**JENKINS**

If you opt LTS version following are the commands to work on macOS.

install: brew install jenkins-lts

start: brew services start jenkins-lts

restart: brew services restart jenkins-lts

update: brew upgrade jenkins-lts

**Initial Password**

/Users/akshayreddygangula/.jenkins/secrets/initialAdminPassword

5e72aaa7e77047b9aa5cc6a3693cf57b

localhost:8080 is to connect with **jenkins.**

**Jenkins URL:** [**http://localhost:8080/**](http://localhost:8080/)

Username: Akshay

Password: axayreddy

**Freestyle vs Pipeline**

**Freestyle job**

* is the most basic type of job in Jenkins. It provides a straightforward configuration interface, where you can define a series of steps to run.
* **Ideal for simpler projects** or smaller tasks that don’t require complex configurations.
* Limited in flexibility, as it’s mostly **GUI-driven** with predefined options and plugins. Not suitable for complex workflows.
* Lacks the advanced control structures like branching, looping, or parallel executions found in pipelines.
* Configurations are stored in the Jenkins UI, making them **less portable and harder to version control**(can’t track if someone made changes).

**Pipeline Job**

* offer a more advanced and programmable approach, allowing for continuous integration and continuous delivery (CI/CD) pipelines as code. Pipelines are defined using a Jenkinsfile, written in a Groovy-based Domain-Specific Language (DSL).
* Designed for more complex, long-running jobs that require greater flexibility, such as CI/CD workflows, multiple stages, and integration with other tools.
* Highly flexible; supports complex workflows, multi-stage builds, parallel execution, and integration with source control (e.g., Git).
* Enables advanced features like error handling, conditional logic, and pipeline steps that allow complex build and deployment workflows.
* Pipeline as code makes it **easy to version control** and manage, enhancing collaboration and reproducibility.

**Jenkins Architecture**

**1. Jenkins Server (Master):**

* **Primary Role**: The Jenkins server, also known as the **master**, is the core of the Jenkins architecture. It is responsible for managing the entire Jenkins setup, including:
  + Scheduling jobs.
  + Dispatching builds to the appropriate **nodes**.
  + Managing configurations and job history.
  + Providing the web interface to users for managing jobs, plugins, and configurations.
* The master can run jobs directly if it's not heavily loaded, but in larger setups, it's generally best to offload job execution to **nodes**.

**2. Jenkins Nodes (Slaves):**

* **Primary Role**: **Nodes** (also known as **slaves**) are machines that are connected to the Jenkins master and are used to execute Jenkins jobs. They allow Jenkins to scale out and distribute workload, so you can have jobs running on different machines.
* Nodes can be physical machines, virtual machines, or containers.
* Jobs can be assigned to specific nodes based on labels or requirements such as available resources, operating systems, or special configurations.

**Pipeline Script**

1. We can see the logs *select the build-number* then **Console Output.**
2. To use SHELL cmds use **sh** in pipeline script {**sh** for linux/Mac} {**bat** for windows}

**sh** '''

echo "I'm using shell"

whoami

'''

For single cmd we can use **sh “whoami”**

1. We can view the **workspace path** & downloads the outputs in the **Workspace**
2. To edit the script select **Configure**
3. Run cleanWs() for the deletion of the Workspace.
4. We can manually exit the execution. ( X on right)
5. exitcodes for failures are (1…255) & success is (0)
   * To manually exit the script we can use **exit 2** (1-255) using **sh**.
6. **test -f <file\_path>** (or) **[** **-f <file\_path>** **]** both executes the condition(finding the given path) and will gives the output **true** or **false**.
7. We can set the environment in the script but it should be mentioned before **stages{}**. **environment**{

BUILD\_FILE\_NAME = ‘hello-file.txt’

}

* + They can be accessed by $BUILD\_FILE\_NAME

**Post function**

To archive the artifacts at the end of Script. Mention below code aligning with stages{}. Not inside stages{}

post {

**success** { // **always | failure | unstable | aborted | changed**

**archiveArtifacts artifacts: 'build/\*\*'**

}

}

* This post block runs after all stages have completed, regardless of individual stage results.
* It’s typically used for actions that should occur once all stages are done, like sending final notifications, performing cleanup, or archiving logs.’
* The Priority of the execution is **always, aborted, failure, unstable, success, changed.** Though you put success before always => success is executed first.

**Tomcat**

* After installation of the Tomcat, head to **bin** folder and run **./startup.sh** (in Terminal)
* Now check **localhost:8080** (it displays Tomcat page)
* To **stop** Tomcat run **./shutdown.sh** cmd in Terminal which is in **bin** folder
* port 8080 is commonly used port for web servers and applications.

Place the jenkins.war file in the **webapps** folder in Apache.

Now freshly run the **./startup.sh** (Tomcat starts)

And also append /jenkins which becomes localhost:8080/jenkins

**Creating and Assigning Roles**

For small group

* Select ***Matrix-based security*** for granular control

For large group of people

* Install ***Role-based Authorization Strategy*** plugin and select ***Role-based Strategy*** under *Authorization.*
* This now provides a new option called **Manage and Assign Roles** under Security
  + **Global Roles**: Apply permissions across the **entire Jenkins instance**.
  + **Item Roles**: Apply permissions to **specific projects, jobs, or folders** (pattern)
  + **We can create a user with Global Admin role & Reader role in specific project**

**Jenkins Job**

Select **+ new item (or) Create a job** for job creating

* Under ***General***
  + **Description** (supports HTML)
  + **Discards Old builds** (Discards builds based on Days or No. of builds {keeps latest})
  + **GitHub Project** (paste the URL)
  + **This project is parametrized**
  + **Throttle Builds** (No. of builds in time interval (minute, hour, day, week, year)
  + **Execute concurrent builds if necessary** (Allowing multiple builds to run simultaneously)
* Under ***VCS***
  + **Git** or **None**
* Build ***Triggers***
  + **Trigger builds remotely (e.g., from scripts)**
  + **Build after other projects are built**
  + **Build periodically** (format: MINUTE HOUR DOM MONTH DOW)
  + **GitHub hook trigger for GITScm polling**
  + **Poll SCM**
* Build ***Environment***
  + **Delete workspace before build starts**
  + **Use secret text(s) or file(s)**
  + **Add timestamps to the Console Output**
  + **Inspect build log for published build scans**
  + **Terminate a build if it's stuck**
  + **With Ant**
* ***Build Steps***
  + *Add build step*
* ***Post-build Actions***
  + *Add post-build action*

**Simple Job Creation**

Create a Free-style job in Jenkins, create a Java program, then in the **Build Steps**

1. address of the Java file.
2. javac HelloWorld.java
3. java HelloWorld (This is automatically generated)

***Running outside Jenkins***

* We can run the job outside the Jenkins by configuring **Build Triggers**.
* Create a random token like: jclkdshgfo9refjceosfh
* Change the URL: JENKINS\_URL/job/Hello-World/build?token=TOKEN\_NAME
  + Replace JENKINS\_URL and TOKEN\_NAME
  + localhost:8080/job/4.Hello-World/build?token=jclkdshgfo9refjceosfh
  + If we run the above link it will build.

**Dependable Jobs Creation**

Create ***Job1, Job2, Job3*** freestyle Projects

* In Job1 go to **Post-build Actions ->Projects to Build** (Select Job2) -> **(stable)**
* In Job3 go to **Build Trigger** -> Build after other projects are built (Select Job2)

**Jenkins Integration with Git**

Two ways of Integration

1. Poll SCM
2. GitHub Webhooks

***Poll SCM***

* Make sure Git plugin is installed beforehand.
* Create a new repo in GH, and a remote in local repo and then push it to GH repo
* In Jenkins job
  + Under **General** select **GH Project** and paste the URL (both works)
  + Under **Source Code Management** select **Git** and paste URL (for public no credentials required), then select the branch
  + Under **Build Triggers** select **Poll SCM** and give a schedule
* Any changes made in GH repo will be directly reflected in the provided location under ***Workspace*** location; this automatically create a **new build**.
* So, while the changes are primarily updated in the Jenkins workspace, you can add steps in the build configuration to push or deploy the changes elsewhere if needed.

*Status*: [Started by an SCM change](https://bab2-2600-1702-5891-4c0-487e-d777-5d2-888b.ngrok-free.app/job/Hello-World/25/pollingLog)

**Pros and Cons:**

* Safer than Webhooks and easier to configure
* Polling is less efficient in both resources and time perspective.
* Resources are continuously wasted and have to wait.

***GitHub Hooks***

* For this you need to expose Jenkins to public internet

We can run in public using **ngrok**

In terminal

* ngrok http 8080
  + This can be accessed in localhost:4040 or the links shown inside.

Go to GitHub -> Settings -> Webhooks -> Add Webhook

* Payload URL (Public URL of Jenkins)
* Content Type (application/json)
* In **Source Code Management** add URL of GitHub
* In **Build Triggers**, Check **GitHub hook trigger for GITScm polling**

*Status*: [Started by GitHub push by AkshayGangula](https://bab2-2600-1702-5891-4c0-487e-d777-5d2-888b.ngrok-free.app/job/Hello-World/27/pollingLog)

**Pros and Cons:**

* Webhooks give you exact build trigger time without checking constantly
* Security Concerns as Jenkins opened in public
* Configuration is complicated