<https://github.com/kodekloudhub/go-webapp-sample>

CI – CD

CI is doing the build process of the application and tests it (includin security checks) before it gives it to CD

CD(continuos delivery) = after the CI process it requires manual intervention until it deploys the application

CD(continuos depolyment) = the deployment does not require manual intervention

Installing Jenkins and dependencies(java):

These commands are used to set up Jenkins on a Red Hat-based system by installing the necessary dependencies, configuring the Jenkins repository, and then installing Jenkins itself. Let's break down each command:

1. **sudo yum install epel-release -y**:
   * Installs the Extra Packages for Enterprise Linux (EPEL) repository. EPEL provides additional software packages that are not included in the default repositories for Red Hat-based systems.
2. **sudo yum install java-11-openjdk -y**:
   * Installs OpenJDK 11, which is a required dependency for running Jenkins. Jenkins is a Java-based application, and it needs a Java Runtime Environment (JRE) to execute.
3. **sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo --no-check-certificate**:
   * Downloads the Jenkins repository configuration file (**jenkins.repo**) and saves it in the **/etc/yum.repos.d/** directory. The **--no-check-certificate** flag is used to bypass SSL certificate checks during the download.
4. **sudo rpm --import http://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key**:
   * Imports the Jenkins repository key, which is used to verify the authenticity of the packages obtained from the Jenkins repository.
5. **sudo yum install jenkins -y**:
   * Installs the Jenkins package using the YUM package manager. The **-y** flag is used to automatically answer "yes" to any prompts that may come up during the installation process.

After these commands are executed, Jenkins should be installed on the system. You can typically access the Jenkins web interface by navigating to http://localhost:8080 or the specific address where Jenkins is configured to run

Edit /lib/systemd/system/jenkins.service file and change Jenkins port to 8090 by updating Environment="JENKINS\_PORT=" variable value:

sudo sed –i ‚’s/8080/8090/g’ /lib/systemd/system/jenkins.service

Once done start Jenkins service: sudo systemctl start jenkins

Open up the firewall so that we are allowed to interact with Jenkings from the UI: **sudo ufw allow 8080**

**Check it: sudo ufw status**

**After you hit the password pannel in the UI, get the password: cat /lib/jenkins/secrets/initialAdminPassword**

**Create first admin user**

Jenkins also has a CLI that can be connected to through curl. User->configure->SSH Pulic Key

cat id\_rsa.pub -> copy it, save it and apply

**To find on which port is the Jenkins server configured to run on:**

**curl -Lv http://localhost:8085/login 2>&1 | grep -i 'x-ssh-endpoint'**

**Ssh into Jenkins:**

**ssh -i /home/mike/.ssh/jenkins\_key -l mike -p 8022 jenkins-server help**

**Keep a note of the options used with the SSH command:**

1. **-i flag is used to point to mike's private SSH key. Remember, we have already added the public key in the Jenkins configuration.**
2. **-l is the login user which in our example is mike**
3. **-p is the port which we found out in the previous step to be 8022**

**After you install a new plugin, you may need to restart Jenkins through sudo systemctl restart jenkins**

**Print Jenkins CLI commands: java -jar jenkins-cli.jar -s http://localhost:8085 -auth 'admin:Adm!n321'**

**Role based Authorization Strategy plugin to manage or assign roles**

**$JENKINS\_HOME folder – here are the configuration files of Jenkins (config.xml) and jobs folder**

**Jenkins Backup is done either through snapshots which is not recommended, through a backup plugin (Google Cloud Backup, Thin Backup) or through a shell script (github)**

**Monitoring Jenkins:**

**- With DataDog**

**- With Prometheus and Grafana**

- job\_name: 'Jenkins'

metrics\_path: /prometheus/

static\_configs:

- targets: ['localhost:8085']

