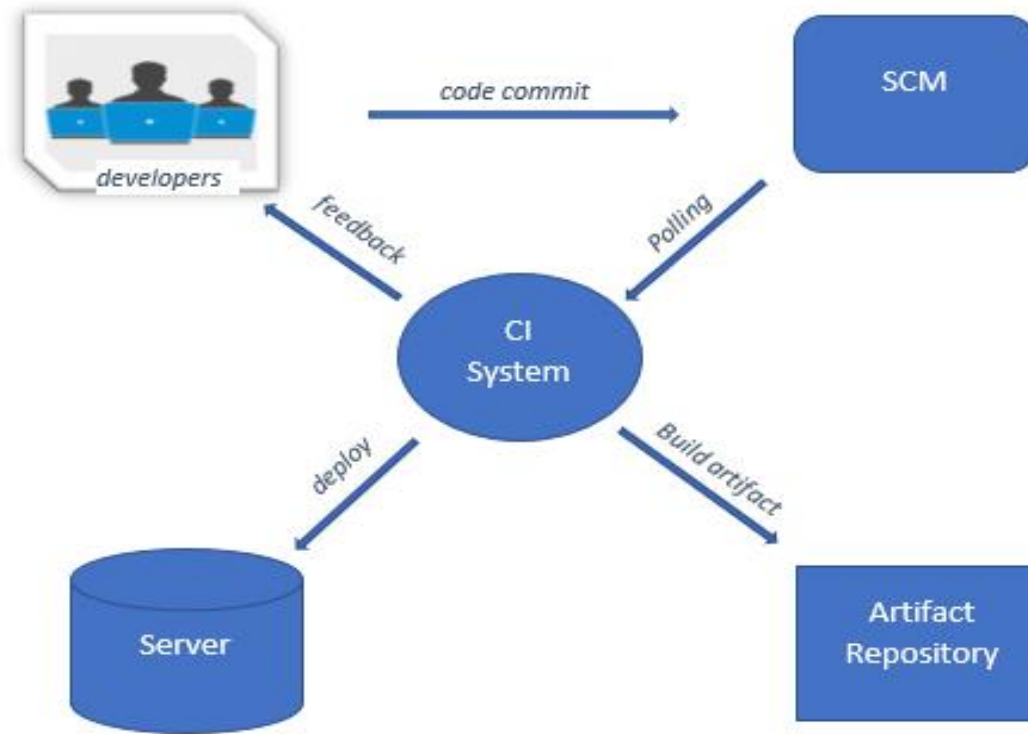


## Module 4- Continuous Integration with Jenkins

# Source Code Polling



# Source Code Polling

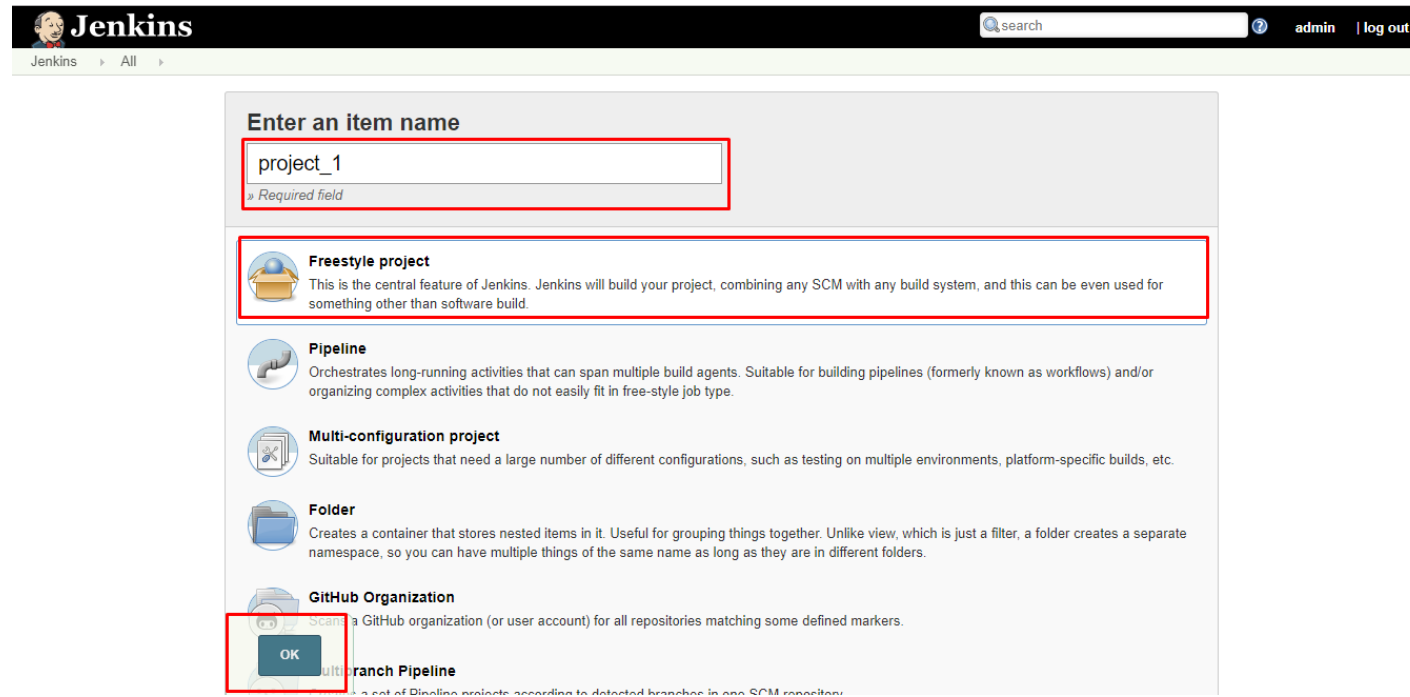
- Developers develop the source code in their local machine and push the code to the central cloud repository (SCM) like GitHub or Bitbucket
- Jenkins will poll the source code from SCM and build the artefacts in the configured environment.
- Once the CI build is completed post build activities will start.
- Post Build activities includes - Deploying to server, storing build artifacts, displaying the test results and sending the feedback to developers are some examples for post build activities.

# Triggering Jenkins Build

- GitHub plugin for Jenkins is the most basic plugin for integrating Jenkins with GitHub projects.
- Using GitHub plugin along with Webhooks enables to schedule build, pull the code and data files from GitHub repository and automatically triggers each build on the Jenkins server after each commit in the GitHub repository.

# Jenkins Builds

- Jenkins project is a repeatable build job which contains steps and post-build actions.
- The freestyle build job is a highly flexible and easy-to-use option.
- Freestyle job is easy to set up and configure



The screenshot shows the Jenkins 'New Item' page. At the top, there is a header with the Jenkins logo, a search bar, and links for 'admin' and 'log out'. Below the header, there is a breadcrumb trail: 'Jenkins > All >'. The main content area is titled 'Enter an item name' and contains a text input field with the value 'project\_1'. Below the input field, there is a red box highlighting the 'Freestyle project' option. The 'Freestyle project' option is described as 'This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.' Below this, there are other options: 'Pipeline', 'Multi-configuration project', 'Folder', and 'GitHub Organization'. At the bottom left, there is a red box highlighting the 'OK' button.

