**Wizard's Almanac’s Jenkin CICD pipeline user manual**

1. **Advantages of building a Jenkin CICD pipeline for Wizard's Almanac application**

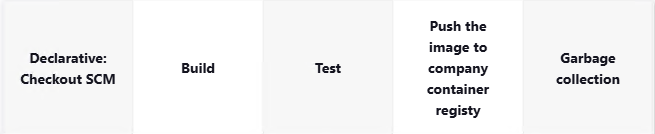
A CICD pipeline helps developers and companies the followings:

* Faster Development and Deployment: Automating repetitive tasks like testing and deployment, CI/CD accelerates the development cycle, enabling teams to release new features, bug fixes, and updates more quickly and frequently.
* Improved Code Quality: Continuous Integration ensures that new code is regularly merged into the main codebase and automatically tested for issues, which helps catch bugs early. This leads to higher-quality software and fewer issues during production.
* Automated Testing: With automated tests integrated into the CI/CD pipeline, developers can quickly verify that new code does not break existing functionality. This reduces the risk of introducing errors and ensures a more stable product.
* Reduced Manual Errors: Automating the build, test, and deployment processes minimizes the risk of human error. This ensures that releases are more predictable and stable.
* Consistent Environments: CI/CD ensures that code is tested and deployed in the same environment it will run in production, reducing issues related to environment differences (e.g., "it works on my machine" problems).\

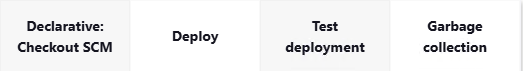
1. **How this Wizard's Almanac’s Jenkin CICD pipeline works**

The Wizard's Almanac’s Jenkin CICD pipeline is divided into two parts: The CI pipeline and the CD pipeline.

* CI pipeline: Used for continuous integration. The input of this pipeline is the source code from git repo and the output is a complete image which has been tested and pushed to company container registry. The CI pipeline is configured by the **jenkin-CI** file in the git repo.
  + There are five stages in CI pipeline: Checkout SCM (Check and pull source code from git repo) > Build (build image with docker file) > Test (run container and test the app) > Push the image to company container registry (publish image) > Garbage collection (remove all garbage left behind)



* CD pipeline: Used for continuous delivery. The input of this pipeline is the built image from the CI pipeline and the output is the deployment of the image onto an environment. The deployment is setup by docker compose. This CD pipeline is only run after an approval is secured. The CD pipeline is configured by the **jenkin-CD** file in the git repo
  + There are 4 stages in the CD pipeline: Checkout SCM (Check and pull source code from git repo) > Deploy (deploy the image with docker compose) > Test deployment (test the deployment with public IP of the app) > Garbage collection (remove all garbage left behind)



1. **How to setup jenkin to run these pipelines**

In this documentation, jenkin is run using docker. There are two ways to achieve this:

* The short way: pull and run the official jenkin image

|  |
| --- |
| docker run \  --name jenkins-pipeline \  --restart=always \  --detach \  --publish 8080:8080 \  --publish 50000:50000 \  --volume jenkins-data:/var/jenkins\_home \  --volume jenkins-docker-certs:/certs/client:ro \  jenkins/jenkins:lts-jdk17 |

* The long way: build jenkin image with dockerfile and run the image. This allows more control over jenkin‘s container and application.
  + Create and name this as Dockerfile-jenkin:

|  |
| --- |
| # Example Jenkins Dockerfile, modify this to complete the challenge  FROM jenkins/jenkins:lts  USER root  RUN apt-get update && apt-get install -y lsb-release  RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \  https://download.docker.com/linux/debian/gpg  RUN echo "deb [arch=$(dpkg --print-architecture) \  signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \  https://download.docker.com/linux/debian \  $(lsb\_release -cs) stable" > /etc/apt/sources.list.d/docker.list  RUN apt-get update && apt-get install -y docker-ce-cli  USER jenkins  RUN jenkins-plugin-cli --plugins "blueocean docker-workflow" |

* + Build the jenkin image with the following command:

|  |
| --- |
| docker build -t myjenkins:lts -f Dockerfile-jenkin . |

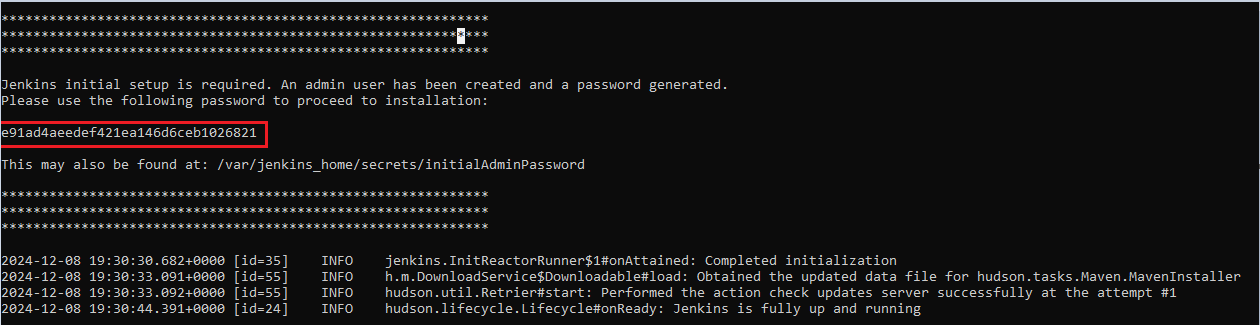
* + After build done, run the jenkin container with the following command:

|  |
| --- |
| docker run \  --name jenkins-pipeline \  --restart=always \  --detach \  --publish 8080:8080 \  --publish 50000:50000 \  --volume jenkins-data:/var/jenkins\_home \  --volume jenkins-docker-certs:/certs/client:ro \  myjenkins:lts |

