**Distributed Builds**

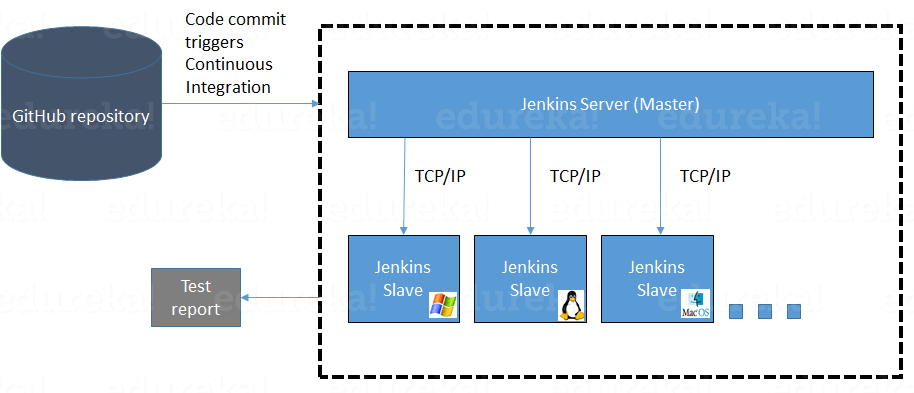
A stand alone Jenkins instance consumes the disk spaces , cpu gets over loaded fast, sometimes you might need several different env to test your build. To perfrom loadbalancing we can scale Jenkins by implementing Master Node Architecture.

Distributed builds are generally used either to absorb extra load, for example absorbing spikes in build activity by dynamically adding extra machines as required, or to run specialized build jobs in specific operating systems or environments. For example, you may need to run particular build jobs on a particular machine or operating system. For example, if you need to run web tests using Internet Explorer, you will need to be use a Windows machine. Or one of your build jobs may be particularly resource-heavy, and need to be run on its own dedicated machine so as not to penalize your other build jobs.

Demand for build servers can also fluctuate over time. If you are working with product release cycles, you may need to run a much higher number of builds jobs towards the end of the cycle, for example, when more comprehensive functional and regression test suites may be more frequent.

**Distributed Build Architecture**

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.



**Jenkins Master**

The Jenkins master serves HTTP requests and stores all important information related to the builds.

Jenkins master manages these nodes to do the actual build work.

Different nodes can run different operating systems and different tools, enabling one Pipeline to build and test the same software for a variety of platforms.

* SCM Polling
* Job Scheduling
* LDAP Authentication
* Build output Reporting and Notifications
* Job History and Build Logs
* Executing Jobs/tasks tied to the Master

**Jenkins Node**

Agents that run on nodes manage the task execution on behalf of the Jenkins master and supply most of the computing power required to build and test the software.

On a distributed Jenkins instance, nodes are separate physical or virtual machines that are configured**.**

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.
* You can configure a project to always run on a particular Slave machine or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

**Benefits of distributed builds**

All production Jenkins instances should run distributed builds. This greatly increases the scalability and security of your instance and allows you to define and use specialized nodes as you build and test your software.

**Scalability**

Running builds on agents makes the Jenkins instance more scalable. If you do not have enough resources to run your builds, just add more agents. Otherwise, you would need to create an entirely new master to support more builds. One master can orchestrate 100s to 1,000s of builds a day by distributing the build load for better resource utilization.

**Security**

Jobs that run on a master have full permissions to all Jenkins resources on the master node. This means that a malicious user could access private information from other builds or Jenkins itself. Running jobs on separate and isolated machines mitigates this risk.

For even greater security, use ephemeral cloud agents that are created for each build and then destroyed when that build is over. This way, if a malicious agent taints the node/executor, no other builds are affected.

**Specialized nodes**

Specialized nodes/agents are used for different build and test stages. This allows you to build and test on different operating systems and different CPU architectures while using different versions of JDK and other tools. Each node is configured with only the tools needed for the builds and/or tests to be run on it. For example, you might configure JDK 8 and Apache Maven on one Linux node or configure JDK 7 and Gradle on a Windows node. Tools are installed directly on the node.

You can also define Docker or Kubernetes containers that contain the required tools and are installed on the node. Pipelines then access this node by using a label that is configured for the node.

**Distributed Jenkins components**

A distributed Jenkins system uses the following components:

* The **Jenkins master** is the Jenkins service itself, which is a webserver that also acts as a "brain" for deciding how, when and where to run tasks.
* A **node** is a server where Jenkins runs build jobs on **executors**. Note that the Jenkins master also runs on a **node**.
* An **executor** is effectively a thread for execution of tasks.
* The **agent** is the tool that manages the **executors** on a remote **node**, on behalf of Jenkins.

### **Jenkins master**

The **Jenkins master** is the Jenkins service itself and is where Jenkins is installed. It is a webserver that also acts as a "brain" for deciding how, when, and where to run tasks. Management tasks (configuration, authorization, and authentication) are executed on the master, which serves HTTP requests. Files written when a Pipeline executes are written to the file system on the master, unless they are offloaded to an artifact repository such as Nexus or Artifactory.

### **Nodes**

Nodes are the "machines" on which build agents run. Jenkins monitors each attached node for disk space, free temp space, free swap, clock time/sync and response time. A node is taken offline if any of these values go outside the configured threshold.

### **Executors**

An executor is a slot for execution of tasks; effectively, it is a thread in the agent. The number of executors on a node defines the number of concurrent tasks that can be executed on that node at one time. In other words, the number of concurrent Pipeline stages that can execute on that node at one time.

### **Agents**

Agents manage the task execution on behalf of the Jenkins master using Executors. Agents are actually small (170KB single jar) Java client processes that connect to a Jenkins master and are assumed to be unreliable. An agent can use any operating system that supports Java. Tools required for builds and tests are installed on the node where the agent runs. They can be installed directly or in a container (Docker or Kubernetes). Each agent is effectively a process with its own PID (Process Identifier) on the host machine.

### **How many executors?**

The proper number of executors per build node must be determined based on the resources available on the node and the resources required for the workload. When determining how many executors to run on a node, consider CPU and memory requirements as well as the amount of I/O and network activity:

* One executor per node is the safest configuration.
* One executor per CPU core may work well if the tasks being run are small.
* Monitor I/O performance CPU load, memory usage, and I/O throughput carefully when running multiple executors on a node.

### **Executors on the master**

On production systems, the node where the Jenkins master runs should be configured with 0 executors to ensure that no builds run on the master.

The reason you should never run builds on the master in production environments is because a build that runs on the Jenkins master can access the same Jenkins home directory with the full permissions of the jenkins user, so the master can steal secrets, modify data, etc., from other processes.

You can run builds on the master for demonstration purposes.

Some administrative tasks, such as backing up the master, do need to run on the master. In this case, you can temporarily define an executor on the master and delete it after the administrative task is complete or take other steps to ensure that the executor is not used for a build job.