Jenkins

search

admin | log out

Jenkins > All >

Enter an item name

project\_1

Required field

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

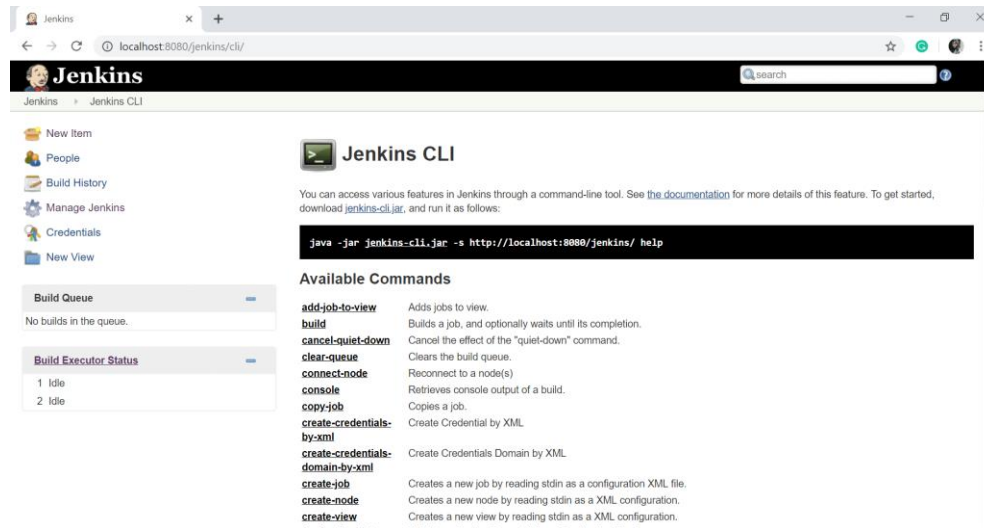
**GitHub Organization**  
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

**OK**

**Branch Pipeline**  
A set of Pipeline projects according to detected branches in one SCM repository.

# Jenkins CLI

- Jenkins has a built-in command line interface that allows you to access Jenkins from a script or from your shell. This is convenient for automation of routine tasks, bulk updates, trouble diagnosis, and so on.
- This interface is accessed via the Jenkins CLI client, which is a Java JAR file distributed with Jenkins.