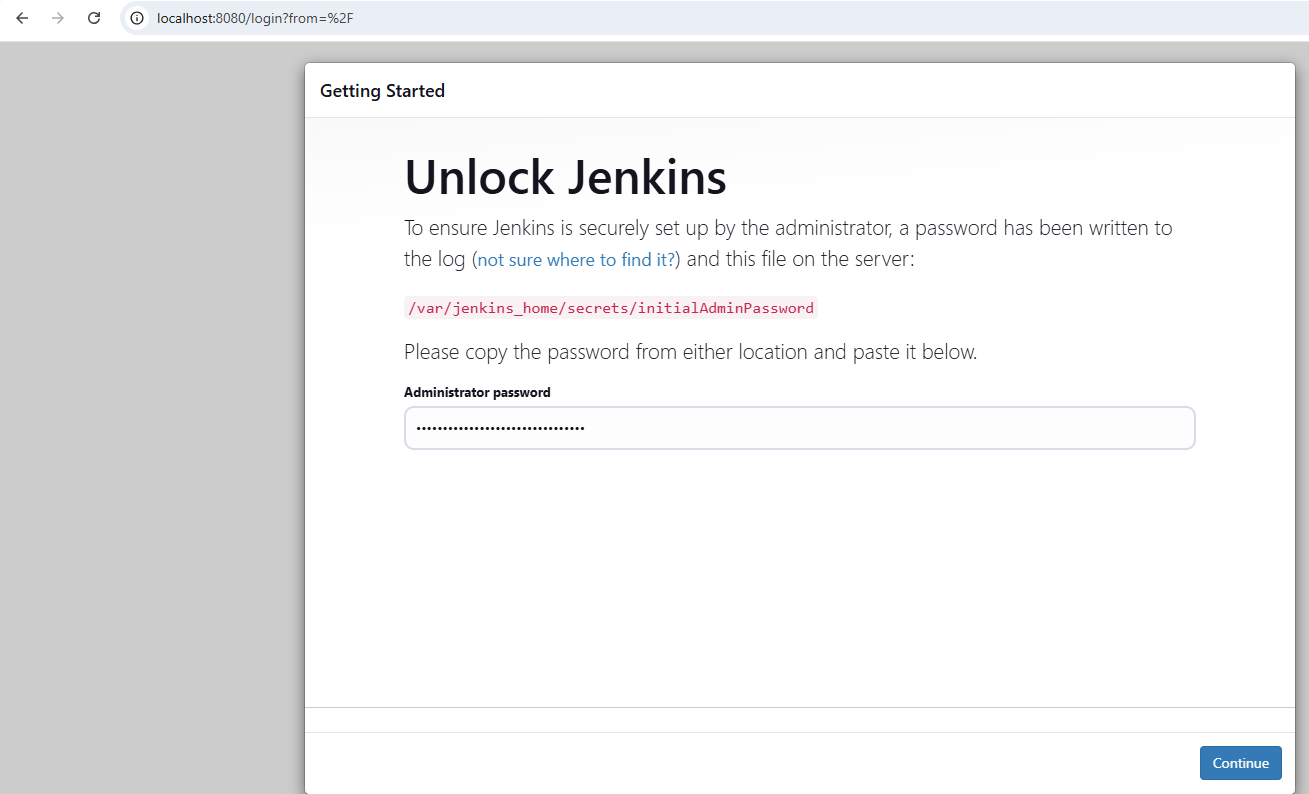
Next, open web browser on [**http://localhost:8080/**](http://localhost:8080/)**.** Using this command to get jenkin’s initial password:

|  |
| --- |
| docker logs jenkins-pipeline |

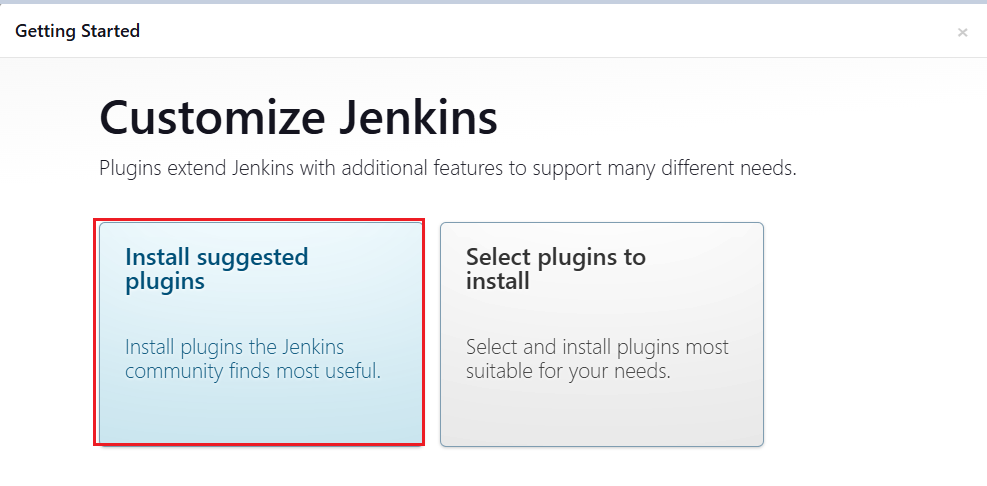
Password should be found here:



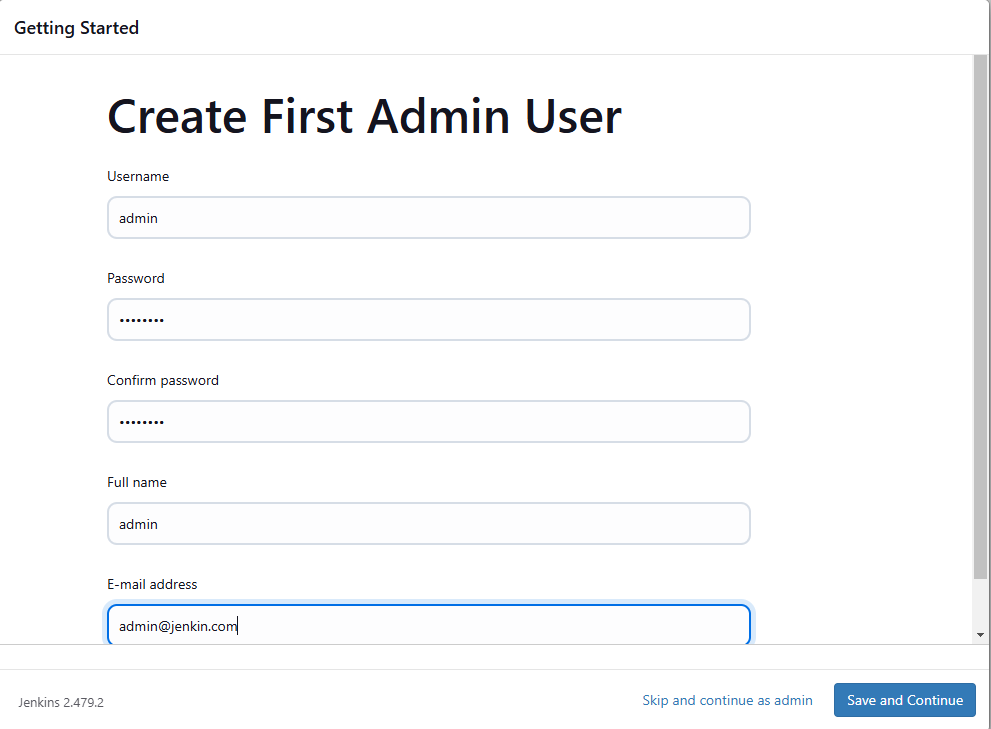
Then, insert this password into jenkin console to start the setup process:



Choose install suggested plugins:

