

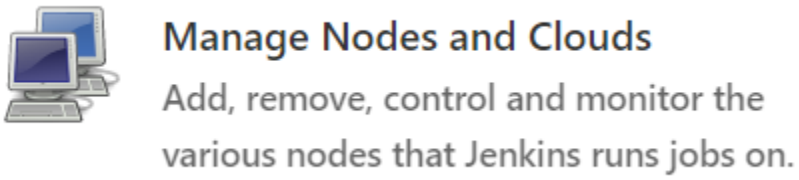



No executor exists!!!! ➔ you can now move to the next steps below

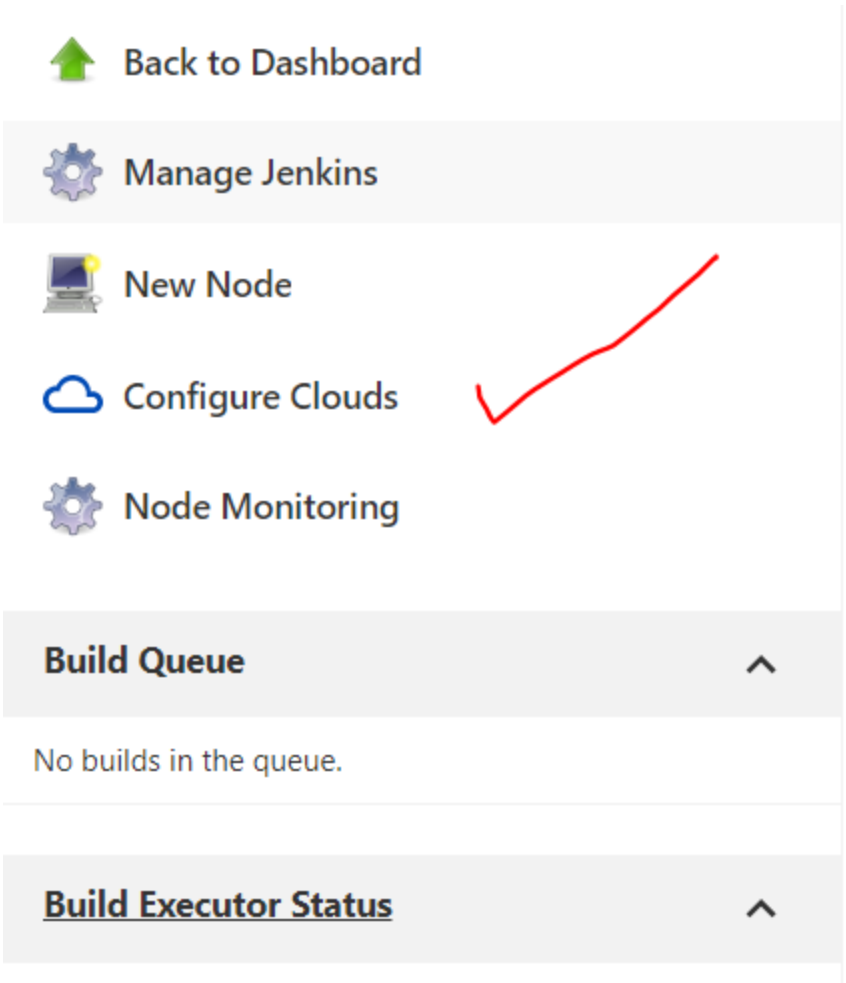
2. Install the Kubernetes plugin - this plugin integrates Jenkins with Kubernetes

Manage Jenkins ➔ manage plugins➔ available ➔ install without restart

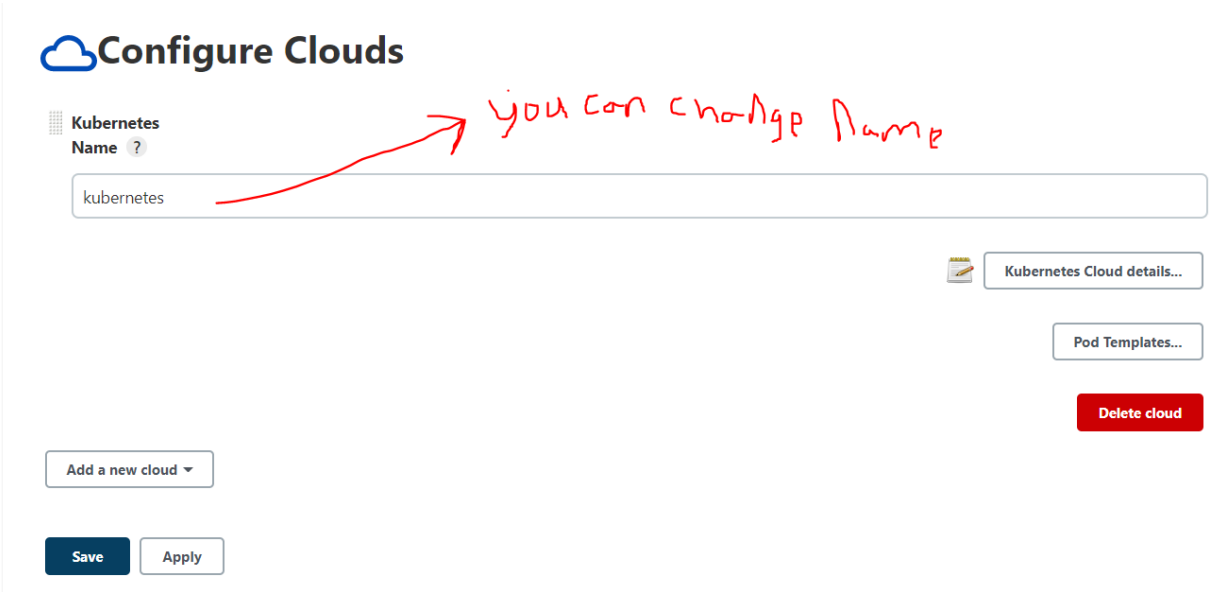
3. Go to Manage Jenkins ➔ manage nodes and clouds




Click on manage nodes and clouds



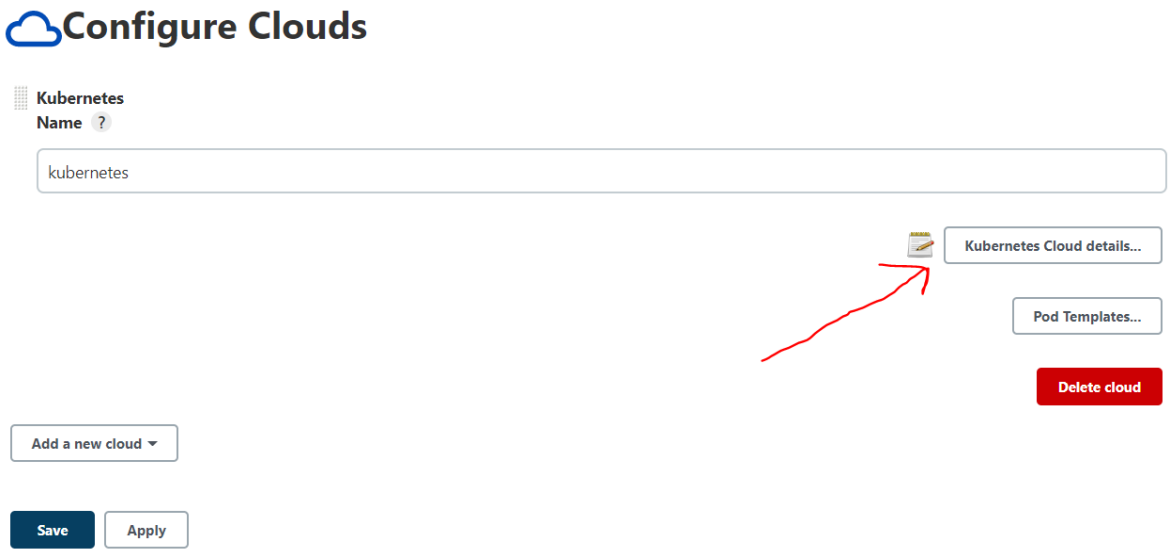

⇨ Click on Configure Cloud



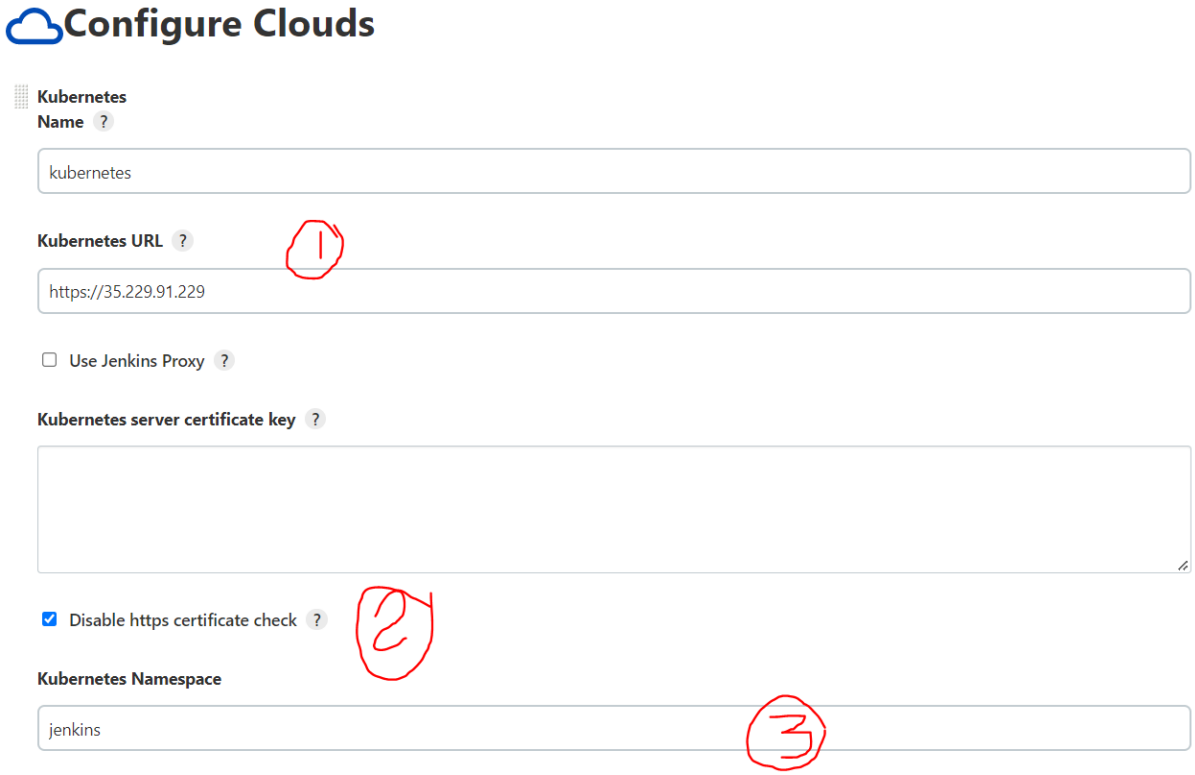
You can change the name from Kubernetes to whatever you like.