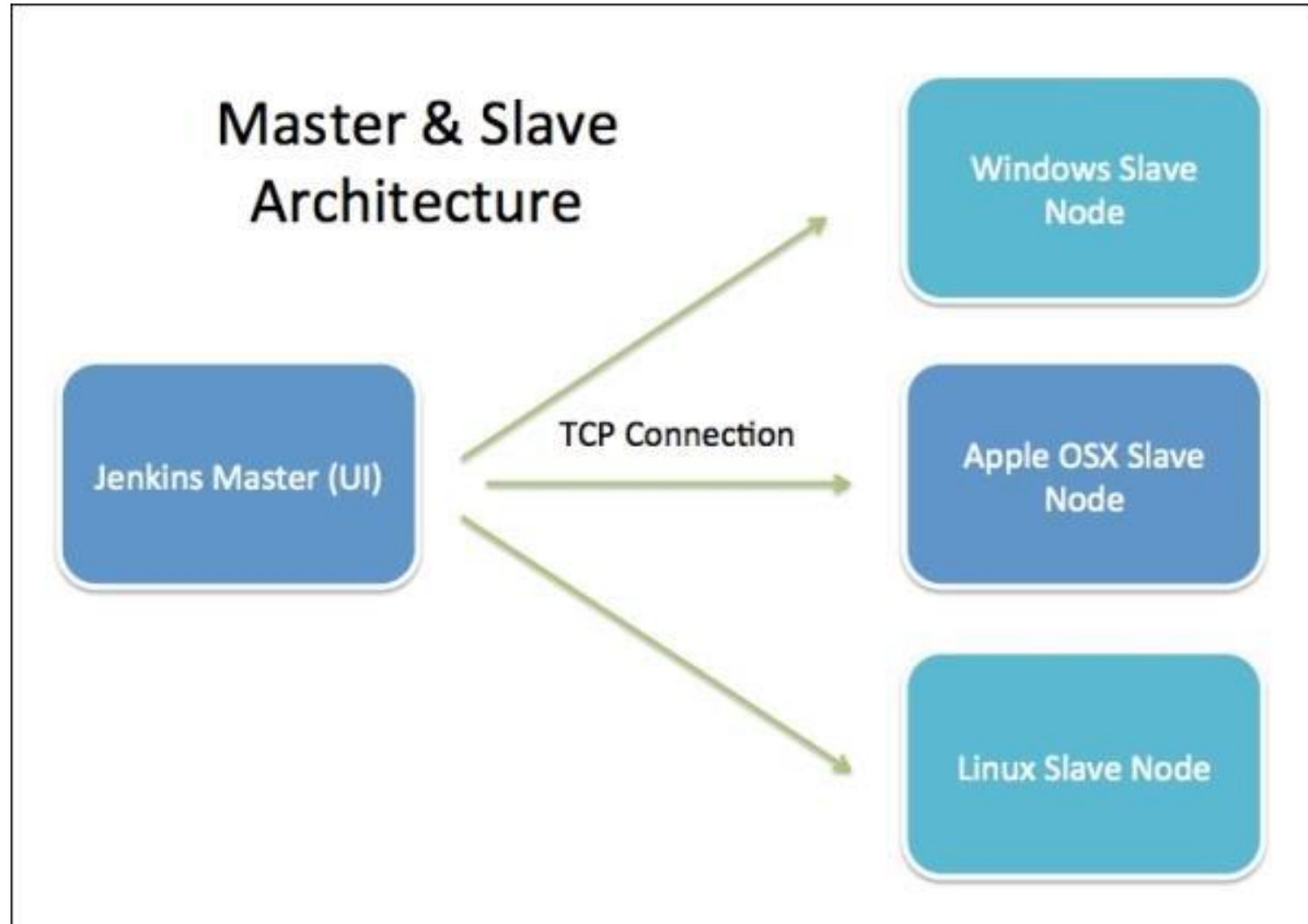


## Running a CLI command

- The general syntax is as follows (the design is similar to tools like svn/git):
- `java -jar jenkins-cli.jar [-s JENKINS_URL] command [options...] [arguments...]`

Ref: <https://wiki.jenkins.io/display/JENKINS/Jenkins+CLI>

# Jenkins Architecture



# Jenkins Master

- Main Jenkins server is the Master.

The Master's job is to handle:

- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.