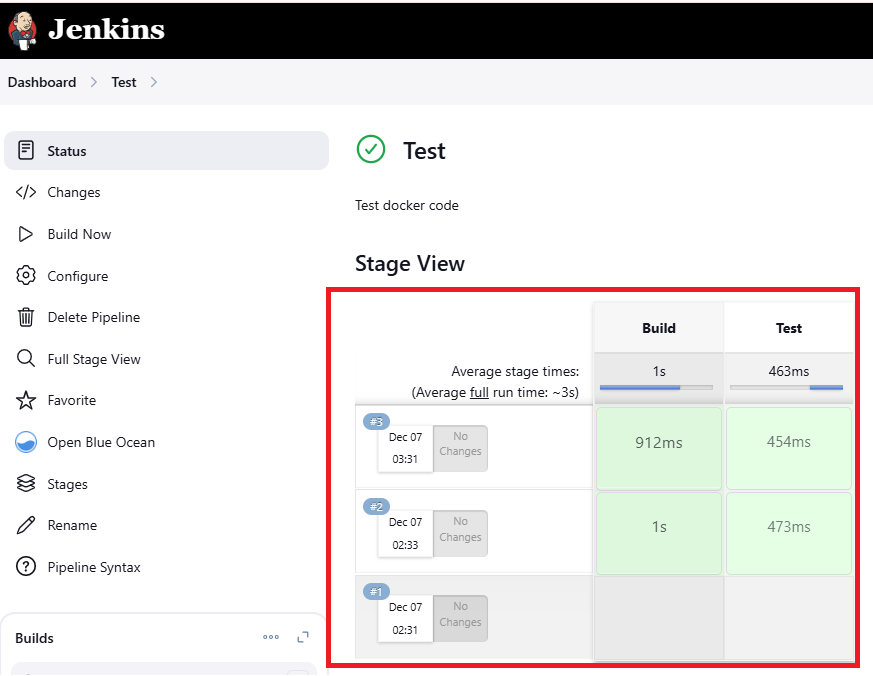


Setup username and password:

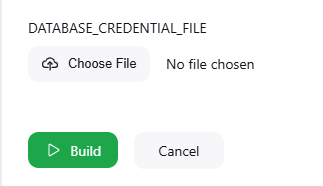


After the initial setup is complete, Install the following jenkin plugins:

* **Pipeline: Stage View** – This plugin is used to visualize stages.



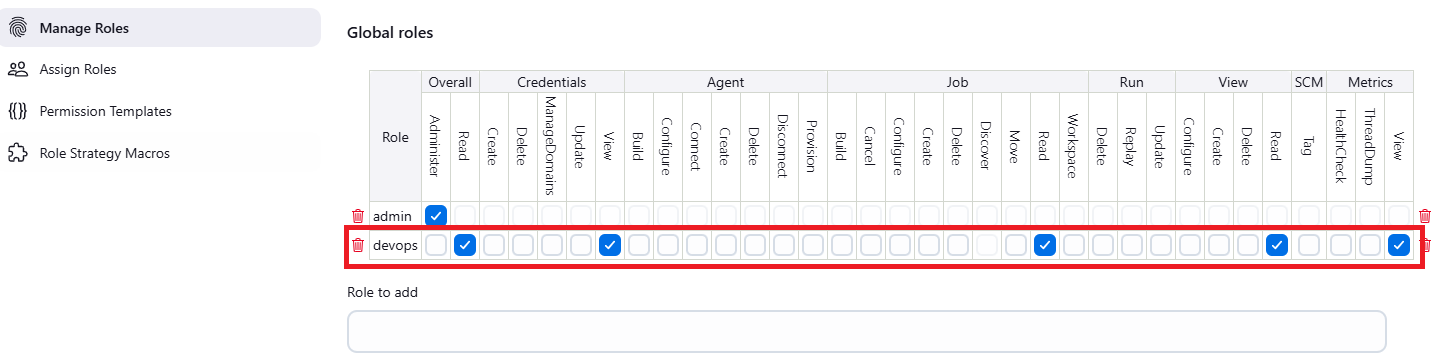
* **file-parameters** – This plugin is used to allow uploading file as parameter, which jenkin inserts into the pipeline as environment variable. The following image shows an environment variable whose name is DATABASE\_CREDENTIAL\_FILE, which is passed into pipeline as environment variable in BASE64 format.

****

* + To convert the file from base64 back to the original format, use the command:

|  |
| --- |
| echo $DATABASE\_CREDENTIAL\_FILE | base64 -d |

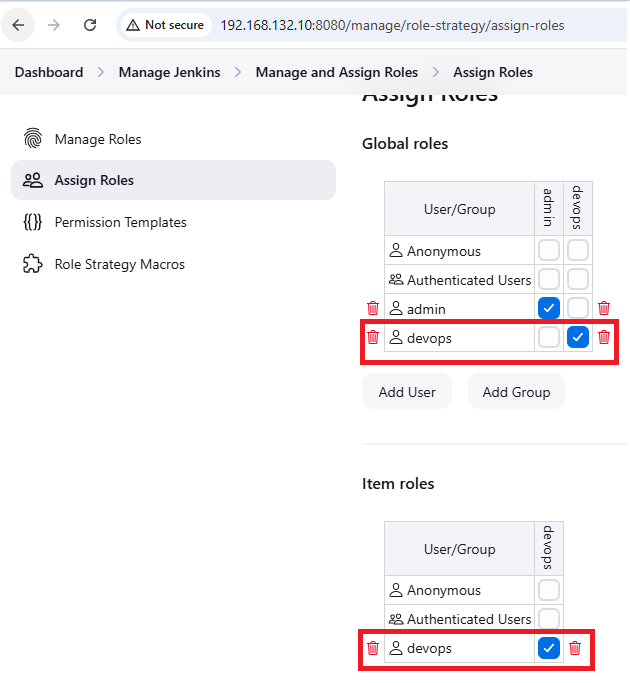
* **Role-based Authorization Strategy** – This plugin is used to manage access control to jenkin with Role-based access control strategy. After installing this plugin, create a new account named **devops**. This account will be used to manage and maintain the CICD pipeline for Wizard's Almanac application.
  + Enable Role-based access control: Manage jenkin > security > Authorization > Role-based strategy
  + Create new devops role and assign permission: Manage Jenkin > Manage and assign role > Manage role:



* + Assign devops role to items: This role has all permissions but to only pipelines whose name starts with “devops-“



* + Assign devops role to user devops:



After all the above-mentioned steps are done, let ‘s set up a slave for the jenkin pipeline. A slave is where jenkin uses to run pipelines on. In this documentation, the slave is an Ubuntu VM:

* SSH to the slave node and install Java runtime environment. Jenkin slave requires Java runtime environment to run task.

