UNIT-4.

Build Systems:

* There are many build systems that have envolved over the history of Software development.
* Sometimes, it might feel as there are more build Systems than programming languages.
* Here is a brief list of build systems;-
  - For Java- Maven, Gradle, Ant
  - For C and C++ - Different flavours of make like CMake Pre Make, GNUMake.
  - For JavaScript- Grunt
  - For Scala - sbt
  - For Ruby - Rake
  - For clojure - a language on JVM [leiningen and Boot]
* Finally, ofCourse we have shellScripts of all kinds.
* Depending on size of your Organization and the type of product you're building, you might Consider any number of these tools
* Normally, Organization Standardize on a single ecosystem, Such as Java and Maven or Ruby and Rake.

* We cannot assume that we will encounter only one build system within our organizations code-base nor only one programming language.

* We should standardize the version control system (VCS) and have a single interface to start builds locally.

* If you have more than one build system to support, you need to wrap one build system in another

Jenkins Build Server:

* A Build Server is a system that builds software based on various triggers.

* These are several to choose from, we will have a look at Jenkins, which is a popular build server written as Java.

* Jenkins is a fork of the Hudson (Oracle) build server

* Jenkins has a special support for building Java code but no way limited to just building Java.

* Setting up a basic Jenkins server is not practically hard.

* Jenkins is handled as a service.

→ `systemctl start jenkins` - Linux Command.

* The fundamental entity in Jenkins is the Job definition, and there are several type to choose from.

* To create a simple Job, follow the below steps:

1. Create a job of the type Freestyle Project
2. Add shell Build step.
3. In the shell entry type and quote.

* Whenever you run the job, the quote will be printed in the Job Log.

* Jobs can be started manually, and you will find a history of job executions and can examine each job Log.

Managing Build Dependencies:

* Some build systems, such as Maven tool are nice in the way Maven POM file (Project Object Model) contains descriptions of which build dependencies are needed, and they are fetched automatically

* Grunt works in a similar way for JavaScript builds.

* There is a build description file that contains the dependencies required for the build.

* Golang builds can even contain links to git hub repositories required for completing the build.

* C and C++ builds present challenges in a different way. AutoConf(package) which adopts itself to the dependencies that are available on that host rather than which dependencies they need.

* If optional dependency is missing. you can still create a build, but the optional feature will not be available in the final executable.

* We would like our builds to behave in an enterprise setting.

* We certainly don't want bad surprises in the form of missing functionality on our production server

Jenkins plugins:

* Jenkins has aplugin system to add functionality to the build server

* There are many different plugins available and they can be installed from within the Jenkins web interface.

* Many of them can be installed without even restarting the Jenkins.

* Among others, we need the git plugin to poll our source code repositories.

The Host Server:

* The Build server is usually a pretty important machine for the organization.

* Building Software is a processor as well as memory and dise intensive.

* Builds should not take too long So you will need a server with good specifications with lots of disc space, processor cores, and RAM.

* Machines are cheaper than people, So don't let this particular machine be the area you save the money on.

Build slaves:

* TO reduce Build Queues, you can add Build Slaves.

* The master server will send Builds to the slaves based ~~to Specific~~ on a Round Robin scheme (or) Tie specific builds to specific Build slaves.

* Build slaves can be used to increase the efficiency of parallel builds

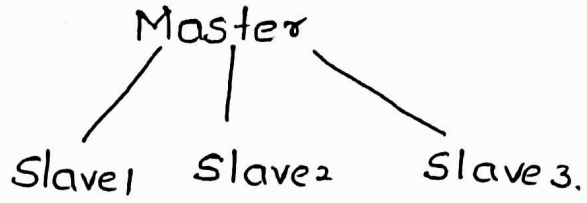


Fig: Master. - Slave Architecture

* They can also used to build Software on different Operating systems.

* For example, you can have a Linux Jenkins master Server and window slaves for Components that use window Build too.

* There are several methods to add Build slaves. to a Jenkins master.

https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds.

* There must be away for the Jenkins master to issue Commands to the Build slave. This Command can be the classic SSH method and Jenkins has a built-in. SSH facility.

Software on the Host:

* Depending on the complexity of your builds, you might need to install many different type of build tools on your build server.

* Jenkins is mostly used to trigger builds.

* It is most convinient to have a linux based host operating system.

* Most of the build system are available in the distribution repositories, so it is very convinient to install them from these.

* TO keep you build server up-to-delete, you can use the same deployment servers that you use to keep your application server. up-to-date

Triggers:

* You can either use a timer to trigger builds, or you can poll the code repository for changes and build if there were changes.

* It can be useful to use both methods at the same time.

• Git repository polling can be used most of the time so that every checkin triggers a build.

* • Nightly builds can be triggered, since these happen at night when no body is suppose to work.

* You can also let the sucessful build of one job trigger another job.

Job changing and Build pipelines:

* It is often useful to be able to chain jobs together

* This works by triggering a second job after the first job finishes sucessfully.

* In the Jenkins terminology, the first build in a chain is called upstream build and the second one is called the downstream build.

* Build chain is often called as a pipeline or workflow.

* These are many plugins that create improved pipelines for Jenkins, multijob plugin and workflow plugin

* The workflow plugin is more advanced, it is promoted by cloudBees Company, who is principle Contributor for Jenkins today.

* Workflow plugin is more advanced it is promoted

* Workflow plugin uses Groovy Build Script

Build Servers and Infrastructure as Code:

* Mismatch that often occurs between GUI based tools such as Jenkins and the DevOps axioms that infrastructure should be described as Code.

* Jenkins job description are text file based.
* It is easy to create ad-hoc solution on top of existing builds with Jenkins.

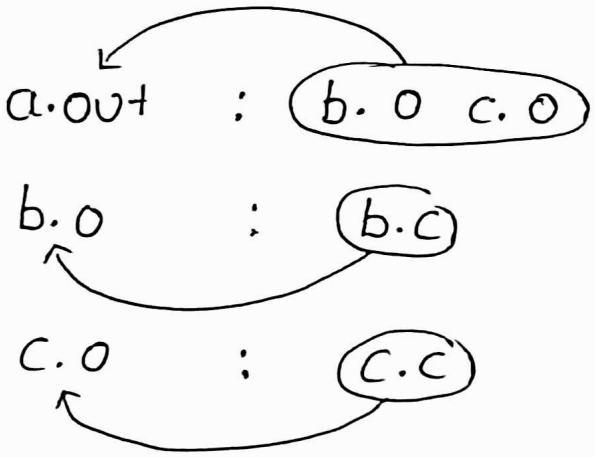
* On the other hand, Jenkins lacks many features, basic features like inheritance and function definitions takes Some effort to provide.

* The build server features in GitLab takes a different approach. Build description are just right from the start

Building by Dependency Order:

* Many build tools have the Concept of build tree where dependencies are built in the order required for the build to Complete.

* In make like tools ~~in~~ this is described explicitly as shown below:



* So, in order to build a.out, both b.o and c.o must be build first.

* In tools such as 'Maven', the build graph is derived from the dependencies we set ~~of~~ for an artifact

* "Gradle", another Java build tool, also create a build graph before building.

* Jenkins has support for visualising the build Order for Maven builds which is called as Reactor

* However, this view is not available for Make style builds.

Build phases:

* One of the principle benfits of Maven Build tool is that it standardizes builds.

* This is very useful for a large organization, Since it won't need to invent its own build standards.

* Other build tools are usually not having clear ways how to implement various build phases

* You can implement the following build phases. with any tool

- Building
- Testing
- Deployment

Alternative Build Servers:

* While Jenkins appears to be pretty dominant in the build Servers.

* "Travis CI" is popular open source build server

* "Buildbot" is a build server that is written in, and Configurable with python.

* The "Goserver" is another one from thoughtworks.
* Bamboo is an offering from Atlassian.
* GitLab also supports build server functionality now.
* Do shop around before deciding on which build server works best for you.

Collating Quality Measures;

* A useful thing that a build Server can do is that Collating of software quality metrics.
* Jenkins has some support for this.
* Java unit test are executed and can be visualized directly on the job page.
* Another more advanced option is using the Sonar Code quality visualizer.
* Sonar tests are run during the build phase and propagated to the Sonar Server, where they are Stored and visualized.
* The drawback of implementing a Sonar server is that it Sometimes slows down the builds.
* The recommedation is to perform Sonar builds in your nightly builds.