# Jenkins Slave Node

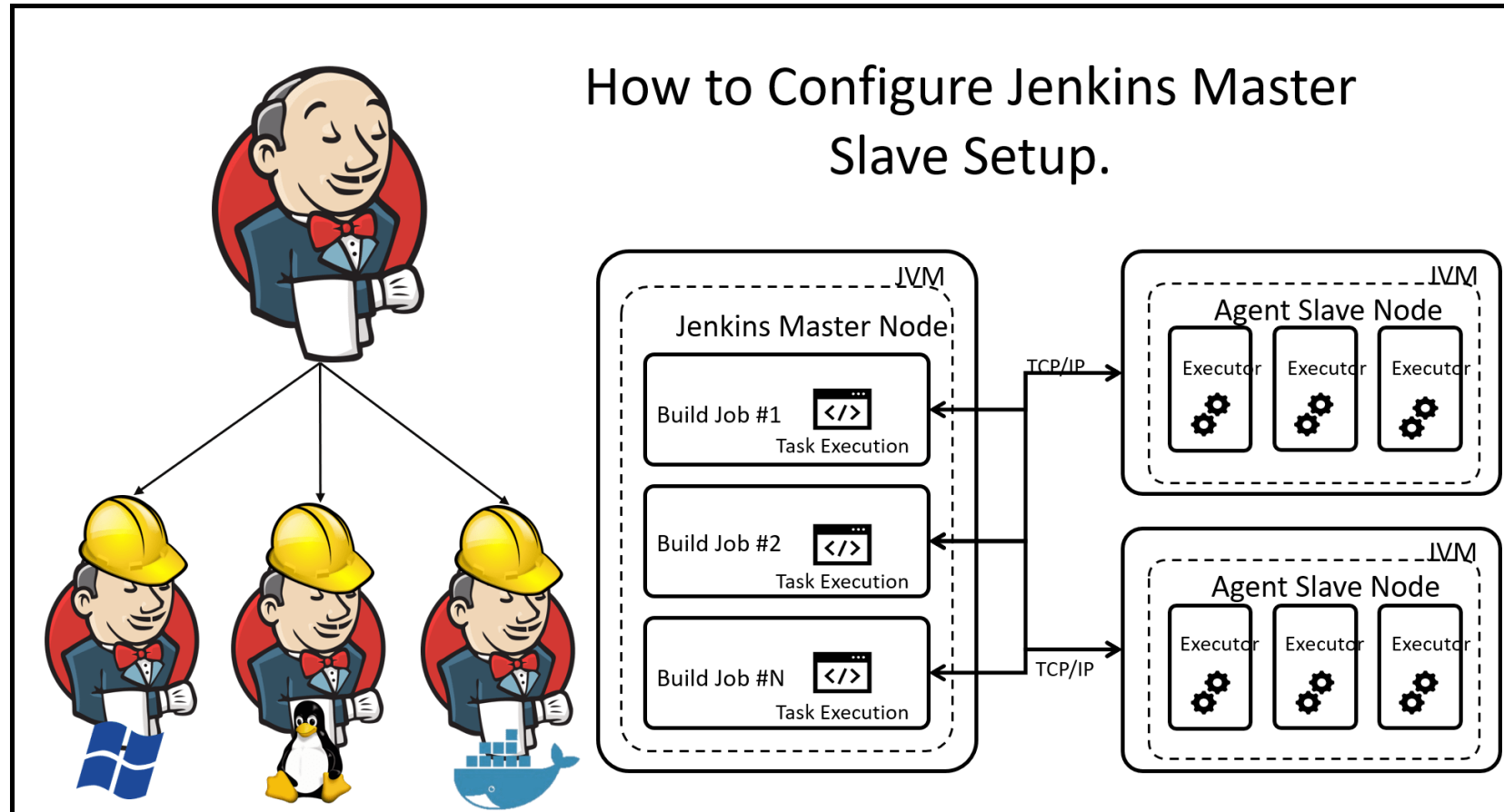
A Slave is a Java executable that runs on a remote machine.

**Following are the characteristics of Jenkins Slaves:**

- It hears requests from the Jenkins Master instance.
- Slaves can run on a variety of operating systems.
- The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- Projects can be configured to always run on a particular Slave machine, or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.



# Configure Jenkins Master Slave Setup

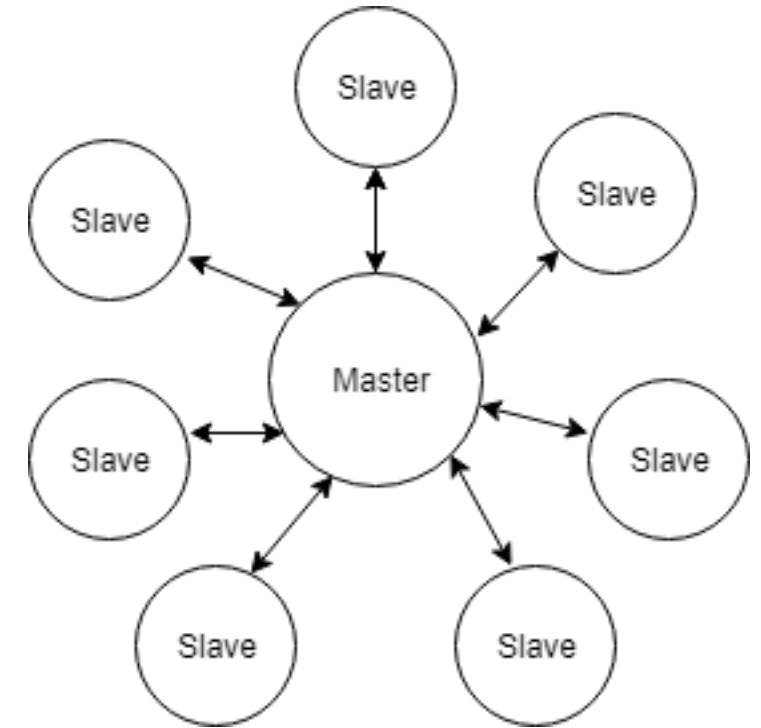


# Jenkins Distributed Build

- Master can delegate the workload of building projects to “slave” nodes. The projects built in this manner is referred to as “Distributed Builds”.
- Once the Master / Slave is setup , this process is automatic
- Stickiness to a particular machine can also be specified .
- Slave agent and Jenkins master needs to establish a bi-directional communication ( TCP/IP) in order to operate.

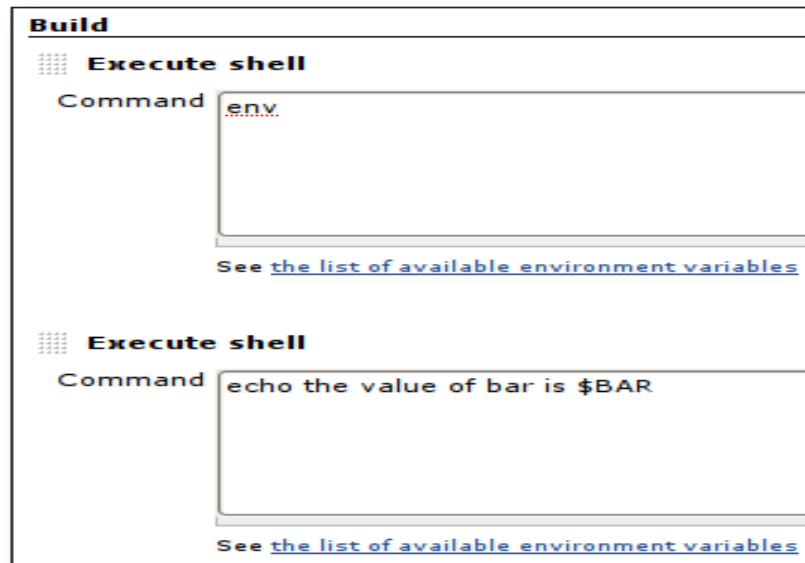
## Use Cases :

- Meet the requirement of different build environment for different projects.
- Building different environments to test builds.



# Advanced Build – Parameterized Build Jobs & Triggers

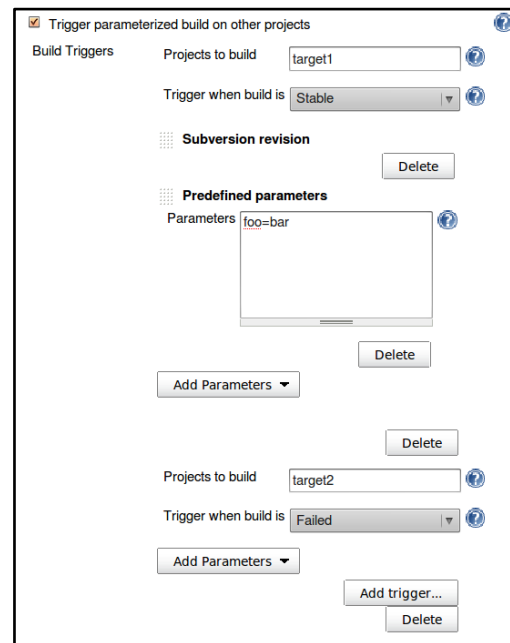
- You set up a test job on Jenkins, and it accepts a distribution bundle as a parameter and perform tests against it. You want to have developers do local builds and let them submit builds for test execution on Jenkins. In such a case, your parameter is a zip file that contains a distribution.
- Your test suite takes so much time to run that in normal execution you can't afford to run the entire test cycle. So you want to control the portion of the test to be executed. In such a case, your parameter is perhaps a string token that indicates that test suite to be run.
- The parameters are available as environment variables. So e.g. a shell (\$FOO, %FOO%)