|  |
| --- |
| sudo apt update  sudo apt install default-jre |

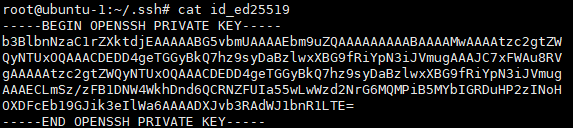
* Create a root working directory for jenkin:

|  |
| --- |
| mkdir –p /home/ubuntu/jenkin |

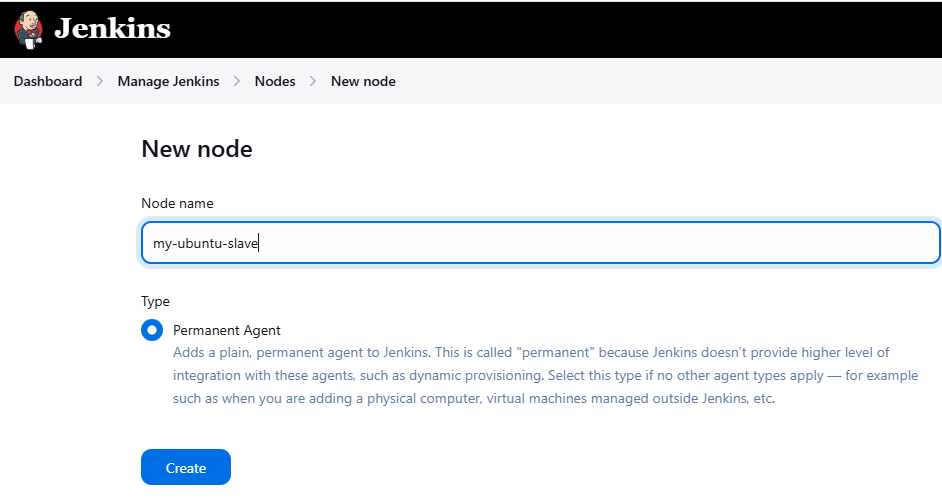
* Setup ssh key for the slave:

|  |
| --- |
| ssh-keygen  ssh-copy-id <slave\_user>@<slave\_ip> |

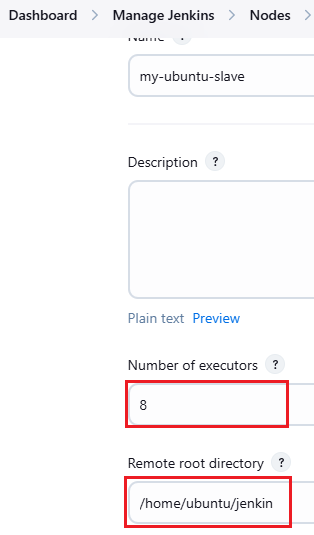
* Get the private key in the directory **~/.ssh/**. For example, the file might look like this:



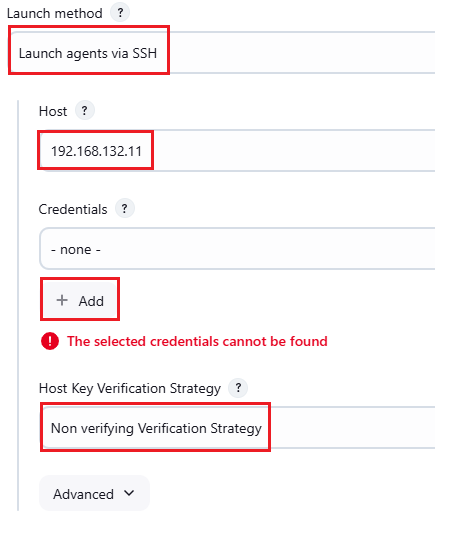
* Go to manage jenkin > node > create new node:



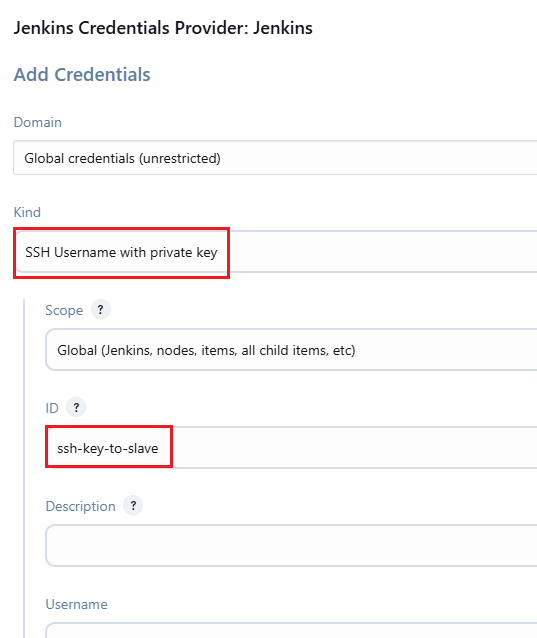
* Click Create button and go to the next page. In the next page, fill in the followings:

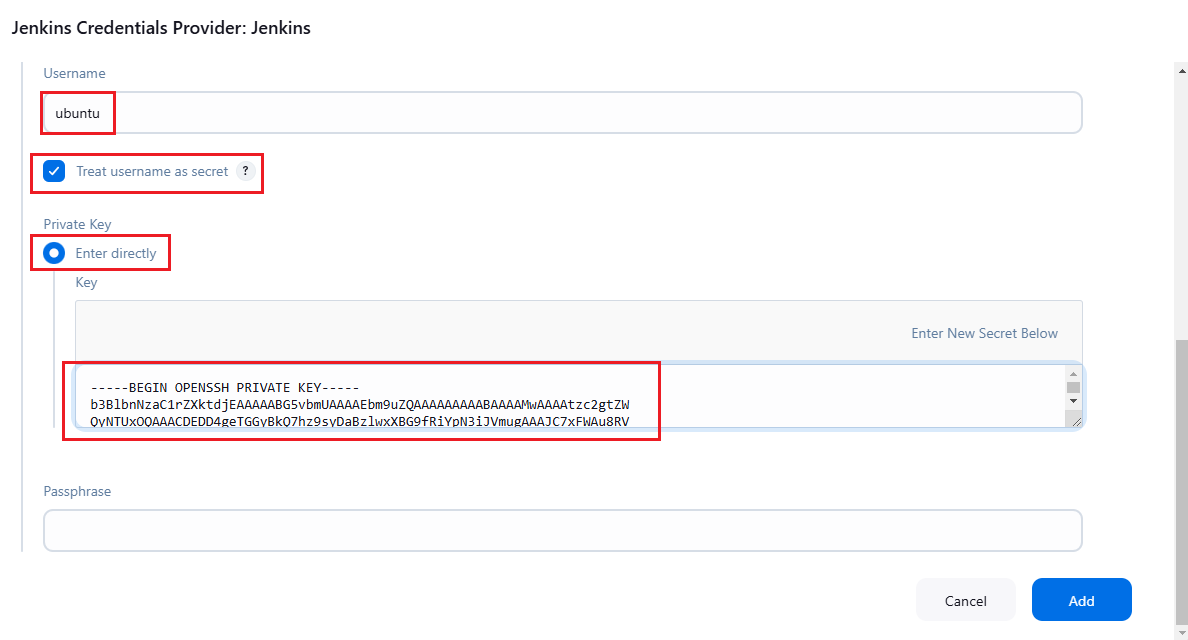


* + Number of executors: This should be equal to total virtual core of the slave
  + Remote root directory: This should be **/home/ubuntu/jenkin,** which was created in the above “Create a root working directory” step
* In Launch method, choose Launch agents via SSH