**for Prometheus with Jenkins**

**root@jenkins-server ~ ➜ sudo cat /etc/prometheus/prometheus.yml**

**global:**

**scrape\_interval: 15s**

**scrape\_configs:**

**- job\_name: 'Prometheus'**

**scrape\_interval: 5s**

**static\_configs:**

**- targets: ['localhost:9090']**

**- job\_name: 'Jenkins'**

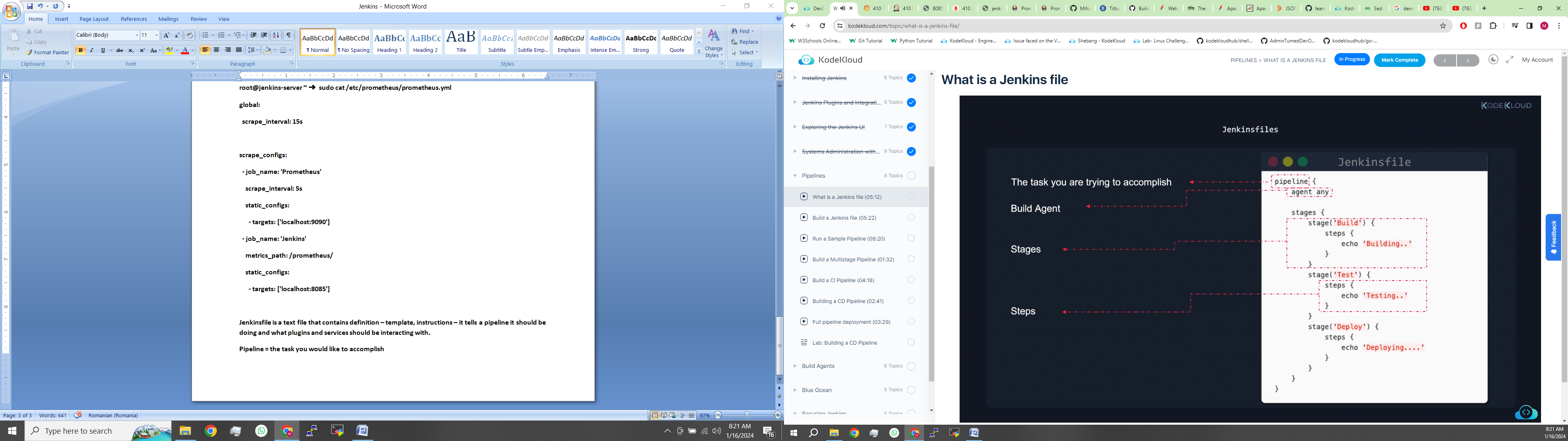
**metrics\_path: /prometheus/**

**static\_configs:**

**- targets: ['localhost:8085']**

**A Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline – template, instructions – it tells a pipeline it should be doing and what plugins and services should be interacting with.**

**Pipeline = the task you would like to accomplish**

****

[**https://github.com/kodekloudhub/go-webapp-sample.git**](https://github.com/kodekloudhub/go-webapp-sample.git)

We have a pipeline job named go-test but its incomplete. Complete the same by adding the required stages/steps as per details mentioned below:

**1.** Clone a git [repository](https://github.com/kodekloudhub/go-webapp-sample).

git 'https://github.com/kodekloudhub/go-webapp-sample.git'

**2.** Run a shell command go test ./...

sh 'go test ./...'

pipeline {

agent {

label {

label 'master'

customWorkspace "${JENKINS\_HOME}/${BUILD\_NUMBER}/"

}

}

environment {

Go111MODULE='on'

}

stages {

stage('Test') {

steps {

git 'https://github.com/kodekloudhub/go-webapp-sample.git'

sh 'go test ./...'

}

}

}

}

3. Remove the sh 'go test ./...' step.  
  
4. Add an another stage (name it as per your choice) to build a docker image from the cloned code. The required Jenkins plugins have already been installed i.e Docker and Docker pipeline. The image should be tagged as adminturneddevops/go-webapp-sample

5. Add an another step to run a docker container using the docker image you are building in this pipeline itself. Make sure to map container port 8000 with docker host port 8090. This is the step you can add:

pipeline {

agent {

label {

label 'master'

customWorkspace "${JENKINS\_HOME}/${BUILD\_NUMBER}/"

}

}

environment {

**Go111MODULE**='on'

}

stages {

**stage**('Test') {

steps {

git 'https://github.com/kodekloudhub/go-webapp-sample.git'

}

}

**stage**('build') {

steps {

script{

image = docker.**build**("adminturneddevops/go-webapp-sample")

sh "docker run -p 8090:8000 -d adminturneddevops/go-webapp-sample"

}

}

}

}

}

**1.Agent - This section specifies where the pipeline should run. In this case, it's configured to run on a Jenkins agent labeled 'master', and it sets a custom workspace based on the Jenkins home directory and the build number.**

**2. Environment - This block sets environment variables for the pipeline. Here, it sets the Go111MODULE variable to 'on'. This is likely related to Go modules, indicating that the Go module support is enabled.**

**3. Stages - This pipeline has one stage named 'Test', and within this stage, there are two steps:**

* **git '**[**https://github.com/kodekloudhub/go-webapp-sample.git**](https://github.com/kodekloudhub/go-webapp-sample.git)**': This step checks out the source code from the specified Git repository (https://github.com/kodekloudhub/go-webapp-sample.git). It fetches the code to the workspace, which is configured earlier in the agent section.**
* **sh 'go test ./...': This step executes a shell command using the sh step. It runs the go test command for the Go project. The ./... represents a wildcard that includes all packages in the project. This step is likely used to run tests for the Go application.**

**Overall, this Jenkinsfile defines a pipeline that checks out a Go application from a Git repository, sets up the environment for Go modules, and then runs the tests for the application. It assumes that the Jenkins instance has a label 'master' for an agent and customizes the workspace based on the Jenkins home directory and build number.**

**A build agent is typically a machine, or container, which connects to a Jenkins controller and executes tasks when directed by the controller.**

**Configuring a Build Agent (slave):**

**Install Jenkins (through the above steps)**

**Create a new user: sudo adduser <name>**

**Add the user in the sudo group: sudo usermod –aG sudo <name>**

**Check if the user is in the sudo group: groups newuser**

**Add credentials in Jenkins UI**

**Manage nodes ad clouds: new node; remote directory: /home/<username; Launch method: Launch agents via SSH, Host – the ip or dns name of the agent server, Host Key Verification Strategy: manually trusted key -> Save**

**Verify if a build agent works:**

**Create a new item -> freestyle project named test -> Restrict where this project can be run with the label expression the same as the agent name – test -> go to built step and select execute shell -> echo ‘test’ -> save -> build**

**Pipeline script:**

pipeline {

agent {

label "node01"

}

stages {

**stage**('Hello') {

steps {

echo 'Hello World'

}

}

}

}

**Pipeline script for a docker agent:**

pipeline {

agent {

docker { image ‘golang:latest’ }

}

stages {

**stage**('Hello') {

steps {

git ‘git url…’

sh ‘go version’

}

}

}

}

**Blue Ocean – visual enhancing of Jenkins – it works by adding /blue to the Jks URL, advantages:**

**- easier to use**

**- sophisticated visualization**

**- fast and intuitive pipeline status**

**- pipeline editor (kind of replaces Jenkinsfile)**

**- personalization for the role-based needs of team members**

**- pinpoint precision when intervention is needed or when issue arise**

**- native integration for branch and pull requests**

**Jenkins Security:**

**1. User and access Control:**

**A. Authentication (users prove who they are) done using Security realm:**

**a) user identity**

**b) group membership**

**2. Authorization (user permitted to actually do something) done by using a authorization strateg:**

**A. Permissions**

**Common Jenkins Security mistakes:**

**- anyone can do anything**

**- logged-in users can do anything**

**- anonymous and authenticated users**

**- built in-node**

**Jenkins security:**

**- Configure Global Security**

**- Manage Credentials**

**- Configure Credential Providers**

**- Manage Users**

**Security plugin in regards to access control for builds: Pipeline: …**

**Popular Jenkins Plugins: Matrix authorization**

**Project:**

Our application development team has recently finished the MVP development for one of the applications and it's ready to be deployed on production.  
  
The go-app repository on the Gitea server contains the source code of this application. The dev branch of this repository is used for testing, hence we want to deploy it on the dev environment and the master branch should be deployed on the production environment.  
  
On dev server /opt/go-app is the document root of the application and we have already setup the dev branch over there for your convenience. In your build step, you will just need to pull the latest dev branch code from the repository using git pull command. You can access the dev server using below given credentials:  
  
Host: gotest-dev01  
  
Similarly on production server /opt/go-app is the document root of the application and we have already setup the master branch over there for your convenience. In your build step, you will just need to pull the latest master branch code from the repository using git pull command. You can access the production server using below given credentials:  
  
Host: gotest-prod01  
  
Create a pipeline job in Jenkins, and configure it as per details mentioned below:

Prerequisites:  
  
-> Install the required plugins which are needed for adding SSH agent nodes and for creating pipeline jobs.  
  
-> Add dev and production servers as SSH node agents in Jenkins.  
  
-> You can use /opt as the Remote root directory for both of these nodes to avoid any permission issues.  
  
-> To make nodes work, you will also need to add the respective credentials in Jenkins for these servers.  
  
  
Job configuration:  
  
-> The job name should be go-app-deployment  
  
-> It must be a pipeline job.  
  
-> The first stage here should be to build the dev branch code which must be deployed on the dev server/environment, you can name the stage as per your choice. As we have already setup the repo/branch on the server so you will just need to run the git pull command.  
  
Here is a hint, how you can make it run on the dev server. To do so, add an agent configuration in each of your stages like this:

Please note that label 'dev' is the label you will give to your node (so update it accordingly) and you will need to set customWorkspace "/opt/go-app" (its same for both of the servers/environments) to make it work on this setup/back-end.  
  
-> The second stage should be to run some unit tests (again you can name the stage as per your choice). You can execute the tests using a stage like:  
-> The last stage for dev would be to deploy the application. You can use a stage like:  
You must have noticed an extra parameter in this stage i.e withEnv ( ['JENKINS\_NODE\_COOKIE=do\_not\_kill'] ), it is needed to make sure Jenkins doesn't kill the background process we are starting in this stage on the respective server.  
-> Similarly configure the same stages (with different name and prod specific values) for the production deployment. So the pipeline will have total 6 stages (three for each environment).  
-> Make sure to build the job at least once and make sure its building successfully.

Login into the Jenkins server and follow the below given steps:  
  
Install Plugins:  
  
**1.** Go to Manage Jenkins.  
**2.** Click on Plugins.  
**3.** Under Available, search for SSH Build Agents plugin and select it.  
**4.** Now search for Pipeline plugin and select it.  
**5.** Now install these plugins.  
**6.** Once installed click on Restart Jenkins when installation is complete and no jobs are running.  
  
  
Add Credentials:  
  
**1.** Go to Manage Jenkins.  
**2.** Click on Credentials.  
**3.** Click on (global) under Domains.  
**4.** From the option on the right side, click on Add Credentials.  
**5.** Enter bob under Username.  
**6.** Enter caleston123 under Password.  
  
**7.** Leave other options as it is and click on OK.  
  
  
Add Nodes:  
  
**1.** Go to Manage Jenkins.  
**2.** Click on Nodes and Clouds.  
**3.** From the option available on the right side, click on New Node.  
  
**4.** Enter the Node name i.e dev.  
  
**5.** Enable Permanent Agent option and click on OK.  
  
**6.** Enter /opt under Remote root directory.  
  
**7.** Enter dev under Labels.  
  
**8.** Select Launch Agents via SSH under Launch method.  
  
**9.** Enter gotest-dev01 under Host and select the credentials you created.  
  
**10.** Under Host Key Verification Strategy, select Non verifying verification strategy.  
  
**11.** Leave all other options as it is and click on Save  
  
**12.** Click on the dev node and there shouldn't be any errors.  
  
**13.** Just in case the dev node is in error state, then try to relaunch it.  
  
**14.** Follow same steps for adding prod node, just take care about the node name, label and host.  
  
  
Create and Configure the job:  
  
**1.** On the left of the Jenkins dashboard, click on New Item.  
  
**2.** Enter the job name go-app-deployment.  
  
**3.** Select Pipeline job and click on OK.  
  
**4.** Under Pipeline section keep selected Pipeline script as Definition and add below given code in the Script

pipeline {

agent none

stages {

//DEV

**stage**('Build Dev') {

agent {

label {

label 'dev'

customWorkspace "/opt/go-app"

}

}

steps {

sh 'git pull'

}

}

**stage**('Test Dev') {

agent {

label {

label 'dev'

customWorkspace "/opt/go-app"

}

}

steps {

sh 'go test ./...'

}

}

**stage**('Deploy Dev') {

agent {

label {

label 'dev'

customWorkspace "/opt/go-app"

}

}

steps {

script {

withEnv ( ['JENKINS\_NODE\_COOKIE=do\_not\_kill'] ) {

sh 'go run main.go &'

}

}

}

}

//PROD

**stage**('Build Prod') {

agent {

label {

label 'prod'

customWorkspace "/opt/go-app"

}

}

steps {

sh 'git pull'

}

}

**stage**('Test Prod') {

agent {

label {

label 'prod'

customWorkspace "/opt/go-app"

}

}

steps {

sh 'go test ./...'

}

}

**stage**('Deploy Prod') {

agent {

label {

label 'prod'

customWorkspace "/opt/go-app"

}

}

steps {

script {

withEnv ( ['JENKINS\_NODE\_COOKIE=do\_not\_kill'] ) {

sh 'go run main.go &'

}

}

}

}

}

}

**5.** Finally save the job and build.

**This Jenkinsfile defines a more complex pipeline with different stages for building, testing, and deploying a Go application in both a development (Dev) and production (Prod) environment. Let's break it down:**

1. **Agent:**

**The agent none line means that there is no default agent specified at the top level of the pipeline. Each stage will explicitly define its own agent, specifying where it should run.**

1. **Stages: The pipeline has multiple stages, each representing a phase in the CI/CD process.**
   1. **Development Environment (Dev):**
      1. **Build Dev: This stage checks out the source code in the '/opt/go-app' directory on a Jenkins agent labeled 'dev', pulls the latest changes from the Git repository, and performs the build process.**
      2. **Test Dev: This stage runs the tests for the Go application in the '/opt/go-app' directory on the 'dev' agent.**
      3. **Deploy Dev: This stage deploys the Go application in the '/opt/go-app' directory on the 'dev' agent. It sets an environment variable to prevent the Jenkins node from being terminated during the deployment. It then runs the application using the 'go run main.go &' command.**
   2. **Production Environment (Prod): The stages for the production environment are similar to the development environment, with the same structure (Build, Test, Deploy), but the agents are labeled 'prod' instead of 'dev'.**
2. **Summary: The pipeline performs a Git pull, runs tests, and deploys a Go application in both the development and production environments. It uses separate agents for 'dev' and 'prod' labeled environments and sets a custom workspace for each stage. Additionally, it prevents Jenkins from killing the node during deployment using the JENKINS\_NODE\_COOKIE environment variable.**