The image shows a screenshot of the Jenkins 'Build' configuration page. It features two 'Execute shell' steps. The first step has a 'Command' field containing the text 'env'. Below this field is a link that says 'See the list of available environment variables'. The second step also has a 'Command' field, which contains the text 'echo the value of bar is \$BAR'. Below this field is another link that says 'See the list of available environment variables'.

# Parameterized Trigger Plugin

- This plugin lets you trigger new builds when your build has completed, with various ways of specifying parameters for the new build.
- You can add multiple configurations: each has a list of projects to trigger, a condition for when to trigger them (based on the result of the current build), and a parameters section.



The screenshot shows the 'Trigger parameterized build on other projects' configuration page. It features a 'Build Triggers' section with a checked checkbox. Below this, there are two configuration blocks. The first block has 'Projects to build' set to 'target1' and 'Trigger when build is' set to 'Stable'. It includes a 'Subversion revision' section with a 'Delete' button and a 'Predefined parameters' section with a text area containing 'foo=bar' and a 'Delete' button. The second block has 'Projects to build' set to 'target2' and 'Trigger when build is' set to 'Failed'. It also has an 'Add Parameters' button and a 'Delete' button. At the bottom, there are 'Add trigger...' and 'Delete' buttons.

Ref: <https://wiki.jenkins.io/pages/viewpage.action?pageId=36602920>

# Running Script Commands

## Steps to execute shell script in Jenkins:

- In the main page of Jenkins select New Item.
- Enter an item name like "my shell script job" and chose Freestyle project. Press OK.
- On the configuration page, in the Build block click in the Add build step dropdown and select Execute shell.
- In the textarea paste the script as given below or inform how to execute

```
#!/bin/bash
echo "hello, today is $(date)" > /tmp/jenkinstest
or just
/path/to/your/script.sh
```
- Click Save.
- Now the newly created job would appear in the main page of Jenkins
- Open it and select Build now to see if it works.
- Once it has finished pick that specific build from the build history and read the Console output to see if everything happened as desired.

Ref: [https://github.com/software saved/build\\_and\\_test\\_examples/blob/master/jenkins/Shell.md](https://github.com/software saved/build_and_test_examples/blob/master/jenkins/Shell.md)

# Scheduling Jobs

- CRON job in Jenkins Job build gives 2 options to build your job periodically. Execute job every time when something new is pushed into the repository since the last build.
- Trigger the job at some fixed or periodic time intervals of your choices, regardless of the fact that if something is changed or not in your repository.

E.g.:

- By setting the schedule period to 20 13 \* \* \* you tell Jenkins to schedule the build every day of every month of every year at the 15th minute of the 13th hour of the day.
- Jenkins used a cron expression, and the different fields are:
- MINUTES: Minutes in one hour (0-59)
- HOURS: Hours in one day (0-23)
- DAYMONTH: Day in a month (1-31)
- MONTH: Month in a year (1-12)
- DAYWEEK: Day of the week (0-7) where 0 and 7 are Sunday

# Support for Build Tools (Grade, Maven, ANT)

- Ant plugin adds Apache Ant support to Jenkins. This functionality used to be a part of the core, but as of Jenkins 1.431, it was split off into separate plugins.
- Maven Integration plugin is a plugin that helps us to build projects that use Apache Maven in Jenkins. In this tutorial, I will guide you how to install this plugin to be able to use it.
- Gradle plugin makes it possible to invoke a Gradle build script as the main build step. It also allows detecting Build Scans in arbitrary console logs, for Maven and Gradle builds and display them in the Jenkins UI.



THANKS