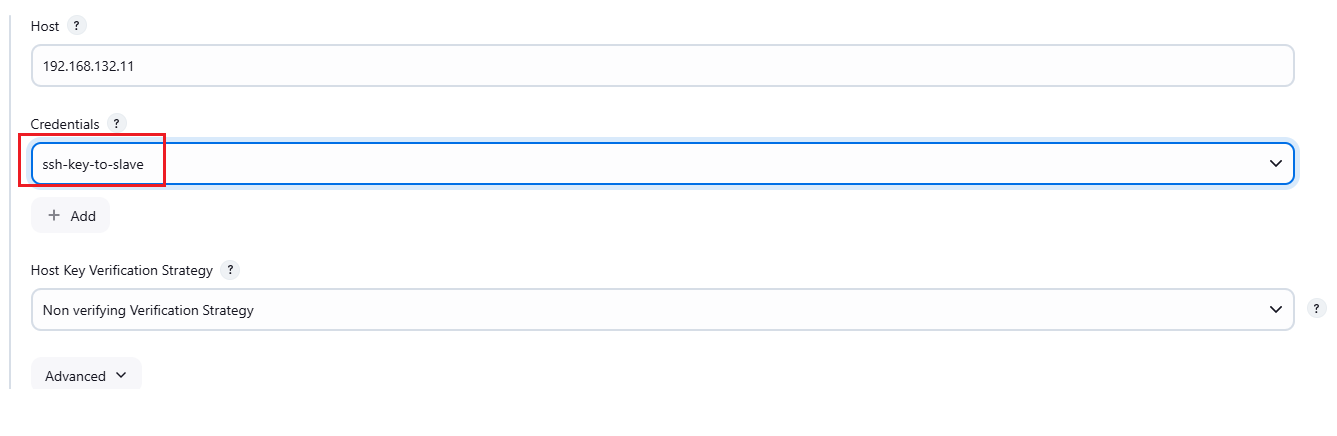


* + Host: IP or DNS name of the slave
  + Host Key Verification Strategy: None verifying Verification Strategy
  + Credentials: click on Add button. In the next page:





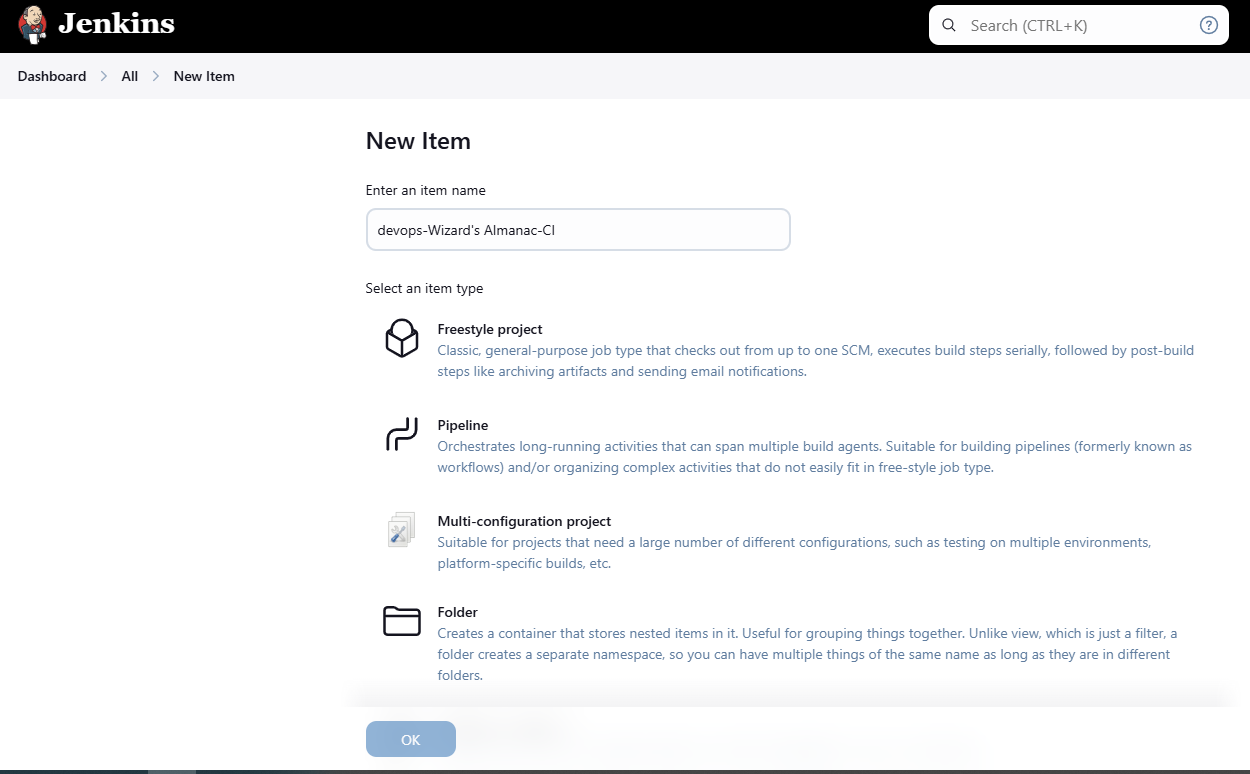
* + Username: Username to ssh to the slave
  + Private Key: The private key which is secured from the above “Get the private key” step.
* Click Add button to add new ssh key credential. Then choose this new credential and click on save button



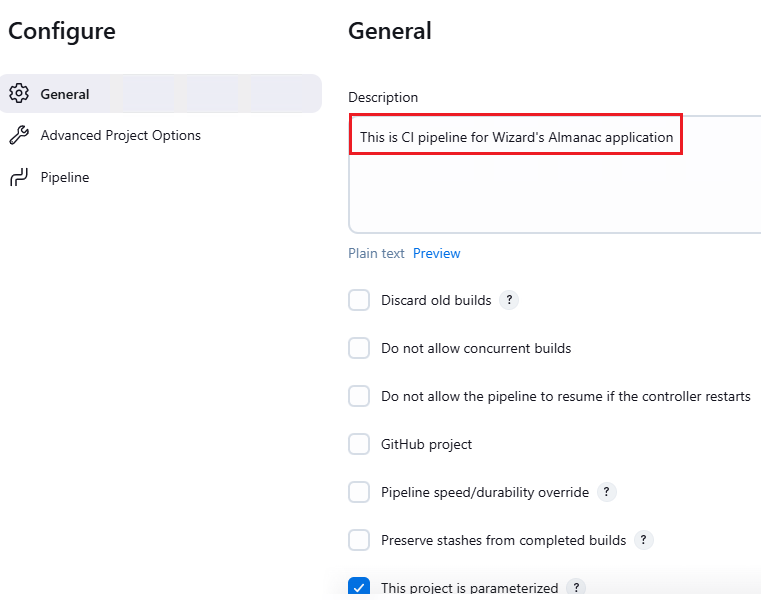
* Now the new agent has been launched successfully and securely via ssh-key.