The rule of the thumb is to leave the name as Kubernetes.

⇨ Next



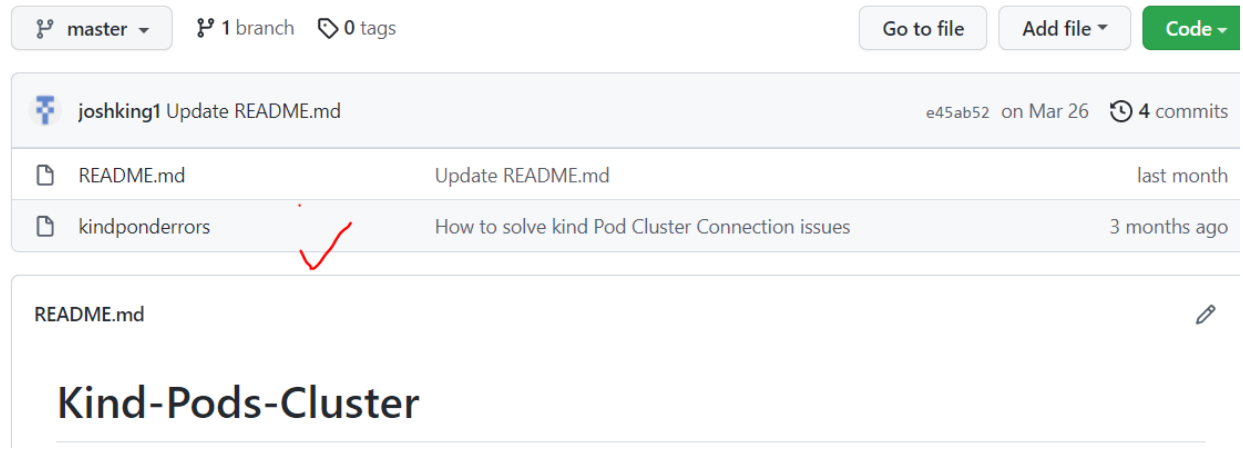

⇨ Next



Step 1: Get the url for the cluster

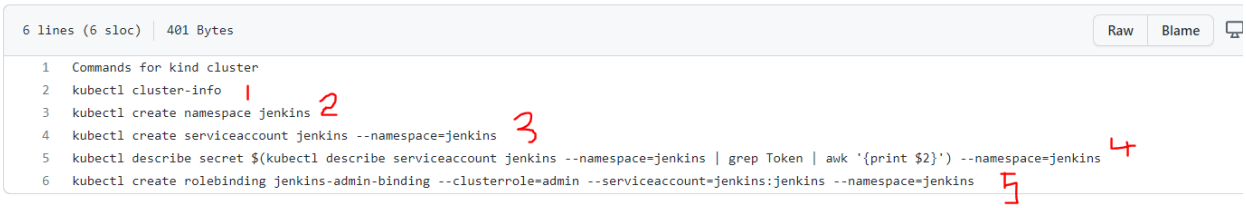
Go to ➔ https://github.com/joshking1/Kind-Pods-Cluster.git and fork it

You should have the following contents




Click on Kindpoderrors – Spelling (never mind). I did fix it.

You should have these commands



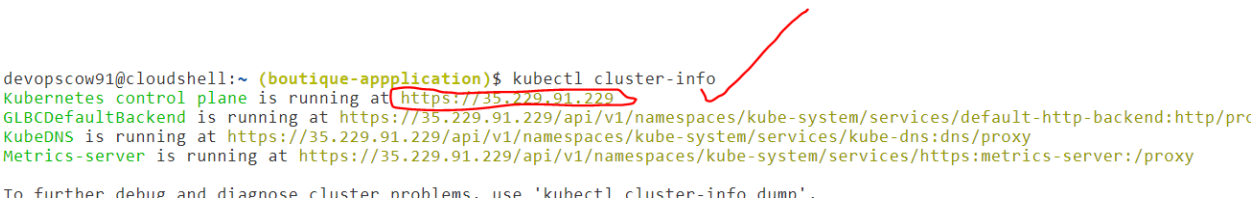

To get the Kubernetes URL, go to the Kubernetes cluster node terminal

For this project we are using the Google Kubernetes Cluster

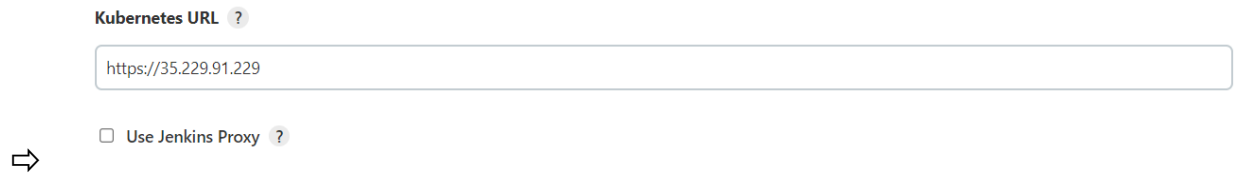
Run the command number 1 in your cluster

Command: #  kubectl cluster-info

Output

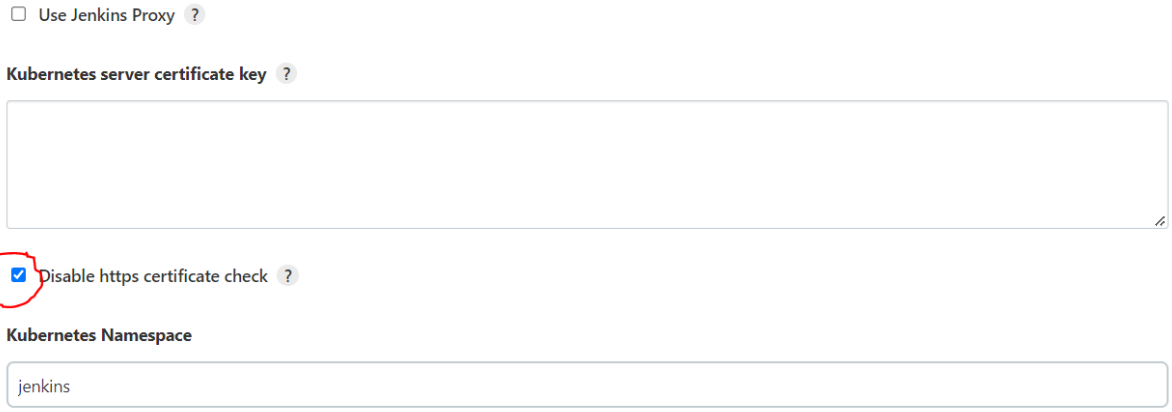


Take the URL and paste it here




⇨

2 Kubernetes certificate




Disable certificate check – this should never happen in production

We can feed the certificate and uncheck the disabled icon, but for now, do not worry about this ➔ I will guide you on this when we reach this stage.

3. Create a namespace called jenkins


Command

# kubectl create namespace jenkins

Output

```
devopscow91@cloudshell:~ (boutique-appplication)$ kubectl create namespace jenkins
namespace/jenkins created
devopscow91@cloudshell:~ (boutique-appplication)$
```

Check whether the namespace exists

```
devopscow91@cloudshell:~ (boutique-appplication)$ kubectl get namespace jenkins
NAME      STATUS   AGE
jenkins   Active   95s
devopscow91@cloudshell:~ (boutique-appplication)$
```

# 4. Create a Service Account

A service account is important in making sure the jenkins control is not anonymous to the cluster.

A service account exists in a namespace within a cluster.

When a service account is created, a token for authentication by the Kubernetes API servers is generated.

Let create a service account using command

Command

# kubectl create serviceaccount jenkins --namespace=jenkins

Output

```
devopscow91@cloudshell:~ (boutique-appplication)$ kubectl create serviceaccount jenkins --namespace=jenkins
serviceaccount/jenkins created
devopscow91@cloudshell:~ (boutique-appplication)$
```

Let us get the token generated and which is connected to this service account

Command: # kubectl describe secret $(kubectl describe serviceaccount jenkins --namespace=jenkins | grep Token | awk '{print $2}') --namespace=jenkins

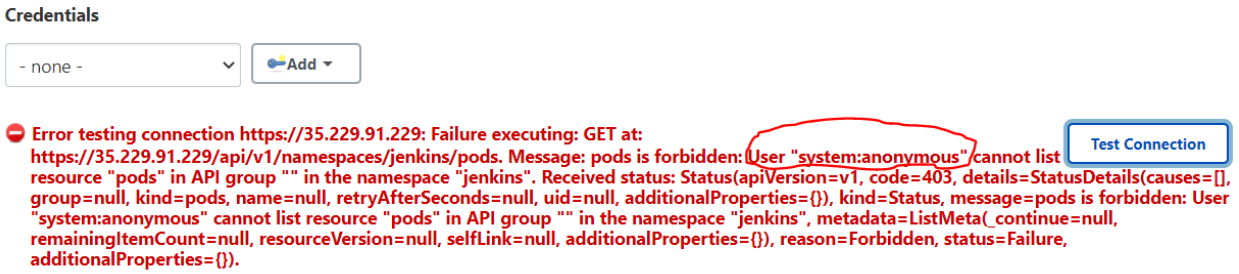
Output

```
grep Token | awk '{print $2}') --namespace=jenkins
Name:         jenkins-token-n7fz5
Namespace:    jenkins
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: jenkins
              kubernetes.io/service-account.uid: 4b839cac-b6a8-4d87-9fcd-9c4cbaf7b2ef

Type:  kubernetes.io/service-account-token

Data
====
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IjhDVTl6ZnZkSUVGOEJVMlVyeWt0OWdZb0RkbFNYSUVVYmVZRG84a2s3Y3MifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2\
hY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJqZW5raW5zIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6I
mplbmtpbnMtdG9rZW4tbjdmejUiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiamVua2lucyIsImt1YmVybmV0ZXMuaW8vc2VydmI
jZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjRiODM5Y2FjLWI2YTgtNGQ4Ny05ZmNkLTljNGNiYWY3YjJlZiIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpqZW5rа
W5zOmplbmtpbnMifQ.OkqwRrob6Y84aaCcnZ4zpH3i4zIjR6FVf4H3htDB7vzoLP1Pc3WhJ0hbcAqGesNYVlAibQ2c4AqxSMtxixOs0H7zdOTaRXdy82e2swH0xUFYhwYatlM5pdXy
QZxiAUnn_CYlzU6syFm_zFALrL7uIF0awTGAZCspnpPa7MNlY1Yd2Wz6wVbDYfRCgH15fll92IrKb54KuD5GUD_fh7slNVlhlZtDk1Xgz-9qQoAF-zWEcs84m5u1eIzyCYk1bEKjLk
3YN_UOKWmCodAXt81NKqSJlgGvgZcN_rvLaXHWNfuHPL-QHDUM8GXjA-rKkkChoUyrl1jCxhHtqelIPuMWoA
ca.crt:     1509 bytes
namespace:  7 bytes
```

Let go to our Jenkins and test the connection




Sure, the cluster does not recognize the account. System anonymous.

WHY?

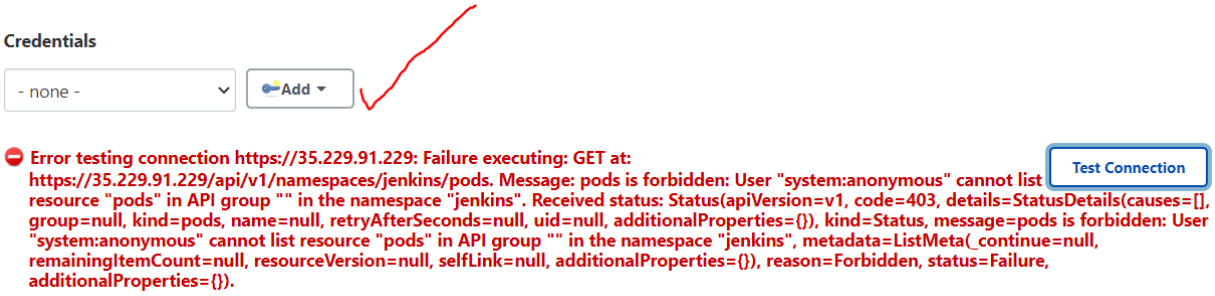
A positive thing – namespace Jenkins is recognized

A negative thing – service account we created is not recognized by the Kubernetes API when we try to connect.

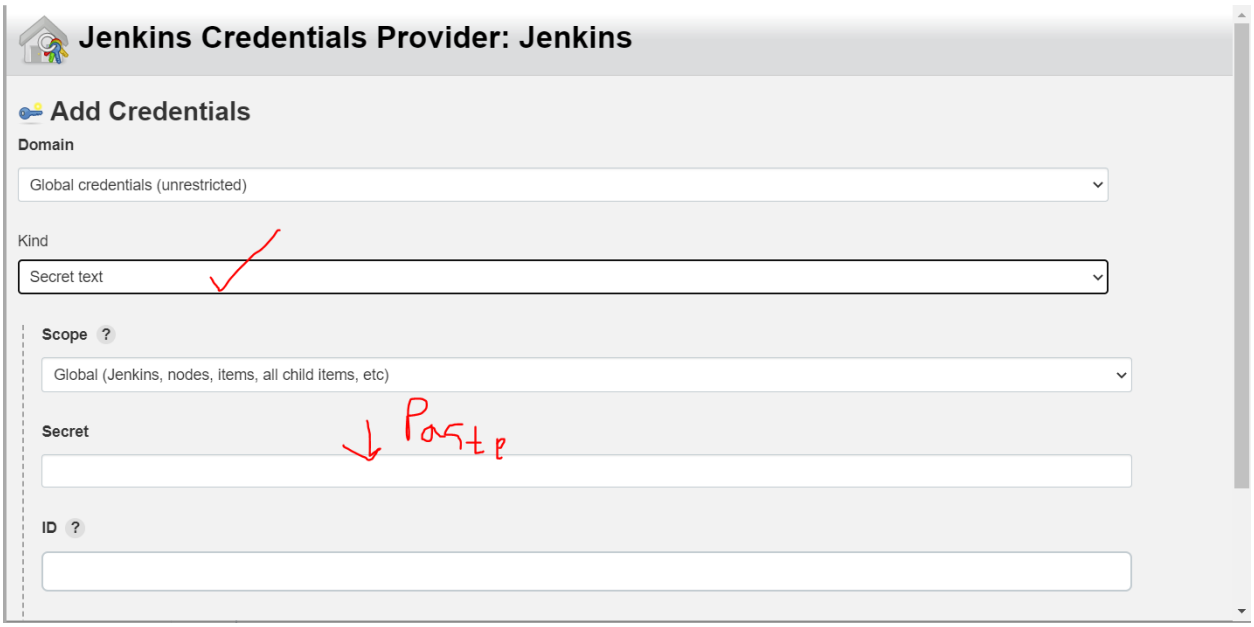
Solutions

1. Create a secret text and save the token generated by the service account.

   Click on ADD




Nex ➔



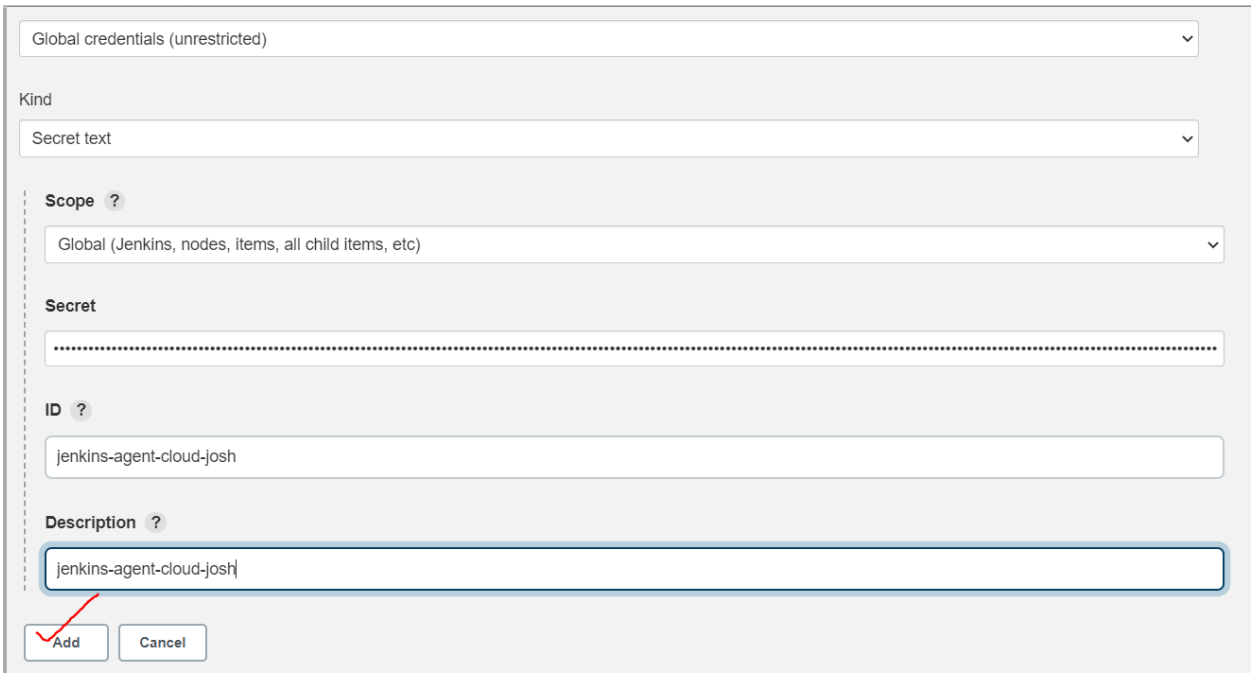

Next ➔

=>Paste the secret generated

=>Create a special ID

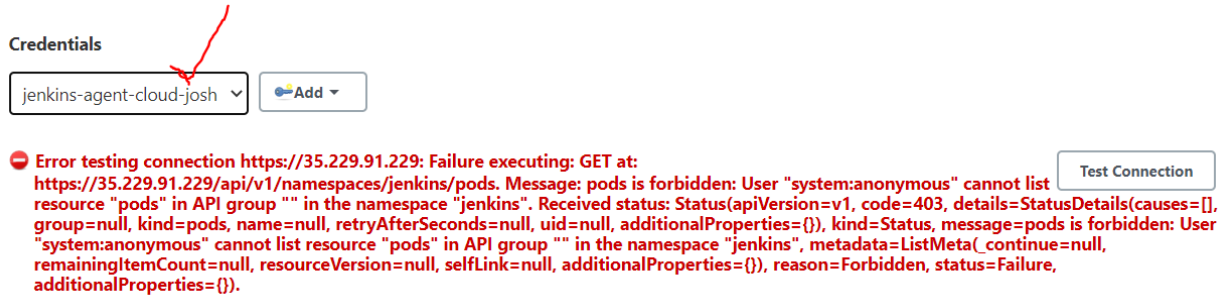
=>Add description

=>Add

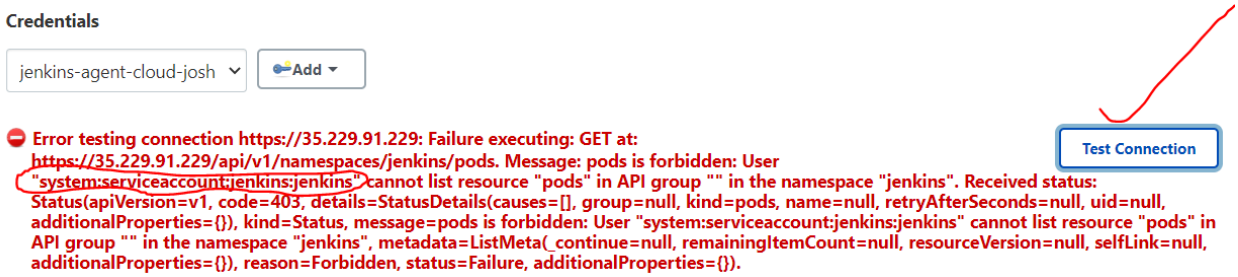



Next ➔

⇨ Select the name of the token we just created.




Next ➔ Run the test connection




Great, we are no longer anonymous. The Kubernetes API have recognized the services account.

Still have a problem ➔ the Service account is forbidden

WHY?

This is because we do not have ROLE (permission created) and ROLEBINDING (attach the permission to the created service account).

Next ➔ Create Role and Role binding

#5 Create rbac (Role based access control)

There are four objects that are used by the Kubernetes API (application programming Interface)

Role – permission

Rolebinding – attaches the permission to the service account created

Both role and rolebinding are namespace oriented and are used to give machines and processes access to resources existing a namespace in the cluster.

Do not forget service account is namespace oriented

Let Create a role and a rolebinding

I am going to teach you how to use YAML file to create the roles and rolebinding instead of using commands when we meet.

Command

kubectl create rolebinding jenkins-admin-binding --role=admin --serviceaccount=jenkins:jenkins --namespace=jenkins

In this role, we are doing the following:

1. Creating a role called admin (access to all the resources in the cluster namespace)
2. Creating a rolebinding (attach the permission created to the services account)

Output

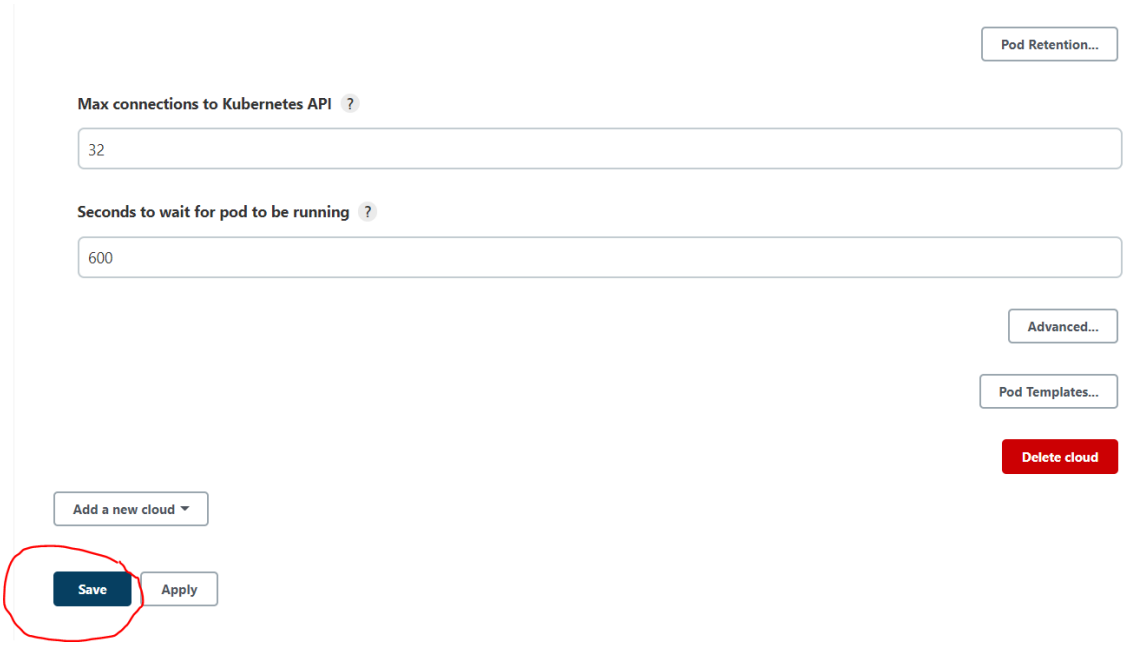
```
devopscow91@cloudshell:~ (boutique-appplication)$ kubectl create rolebinding jenkins-admin-binding --clusterrole=admin --serviceaccount=je
nkins:jenkins --namespace=jenkins
rolebinding.rbac.authorization.k8s.io/jenkins-admin-binding created
devopscow91@cloudshell:~ (boutique-appplication)$
```

Next ➔

Let test the connection

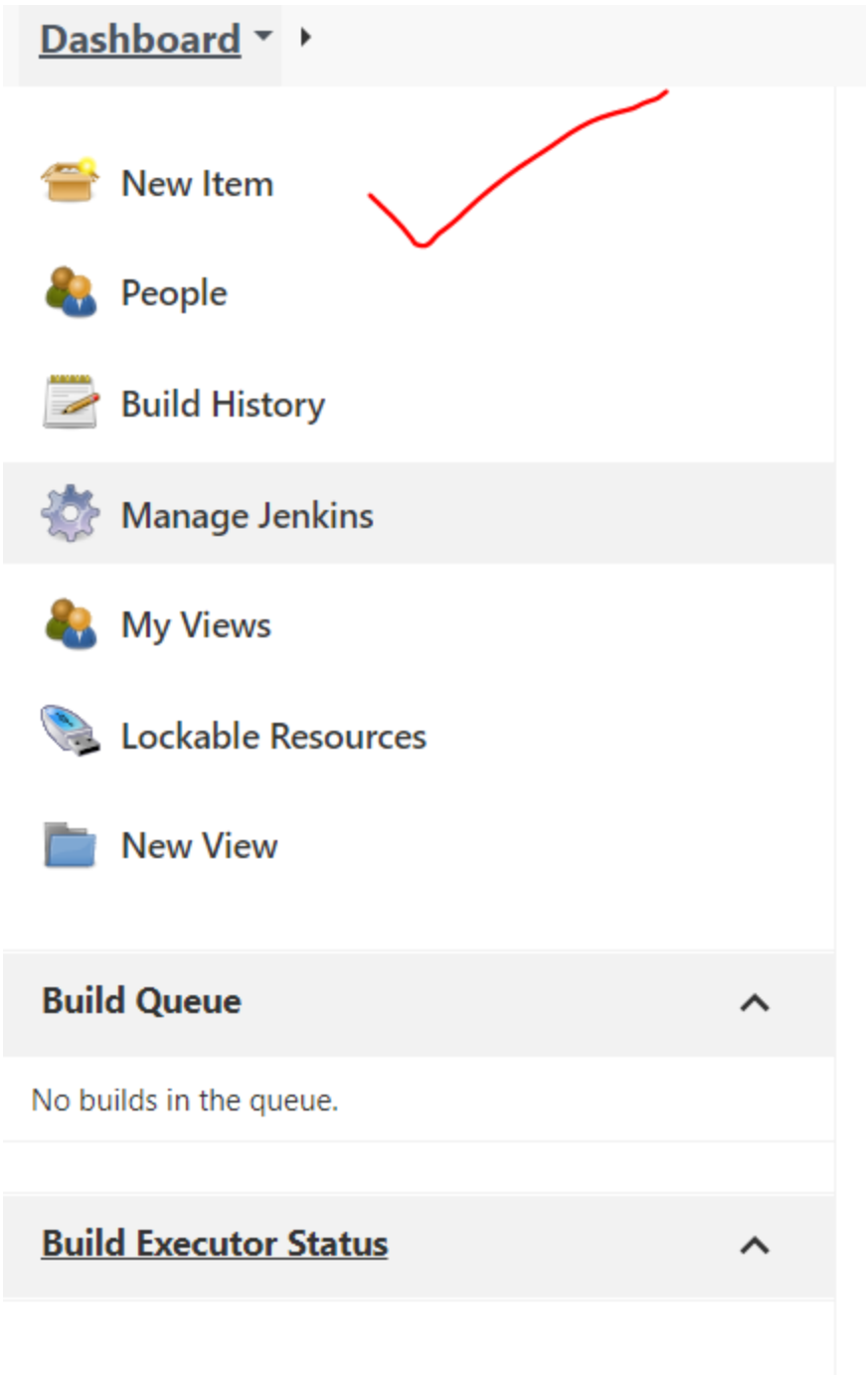

Next ➔

Save



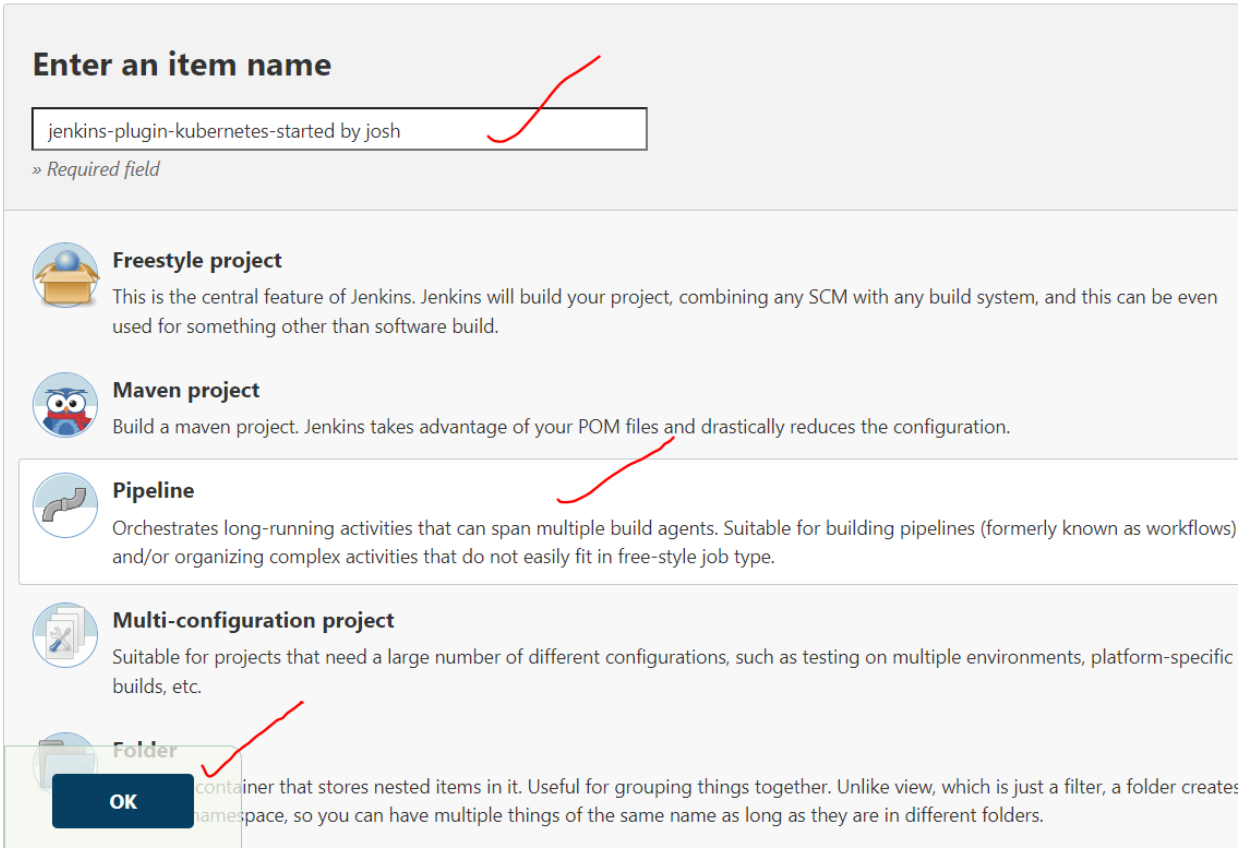

We are ready to create a Dynamic Pipeline

⇨ Go to ➔ manage Jenkins ➔ New Item




Create a name for your pipeline

- ⇨ Select pipeline
- ⇨ ok




Familiarize yourself with build trigger

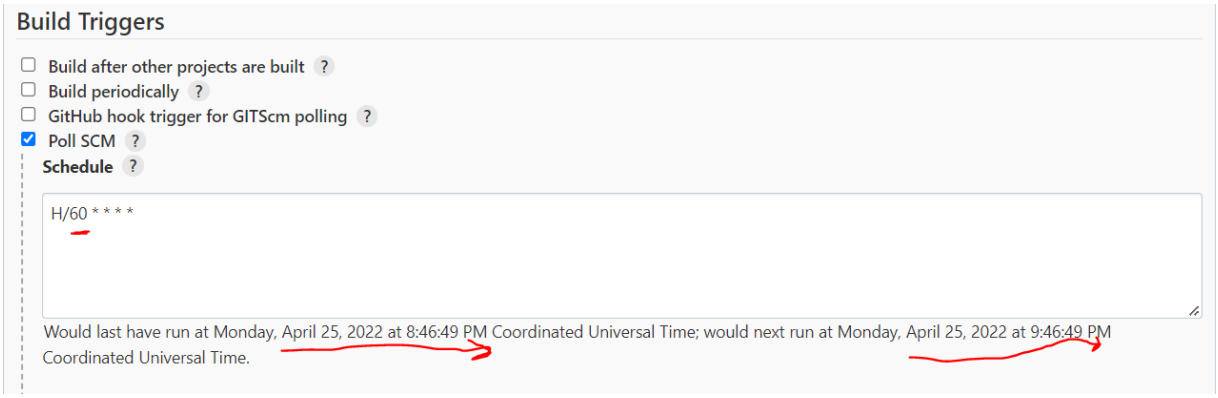



Bild trigger will automatically run a build

In our case, the build will be triggered after every 60 minutes

DevOps Commandment #2

We only run declarative jenkins pipeline script as DevOps engineer.

Never run a scripted jenkins pipeline. This kind of a pipeline has a steep learning curve and companies do not like it.

In addition to that, most tools such as terraform, terragrunt, ansible, and Kubernetes are all based on declarative languages.

Pipeline to copy and paste on the pipeline section

```
pipeline {
  tools {
      maven 'Maven-3.8.4'
  }
  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'
          }
        }
      }
      stage ("Git Checkout") {
        steps {
          git credentialsId: 'GIT_HUB_CREDENTIALS', url:
'https://github.com/joshking1/myRepoForJavaApp.git'
        }
      }
      stage ("Maven Clean Build") {
        steps {
          sh 'mvn clean install -f MyWebApp/pom.xml'
        }
      }
    }
}
```

Pipeline

Definition

Pipeline script

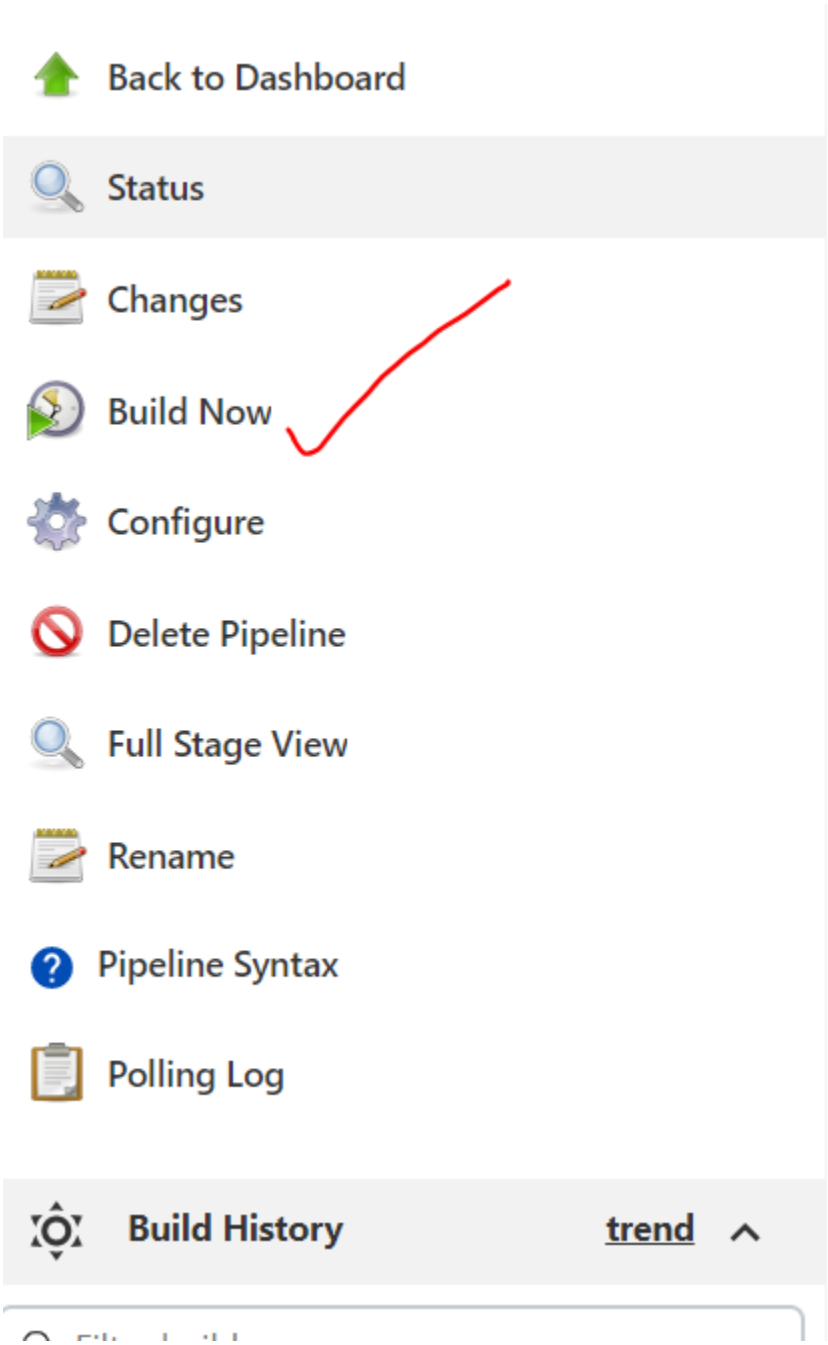
Script ?

try sample Pipeline...

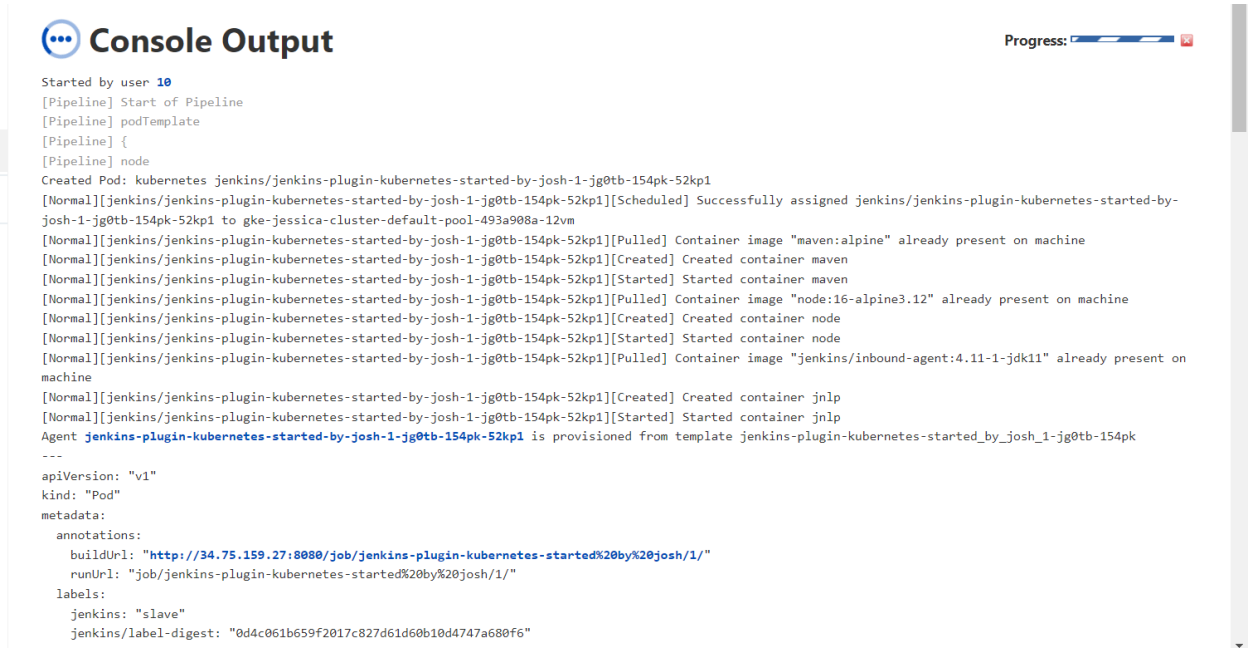
```
pipeline {
  tools {
      maven 'Maven-3.8.4'
  }
  agent {
    kubernetes {
      yaml '''
        apiVersion: v1
        kind: Pod
        spec:
          containers:
          - name: maven
            image: maven:alpine
            command:
            - cat
            tty: true
          - 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'
        }
      }
    }
    stage ("Git Checkout") {
      steps {
        git credentialsId: 'GIT_HUB_CREDENTIALS', url: 'https://github.com/joshking1/myRepoForJavaApp.git'
      }
    }
    stage ("Maven Clean Build") {
      steps {
        sh 'mvn clean install -f MyWebApp/pom.xml'
      }
    }
  }
}
```

Save

Now let us build




Let see the magic of dynamic agents provisioned as a pod by the cluster



## Console Output

Progress:

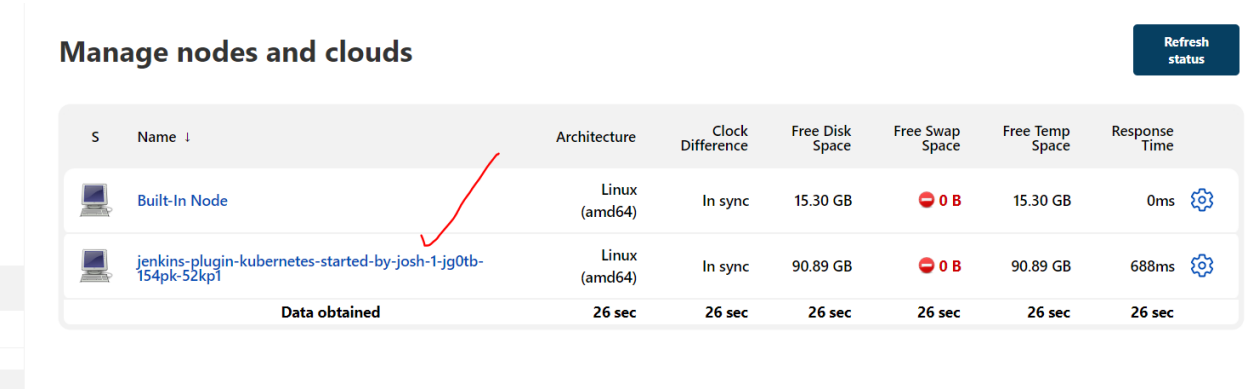
```
Started by user 10
[Pipeline] Start of Pipeline
[Pipeline] podTemplate
[Pipeline] {
[Pipeline] node
Created Pod: kubernetes jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Scheduled] Successfully assigned jenkins/jenkins-plugin-kubernetes-started-by-
josh-1-jg0tb-154pk-52kp1 to gke-jessica-cluster-default-pool-493a908a-12vm
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Pulled] Container image "maven:alpine" already present on machine
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Created] Created container maven
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Started] Started container maven
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Pulled] Container image "node:16-alpine3.12" already present on machine
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Created] Created container node
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Started] Started container node
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Pulled] Container image "jenkins/inbound-agent:4.11-1-jdk11" already present on
machine
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Created] Created container jnlp
[Normal][jenkins/jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1][Started] Started container jnlp
Agent jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1 is provisioned from template jenkins-plugin-kubernetes-started_by_josh_1-jg0tb-154pk
---
apiVersion: "v1"
kind: "Pod"
metadata:
  annotations:
    buildUrl: "http://34.75.159.27:8080/job/jenkins-plugin-kubernetes-started%20by%20josh/1/"
    runUrl: "job/jenkins-plugin-kubernetes-started%20by%20josh/1/"
  labels:
    jenkins: "slave"
    jenkins/label-digest: "0d4c061b659f2017c827d61d60b10d4747a680f6"
```

This is the kind of output you should view

You should have two nodes.

One for the build in node for Jenkins (permanent)

The second one is a jenkins/inbound-agent created by Kubernetes when the job started ( Fried on demand and should disappear after the build is completed)



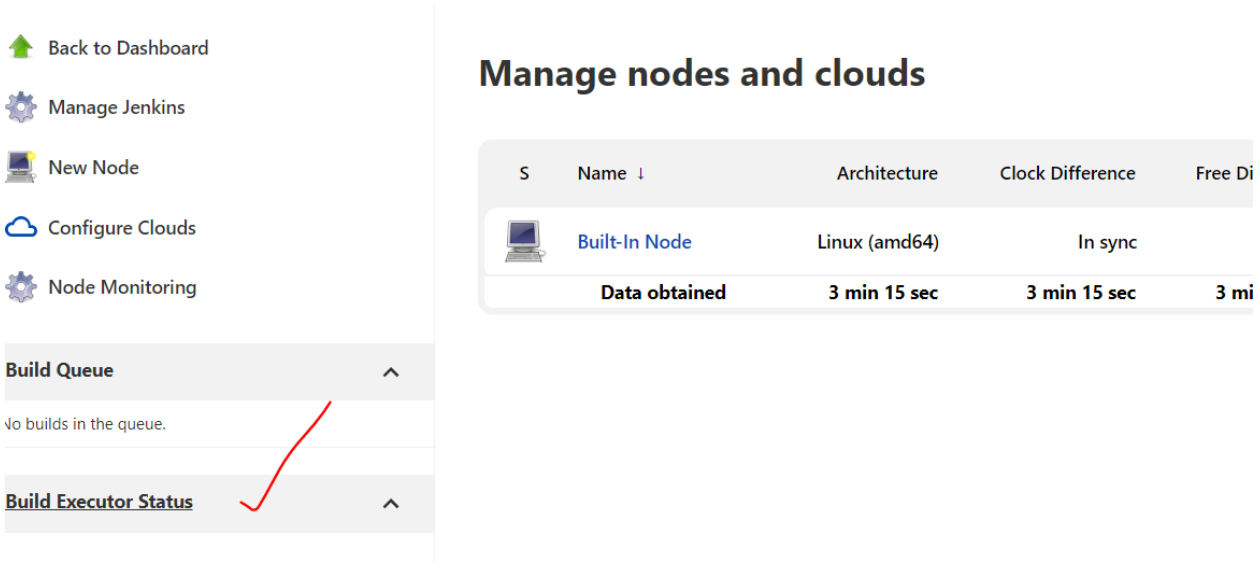
## Manage nodes and clouds

Refresh status

| S | Name ↓ | Architecture | Clock Difference | Free Disk Space | Free Swap Space | Free Temp Space | Response Time |
|---|---|---|---|---|---|---|---|
| | Built-In Node | Linux (amd64) | In sync | 15.30 GB | 0 B | 15.30 GB | 0ms |
| | jenkins-plugin-kubernetes-started-by-josh-1-jg0tb-154pk-52kp1 | Linux (amd64) | In sync | 90.89 GB | 0 B | 90.89 GB | 688ms |
| | Data obtained | 26 sec | 26 sec | 26 sec | 26 sec | 26 sec | 26 sec |

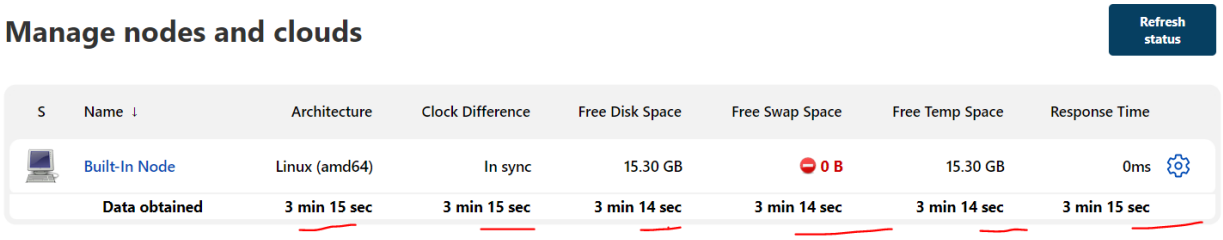
Let confirm the agent has disappeared

⇨ go to executor




YESSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS!!!!!!!!!!!!!!!

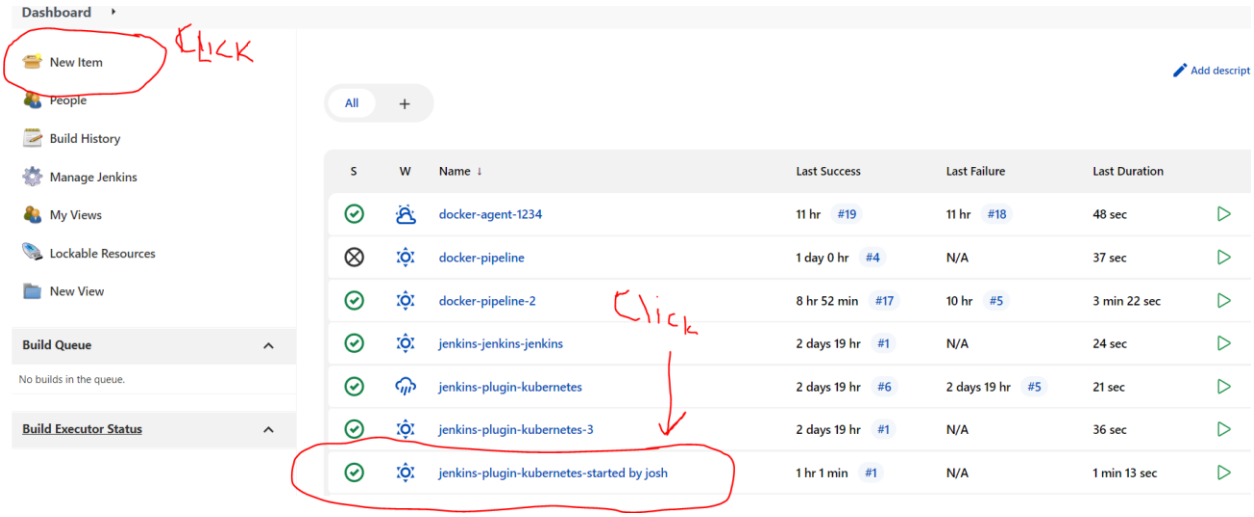
The node created by cluster disappeared



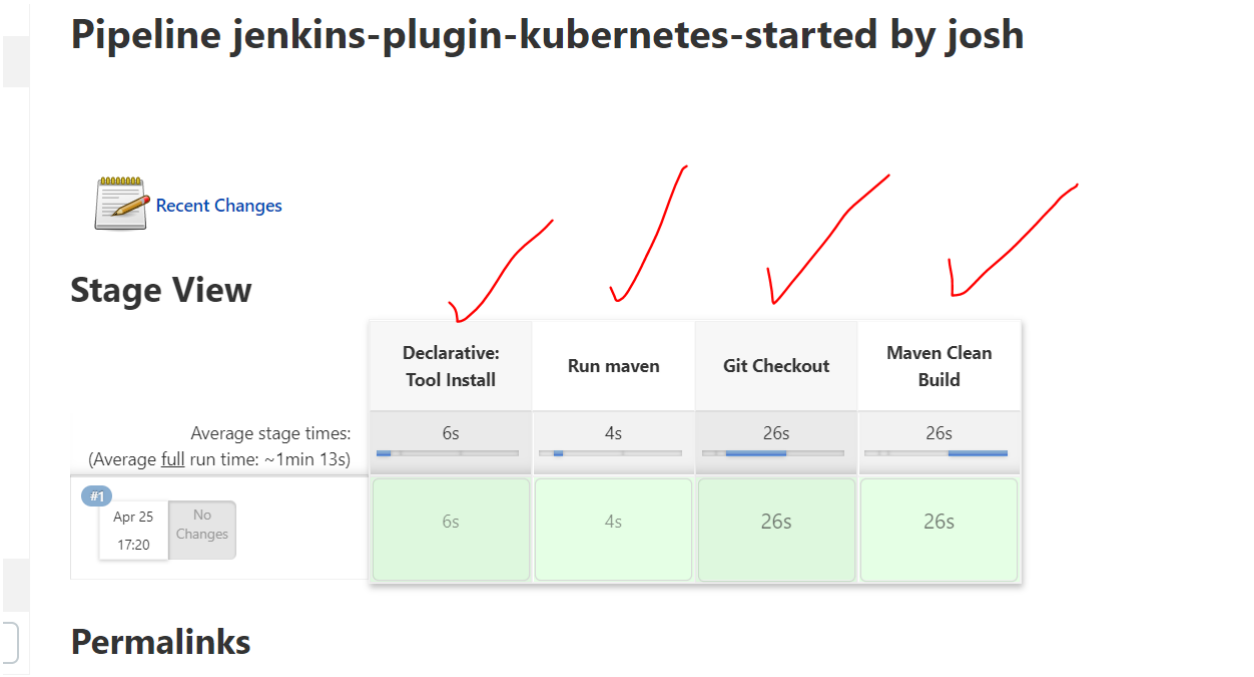

Was the build successfully created?

Let check

Go ➔ Dashboard




You can add more stages if you want



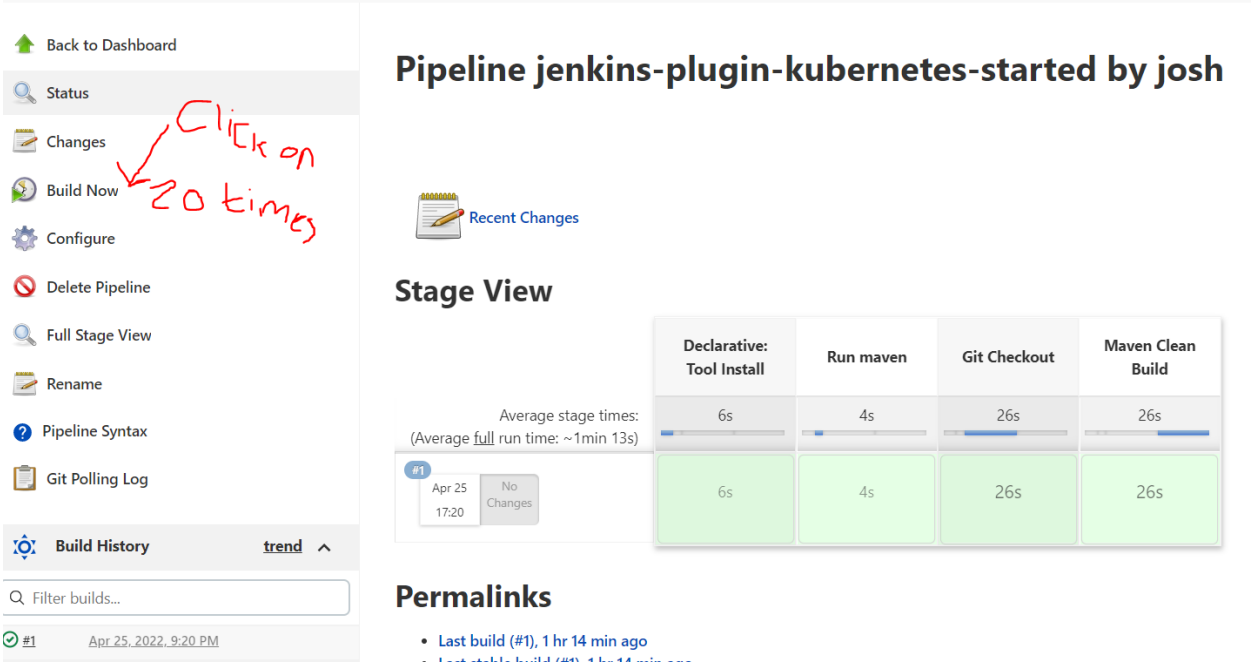
Let enjoy the **DYNAMIC Miracles** of Kubernetes

Follow Me

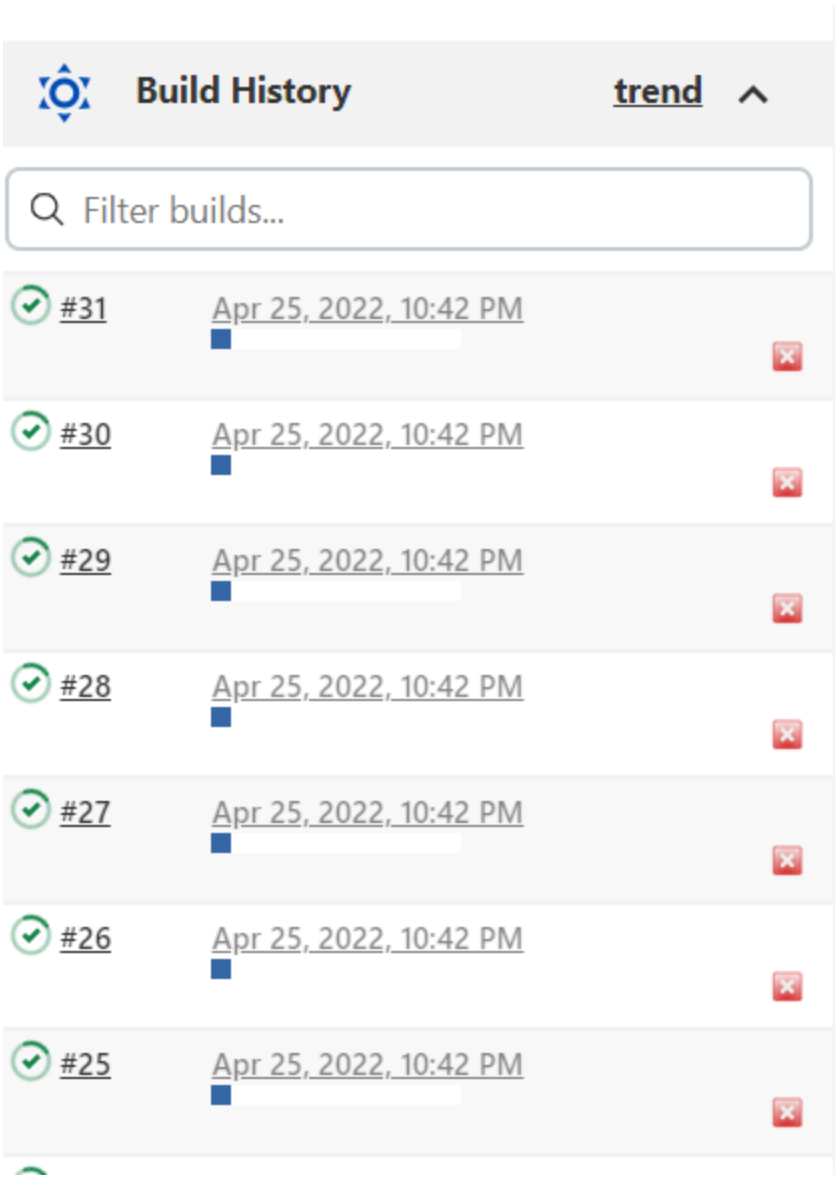
1. Go to ➔ Dashboard

   Select the descriptive pipeline you previously ran

   Look for the Build Now

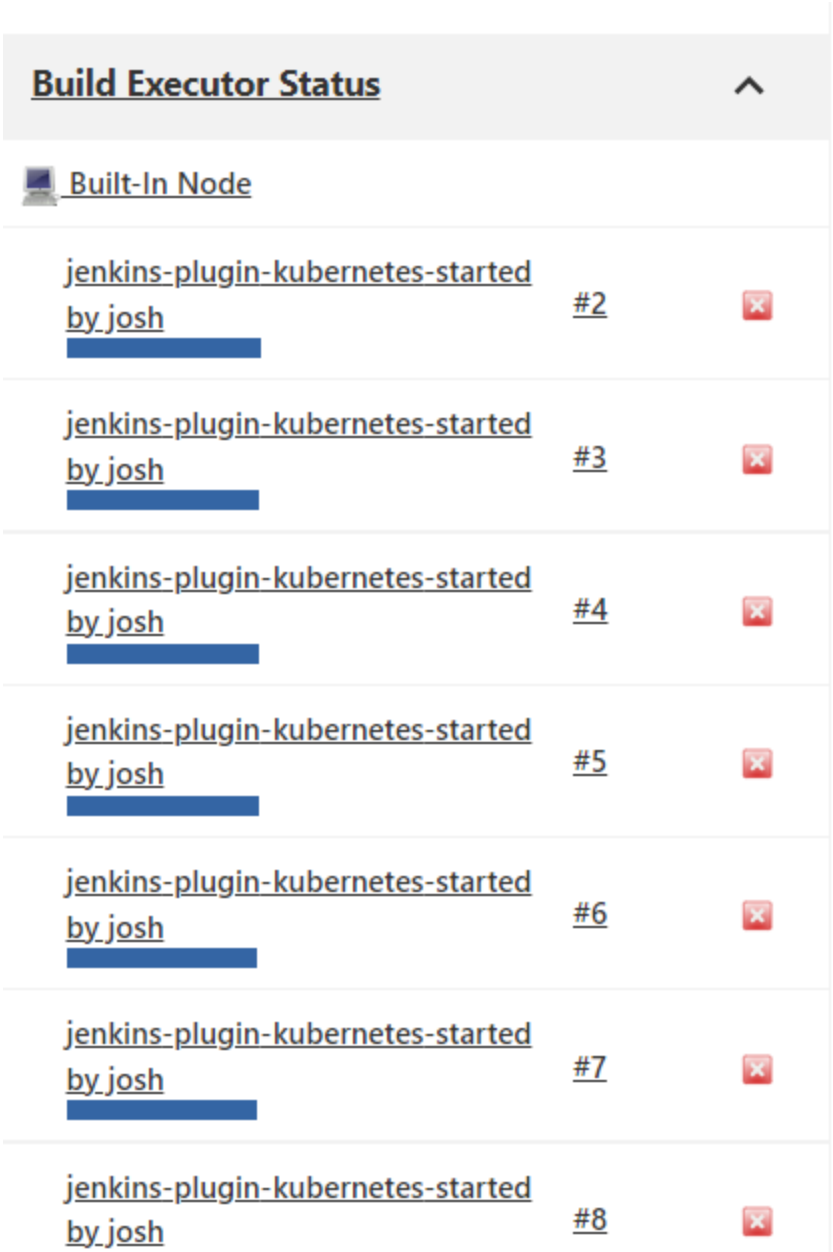



2. Click the Build Now 20 times and you should view jobs aligning




I have 28 builds progress

3. Go to Build Executor Status




For my case 28 agents were provisioned

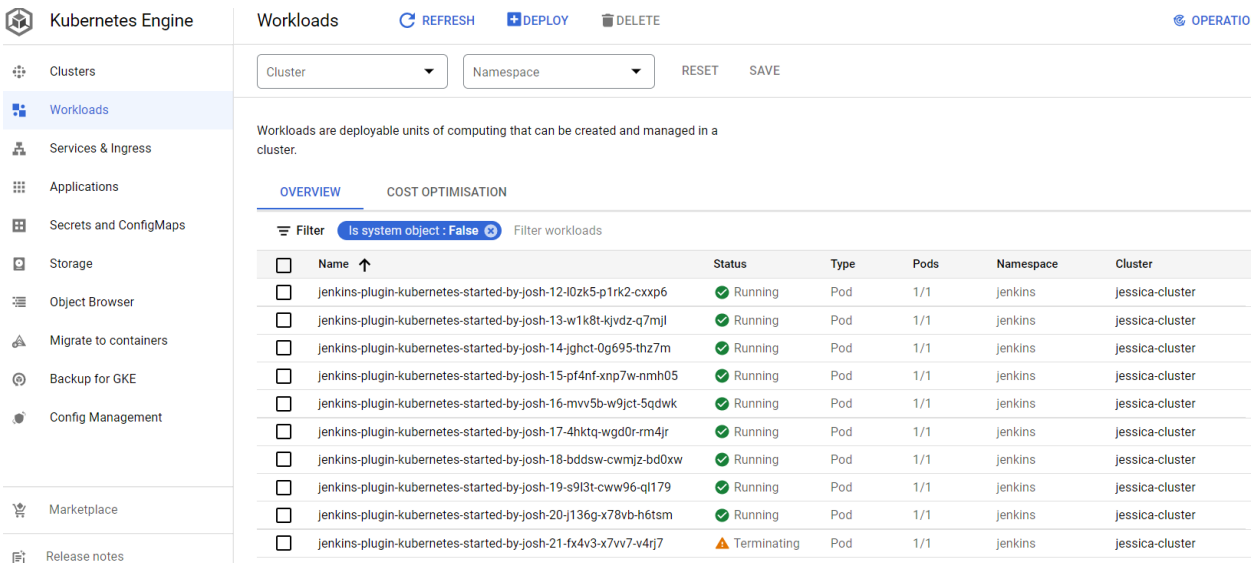
Your number of the Jenkins agents might be different

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Built-In Node | Linux (amd64) | In sync | 15.30 GB | 0 B | 15.30 GB | 0ms | |
| jenkins-plugin-kubernetes-started-by-josh-10-jfftj-qkjrd-k47qg | | N/A | N/A | N/A | N/A | N/A | |
| jenkins-plugin-kubernetes-started-by-josh-11-c30lq-rz578-j2rks | | N/A | N/A | N/A | N/A | N/A | |
| jenkins-plugin-kubernetes-started-by-josh-2-zwdj2-bwl9k-5xdr6 | Linux (amd64) | In sync | 90.80 GB | 0 B | 90.80 GB | 1058ms | |
| jenkins-plugin-kubernetes-started-by-josh-3-lw2wc-ms9r8-p5vts | Linux (amd64) | In sync | 90.80 GB | 0 B | 90.80 GB | 1057ms | |
| jenkins-plugin-kubernetes-started-by-josh-4-7088f-31fv4-g21v4 | Linux (amd64) | In sync | 90.53 GB | 0 B | 90.53 GB | 1057ms | |
| jenkins-plugin-kubernetes-started-by-josh-5-2hsch-1103l-l88rk | Linux (amd64) | In sync | 90.53 GB | 0 B | 90.53 GB | 243ms | |
| jenkins-plugin-kubernetes-started-by-josh-6-pbw77-5fwpt-vcmvf | Linux (amd64) | In sync | 90.80 GB | 0 B | 90.80 GB | 1057ms | |
| jenkins-plugin-kubernetes-started-by-josh-7-xcr5l-hz0v9-wt87g | Linux (amd64) | In sync | 90.80 GB | 0 B | 90.80 GB | 1056ms | |
| jenkins-plugin-kubernetes-started-by-josh-8-z3t3r-qw8g3-615s9 | Linux (amd64) | In sync | 90.53 GB | 0 B | 90.53 GB | 242ms | |
| jenkins-plugin-kubernetes-started-by-josh-9-8s2kz-1xt3g-xzqkr | Linux (amd64) | In sync | 90.53 GB | 0 B | 90.53 GB | 601ms | |
| **Data obtained** | **43 sec** | **43 sec** | **42 sec** | **41 sec** | **42 sec** | **43 sec** | |

Each job that is building is assigned a special and dynamic node.

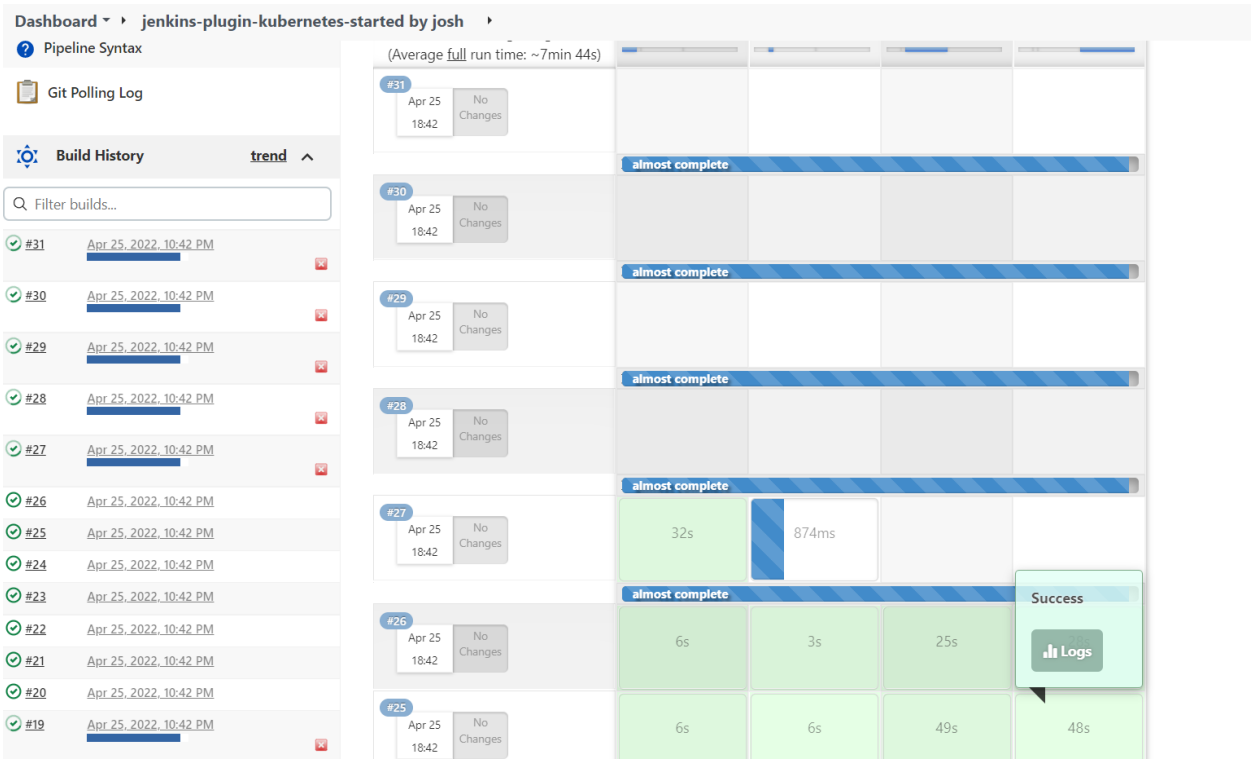
In total you should have nodes that are equal to the number of jobs building simultaneously.

If you go to your Google Kubernetes cluster, and they click on Workload, you should be able to view pods running and terminating at the same time.



Finally, you can check the Jenkins interface and confirm the progress of the builds/

In my case, I have 31 builds occurring simultaneously.




One final check.

We had 31 nodes running simultaneously and all of them were terminated after the completion of the build process.