1. **How to run CI pipeline**

Go to the Jenkin console > create new item > create new pipeline named “**devops-Wizard's Almanac-CI**”

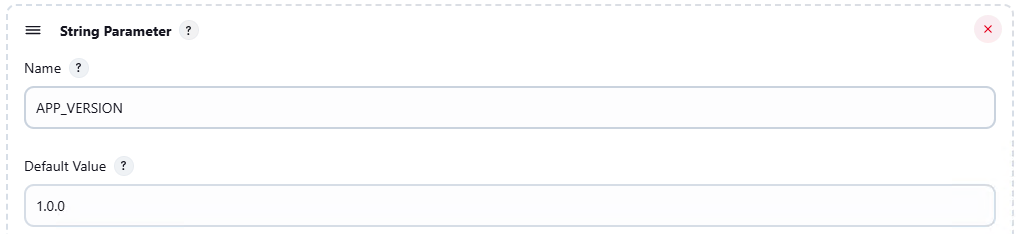


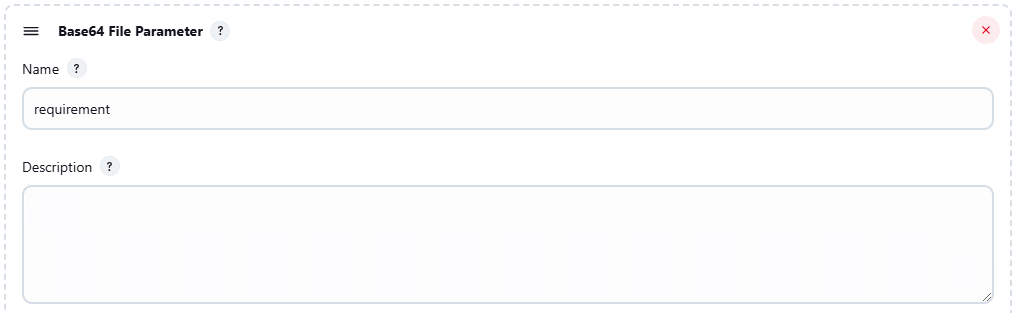
Go to configure pipeline and do the followings:



* + Description: Make a short description about this CI pipeline
  + Tick on the option “**This project is parameterized**” and add the following parameters: (the systax is the following: parameter\_type: parameter\_name : parameter\_default\_value)
    - String: PYTHON\_BASE\_VERSION: 3.11-slim
    - String: APP\_VERSION: 1.0.0
    - String: DOCKER\_USER\_NAME: <no\_default\_value\_for\_this>
    - String: DOCKER\_PASSWORD: <no\_default\_value\_for\_this>
    - String: COMPANY\_CONTAINER\_REGISTRY: <company\_container\_registry>
    - Base64 file parameter: requirement: <no\_default\_value\_for\_this>







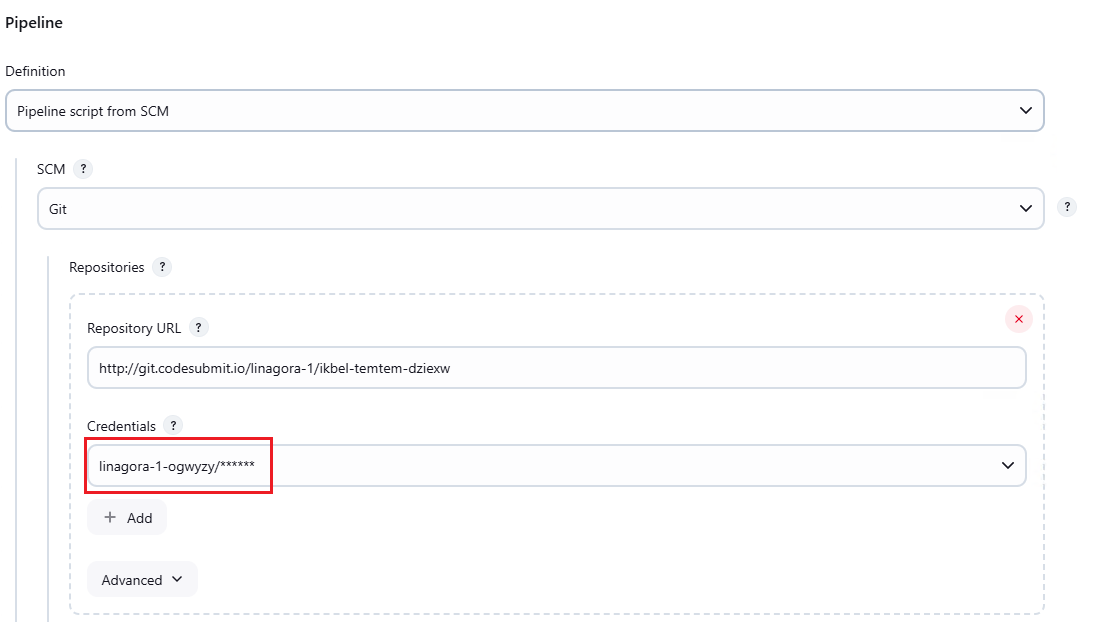
* + Next, choose the Pipeline script from SCM option:



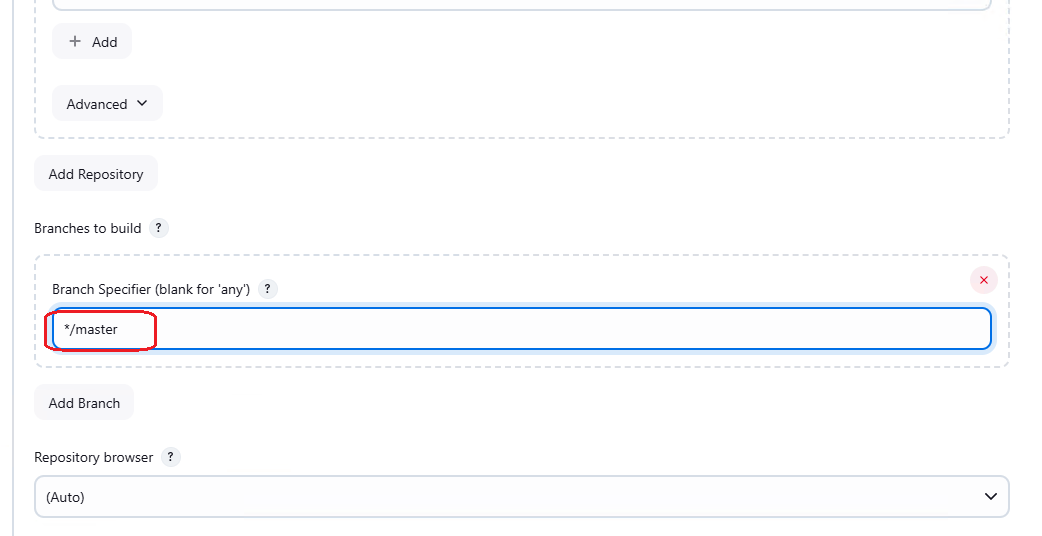
* + Fill in the Repository URL and add new credentials:



* + Kind should be Username with password. Fill in Username and access token into Username and Password box respectively



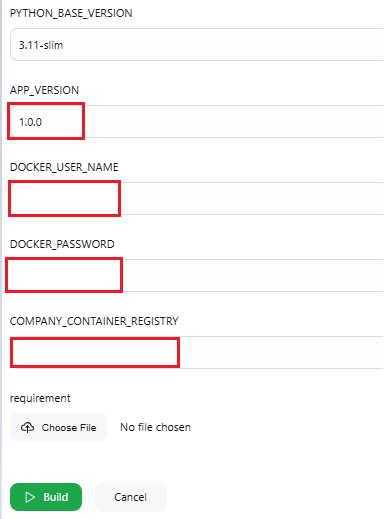
* + Branch specifier: Specify the branch which code is pulled from. In this case, the branch is master



* + After done all the steps, click apply > save

To run the CI pipeline, do the followings:

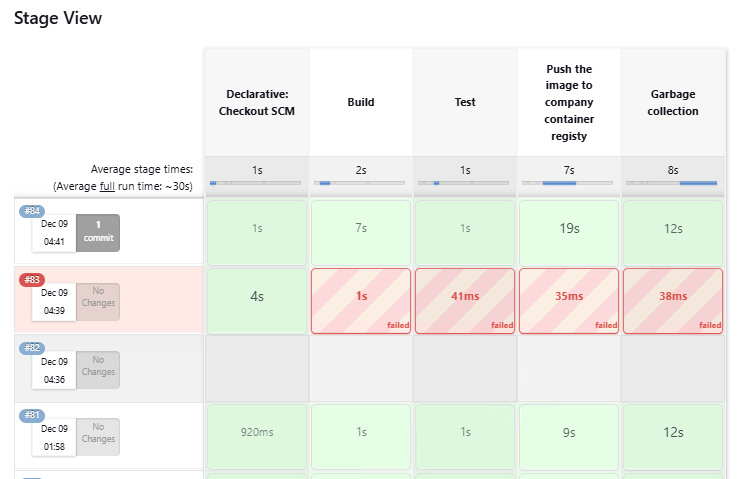
* Dashboard > devops-Wizard's Almanac-CI > Build with parameters



* + APP\_VERSON: The version of the docker image which is built with this CI pipeline
  + DOCKER\_USER\_NAME: Name of the container registry that is used to push the resulted image
  + DOCKER\_PASSWORD: Password/token of the container registry that is used to push the resulted image
  + COMPANY\_CONTAINER\_REGISTRY: The repo name of the container registry which holds the image
  + Requirement: This is a file which specifies all the packages that pip should install during build stage. The file should be like the below. If not set, then the default file (flask==3.1.0) is used



* + After all parameters are set, click on build and wait for result:



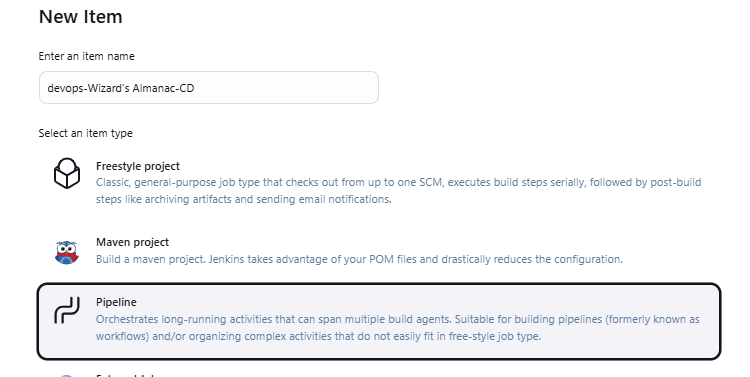
* + As mention above, if all steps succeed (build, test), the built image is pushed to the company‘s container registry. After this step, garbage is removed from the slave.

In summary, the CI pipeline includes:

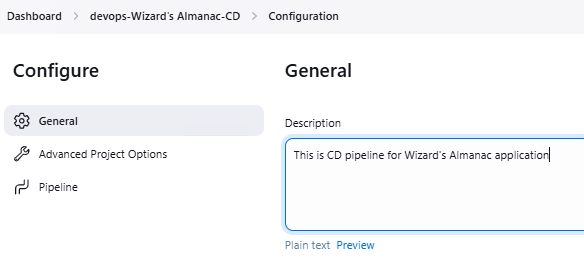
* Checkout SCM: Pull source code from git repo
* Build: Build image from dockerfile instructions
* Test: Run the container and map it to localhost:5000 of the slave. Make a curl request to localhost:5000 and grep the string "Welcome to the Wizard's Almanac". If found, the app is running normally.
* Push the image: push the built image to the container registry
* Garbage collection: Stop and remove the container, also, remove all files which have been pulled from the git repo.

1. **How to run CD pipeline:**

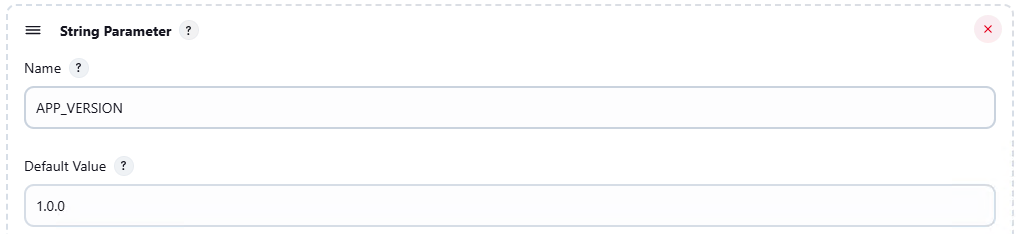
Go to the Jenkin console > create new item > create new pipeline named “**devops-Wizard's Almanac-CD**”

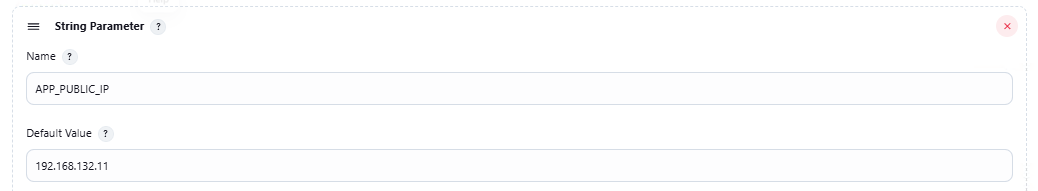


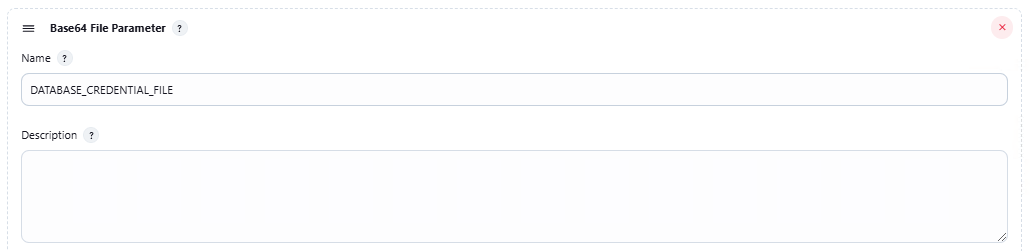
Go to configure pipeline and do the followings:



* + Description: Make a short description about this CD pipeline
  + Tick on the option “**This project is parameterized**” and add the following parameters: (the systax is the following: parameter\_type: parameter\_name : parameter\_default\_value)
    - String: APP\_VERSION: 1.0.0
    - String: DOCKER\_USER\_NAME: <no\_default\_value\_for\_this>
    - String: DOCKER\_PASSWORD: <no\_default\_value\_for\_this>
    - String: COMPANY\_CONTAINER\_REGISTRY: <company\_container\_registry>
    - String: APP\_PUBLIC\_IP: <no\_default\_value\_for\_this> (this is the public IP of the app after deployment)
    - Base64 file parameter: DATABASE\_CREDENTIAL\_FILE: <no\_default\_value\_for\_this> (If this flask app needs to connect to a database, pass this parameter into the pipeline, else, leave it empty)







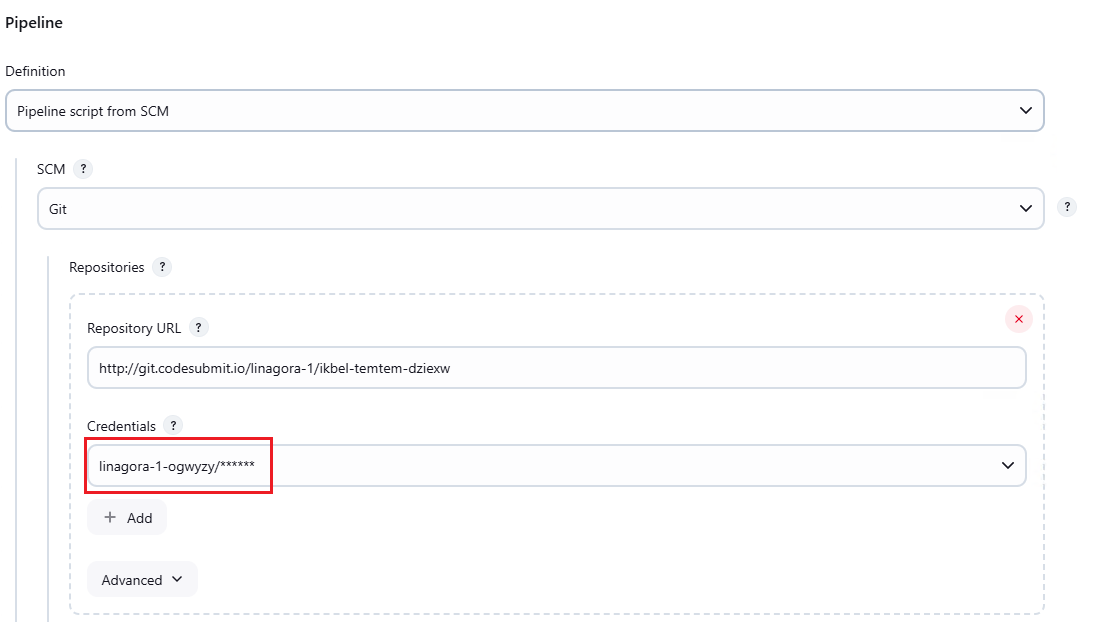
* + Next, choose the Pipeline script from SCM option:



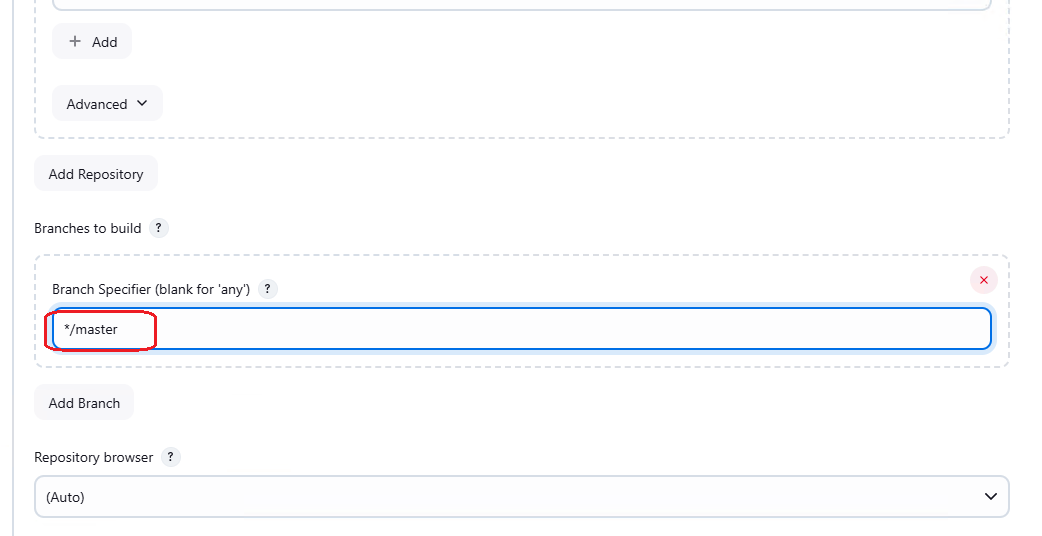
* + Fill in the Repository URL and add new credentials:



* + Kind should be Username with password. Fill in Username and access token into Username and Password box respectively



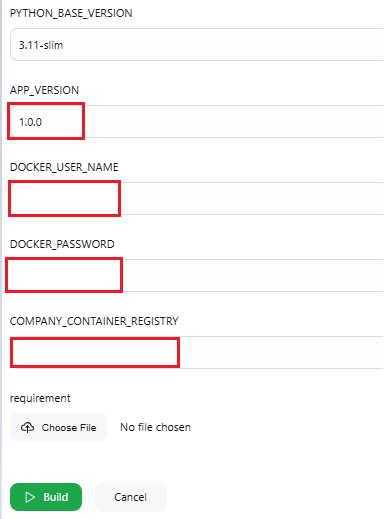
* + Branch specifier: Specify the branch which code is pulled from. In this case, the branch is master



* + After done all the steps, click apply > save

To run the CD pipeline, do the followings:

* Dashboard > devops-Wizard's Almanac-CI > Build with parameters



* + APP\_VERSON: The version of the docker image which is built with this CI pipeline
  + DOCKER\_USER\_NAME: Name of the container registry that is used to push the resulted image
  + DOCKER\_PASSWORD: Password/token of the container registry that is used to push the resulted image
  + COMPANY\_CONTAINER\_REGISTRY: The repo name of the container registry which holds the image
  + Requirement: This is a file which specifies all the packages that pip should install during build stage. The file should be like the below. If not set, then the default file (flask==3.1.0) is used



* + After all parameters are set, click on build and wait for result: