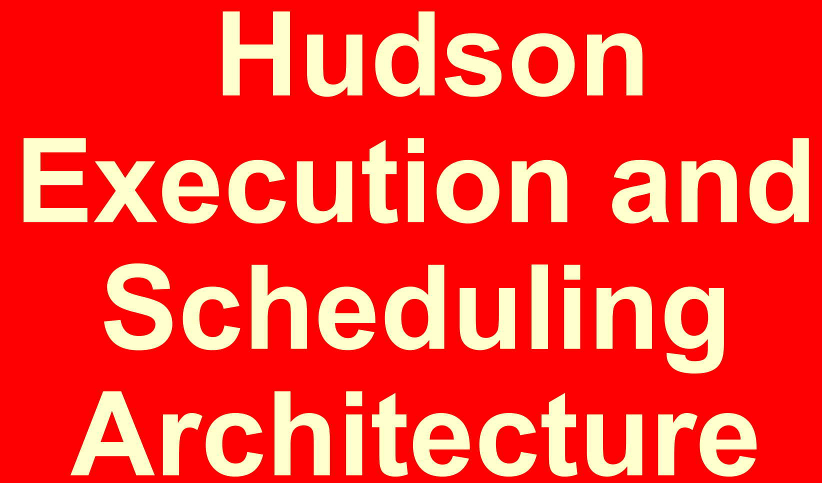
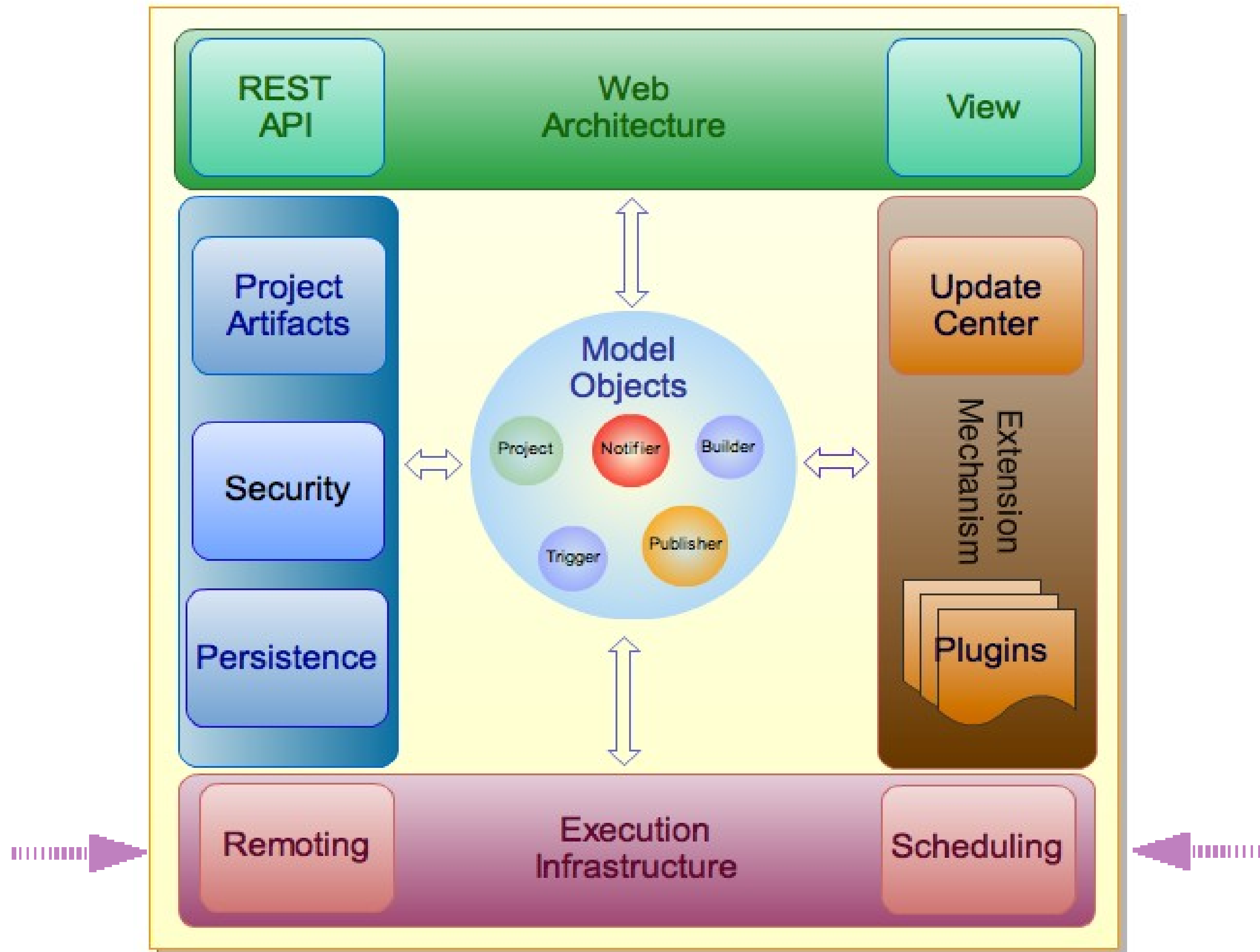
Hudson Execution and Scheduling Architecture



ORACLE

Winston Prakash

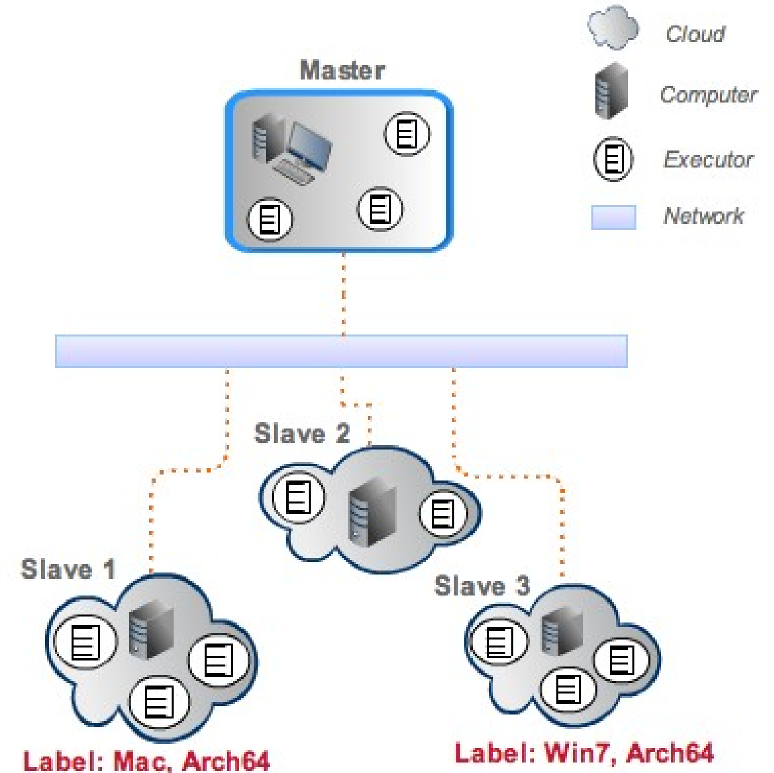
Hudson Architecture Overview



The Execution Architecture

Hudson is a distributed build system and it is accomplished via "master/slave" configuration.

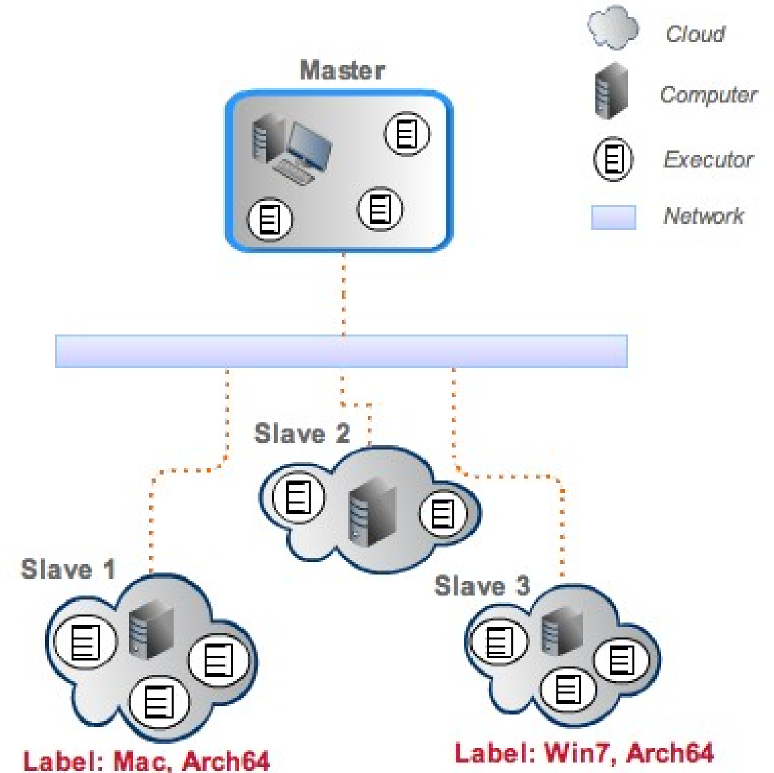
- Master – A full-fledged hudson server running on a machine. It can do build on its owb or requests a slave to do the build. It also provides the user view; Dashboard.
- Slave – An agent running on the same machine or different machine and builds on behalf of the master. A slave can live anywhere in the cloud as long as it is accessible to the master via TCP/IP connection.
- Executor – A thread that does the actual run of the build. Master or slave can have multiple executors (expected to equivalent of number of cores in the CPU)
- Label – String provide by Slave to advertise its capabilities.



The Execution Architecture

Hudson is a distributed build system and it is accomplished via "master/slave" configuration.

- Master – A full-fledged hudson server running on a machine. It can do build on its owb or requests a slave to do the build. It also provides the user view; Dashboard.
- Slave – An agent running on the same machine or different machine and builds on behalf of the master. A slave can live anywhere in the cloud as long as it is accessible to the master via TCP/IP connection.
- Executor – A thread that does the actual run of the build. Master or slave can have multiple executors (expected to equivalent of number of cores in the CPU)
- Label – String provide by Slave to advertise its capabilities.



The Slave

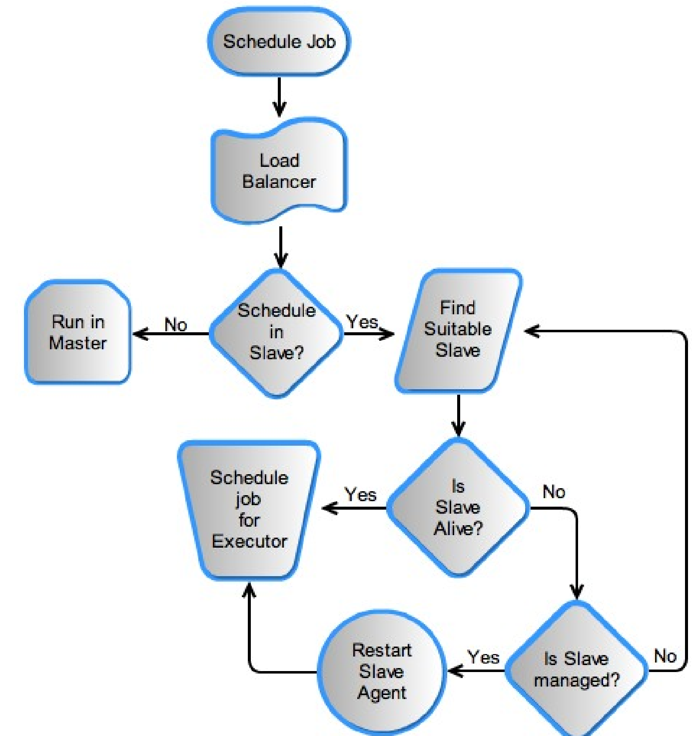
Slaves are computers that are set up to build projects for a Master Hudson. Slave machines run a separate program called "Slave Agent". For security reason the origination of the "Slave Agent" is always from the Master Hudson. There are various ways to start slave agents

Managed Slaves

The Slave Agent of the Managed Slaves are in the control of Hudson Master. If the Slave Agent dies by any chance, Hudson restarts it when its services are required. In some case like EC2, VmWare or VirtualBox slaves, the VM is provisioned and managed (activate and passivate) on demand.

Unmanaged Slaves

In unmanaged slave, when the Slave Agent is launched, then it is not managed by the Master Hudson. Typical example is the Slave Agent launched via JNLP. Hudson has no control over the Slave Agent, so if it dies it has to be invoked manually by the user again.



Master Slave Communication

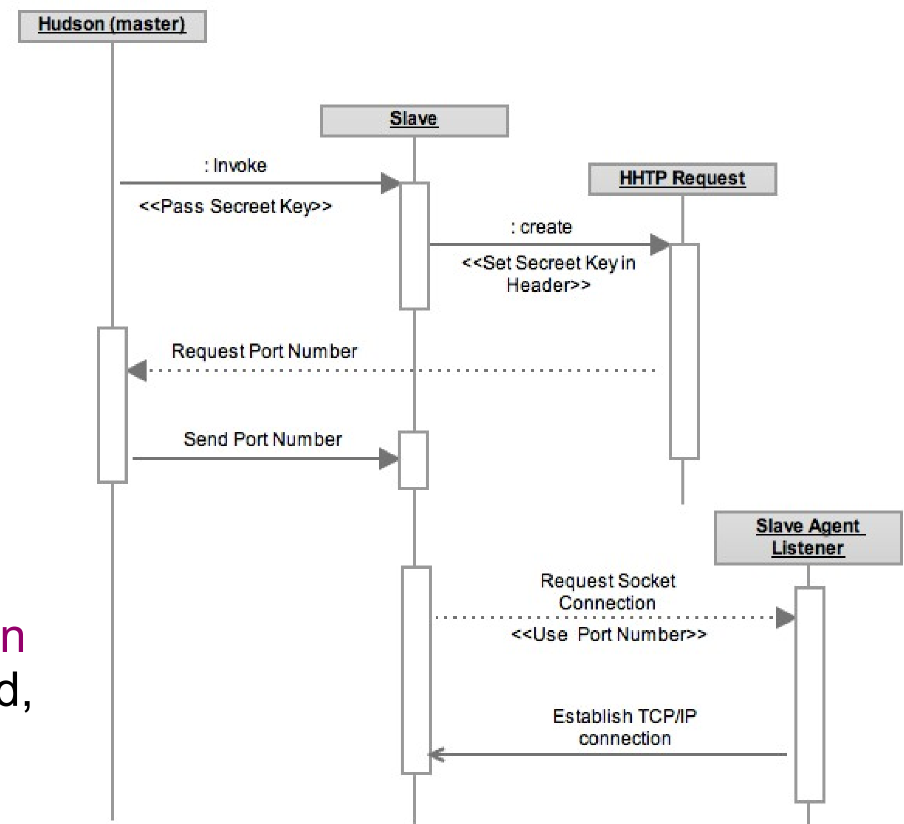
Slave Agent and Hudson Master need to establish a bi-directional byte stream (for example a TCP/IP socket) to do business together.

When the Slave Agent was invoked by Hudson Master a secret Key is passed as a parameter to the Main class.

Ex. `Java -jar slave.jar -credential <secretKey> -url <Master URL>`

When the Slave Agent is started, it tries to create a HTTP connection to the master putting the secret key in the header. Hudson responds with the port number on which the SlaveAgentListener is listening for Master-Slave communication.

Slave Agent requests a Secure socket connection with that port number and credentials. If accepted, a secure communication channel is established between master and slave



Building Blocks of the Execution System

The entire execution system can be broadly divided in to the following components.

Execution Service

Does the actual computation in a master-slave configuration

Queueing

Responsible for queuing the jobs and
does the scheduling logic.

Job Type

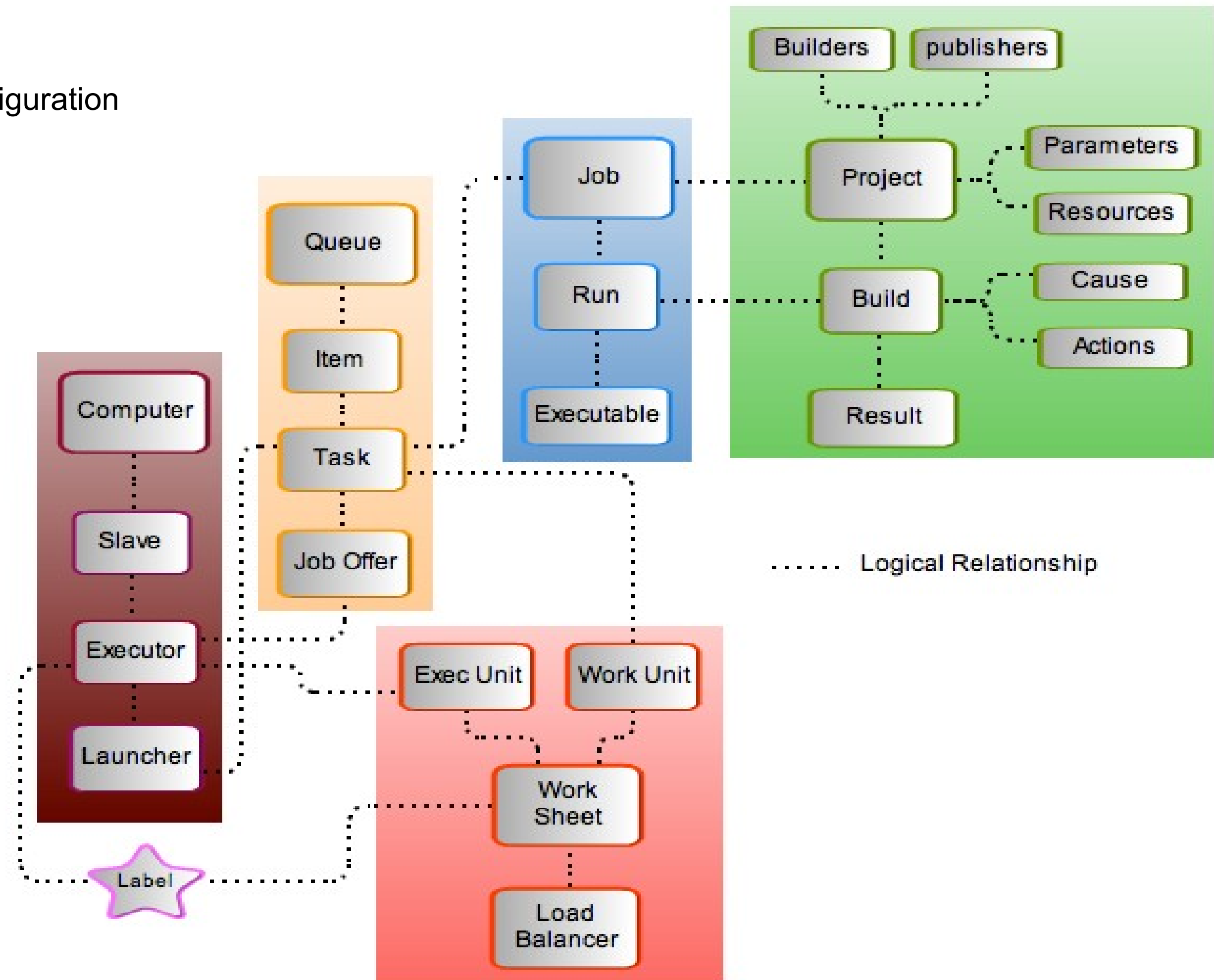
Defines the job types and run types

Load Balancer

Responsible for channeling the queued
jobs to the right execution components

Project

Defines the objects required to build a project



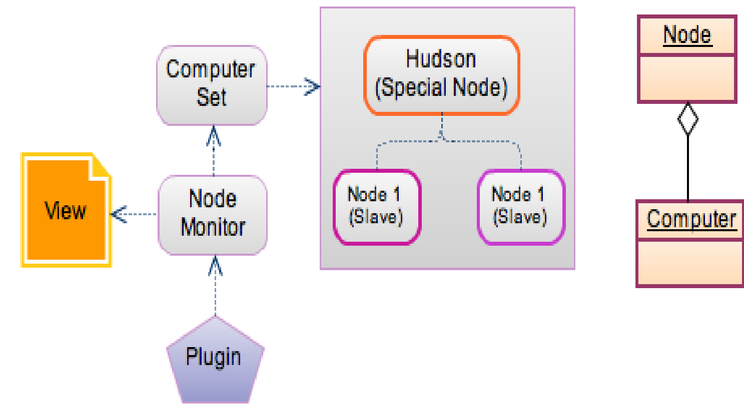
Computer Object

A Node is a base type for Hudson Slave.

A Slave is a unit that provides service for execution via Computer, so has an aggregation relationship with a computer object.

Hudson Object itself is a special type of Node that is consider as Master.

A NodeMonitor is an Extension point hooks for Plugins to provide View and configuration for Nodes.



Containment	Status	REST service	Message	Events
Executors Environment Variables HostName LogRecords Node ThreadDump (of Slave) BuildTimelineWidget ACL RunList	Alive Idle AcceptingTasks JNLPAgent Node LaunchSupported Online Temporarily Offline	Delete DumpExportTable LaunchSlaveAgent ProgressiveLog RSSAll Script (system diagnostics) ScriptText (run groovy text) Temporarily Offline	Connect/Disconnect CliConnect/CliDisconnect checkPermission	Task Accepted Task Completed Task Completed With Problem

Executor Object

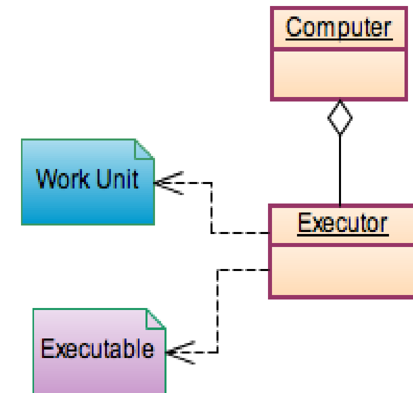
Executor Object is the thread that does the builds .

Once it gets a Work-Unit (Project), it create the Executable (For a project it is a Build)

Does a Synchronized start (make sure resources are synchronized)

If Work-Unit is actionable (contributed by Plugins) add all actions to Executable

Execute using ResourceSynchronizer.execute()



Containment	Status	REST service	Message	Events
Workspace Executable WorkUnit	Idle LikelyStuck	stop	ElapsedTime Number Of Executors Progress TimeStamp Owner Run RemainingTime	None

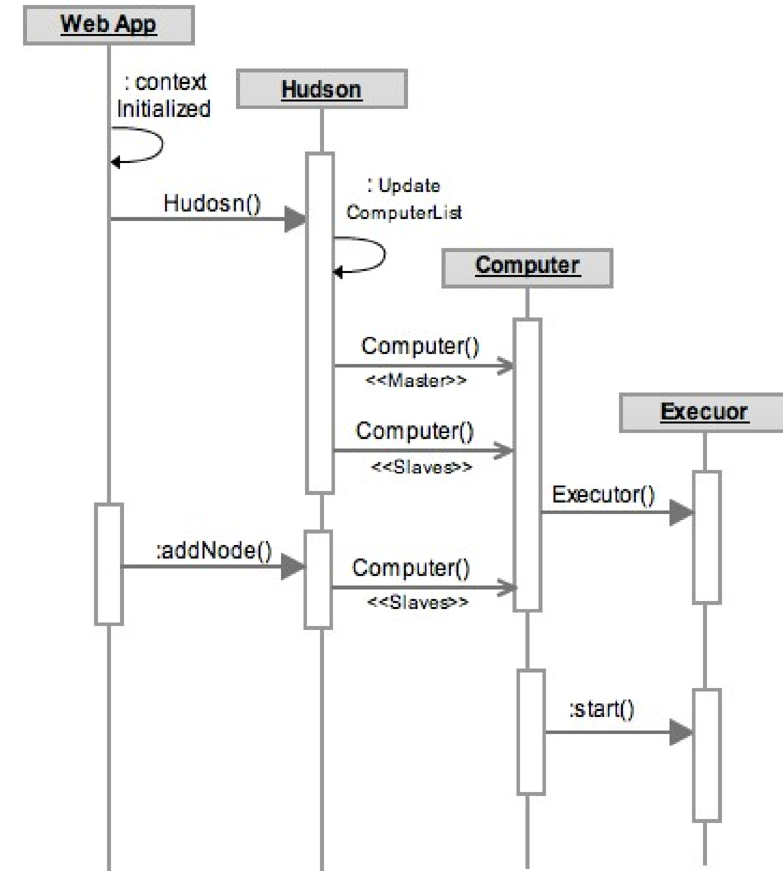
Executor Initialization

When Hudson Web Application Context starts for the first time, it always has one Node (Master) which is started by default with 2 Executors.

During initialization process it starts all the Executor threads. The Executors then goes in to waiting state to accept jobs for execution.

If Hudson is restarted, if local or remote slaves are added, if they are managed then they are started and message sent to the slaves to start the Executor Thread.

In case of Unmanaged Slaves, they become zombies?

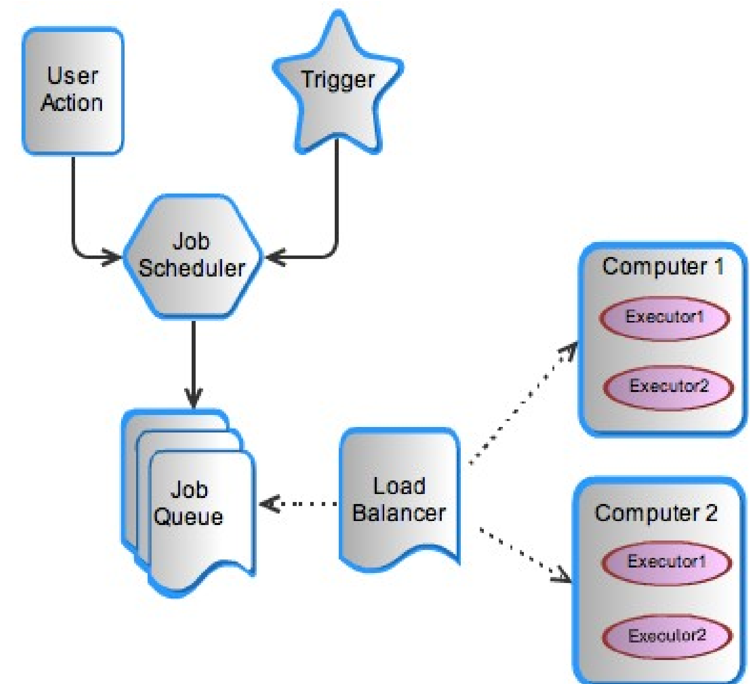


Note: If hudson is properly shutdown a broadcast message is sent before the actual shutdown, giving a chance for Slaves to clean up the resources.

Job Submission

A Job gets submitted by couple of methods

- User Action – User explicitly clicks build button to start the job.
- Trigger – A trigger like SCM modification
- Upstream project starts the downstream project build
- REST API invocation.



In a broad sense, once a Job gets submitted following happens

- A Job Scheduler put the job in a Job Queue.
- A Load Balancer determines if the Job should be built by Master or off loaded to one of the Slave computer.
- Next available Executor in the Master or Slave computer picks up the Job from the queue and build it.

Scheduling logic

After the Job is submitted, it goes through the following states, before picked up by a Executor for Execution.

Waiting List

Items that can not be built yet, waiting for queue maintenance.

Blocked Job

Can be built, but blocked due to various reasons - depending on upstream project or resource

Buildable Job

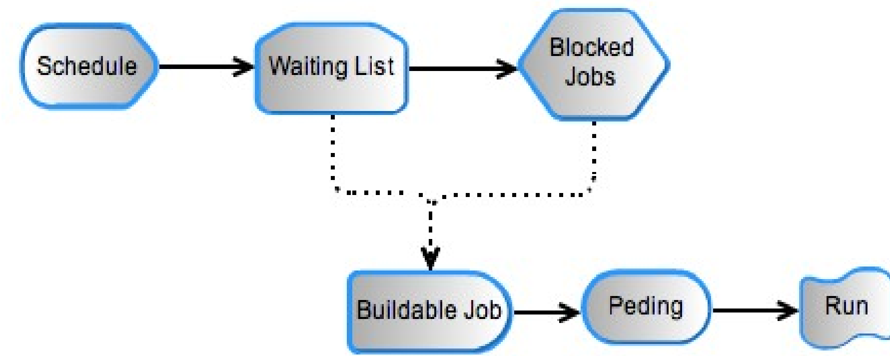
Promoted to buildable and will be start building as soon as an Executor becomes available.

Pending

Got the Executor, but in pending state due to quiet period etc.

Run state

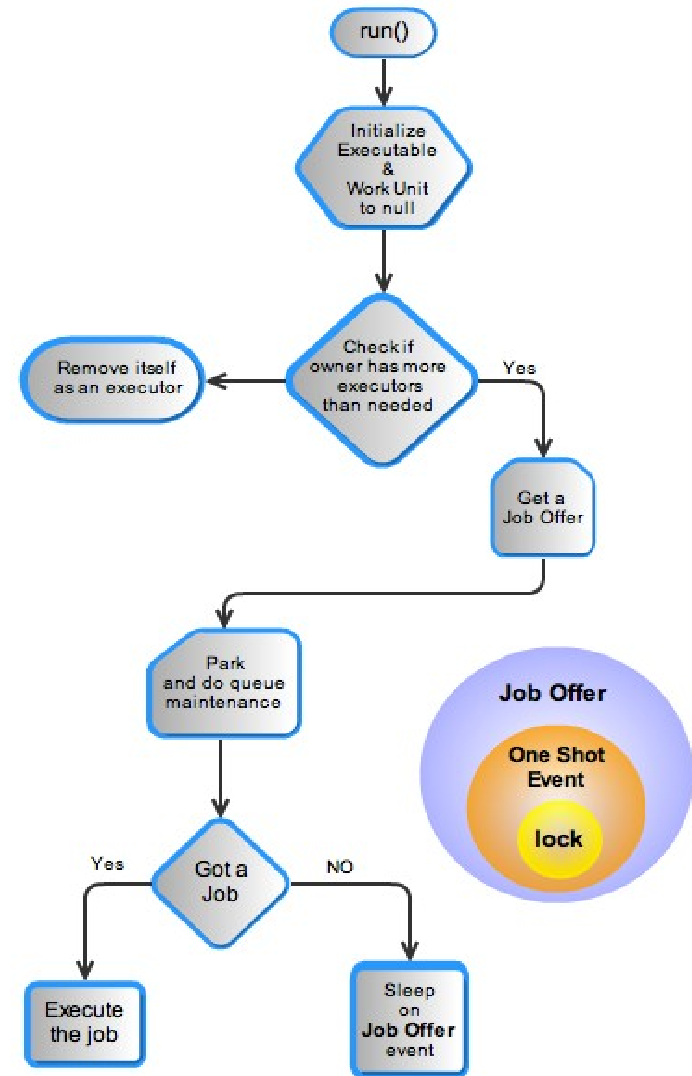
When the job is at run state, it can be scheduled again (Ex. Due to SCM trigger)



Executor Run Logic

When Hudson initializes or a new Slave node is added, always all their Executors are started. In the run state,

- Always initializes Executable and Work Unit to Null (in case of Hudson restart)
- Check if there are more number of Executors than needed, then remove itself.
- Get a Job Offer
- Park itself
- Do queue maintenance
- Go to sleep
- Wait for a wake up signal
- If got wake up signal execute the Job



Queue Maintenance

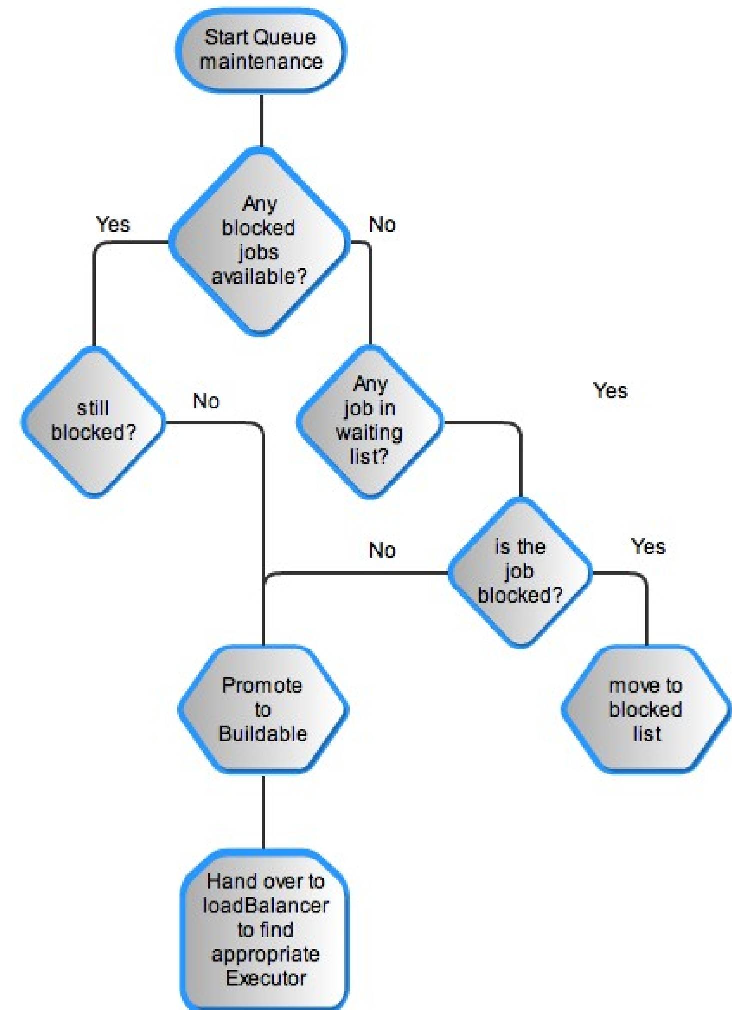
The Job reaches various states by Queue Maintenance. The maintenance occurs at two point of time.

- When a new Job got submitted
- When a Executor finishes an execution and ready for a new Job.

During maintenance the queue is first checked to see if it has any blocked Jobs. If exists, check if it is still blocked, if not move it to the buildable list.

If there are any in the waiting list, check if they are blocked. If so moving them to blocked list, else move them to buildable list.

Send signal to all waiting Executors.



Load Balancer

The fundamental principle of Load Balancer is to define which Executor executes which task. So it is a mapping problem between

- Tasks, which in the general has sets of SubTasks.
- Number of idle Executors

Mapping constraints:

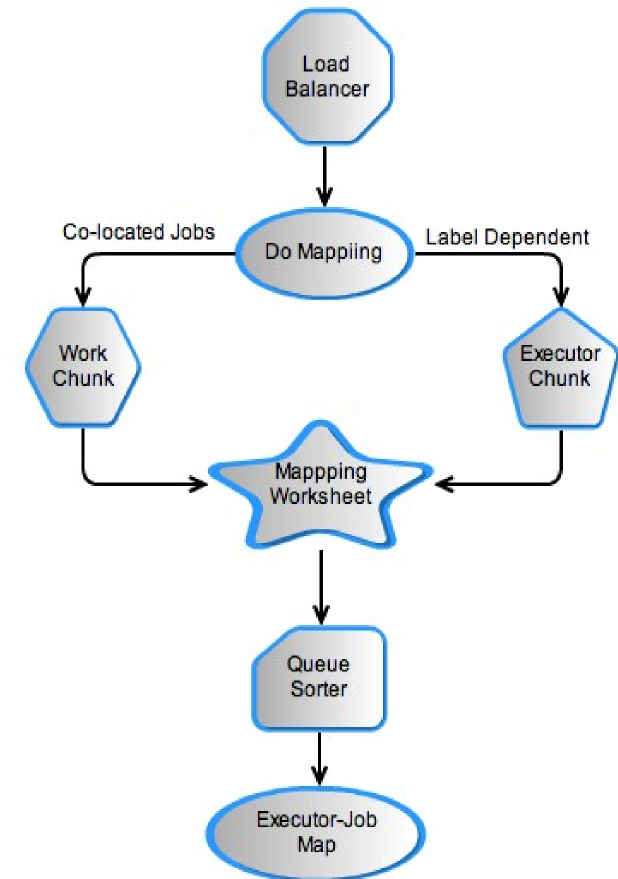
- Some of the subtasks need to be co-located on the same node.
- Tasks can specify that it can be only run on nodes that has particular label.

Mapping Definition:

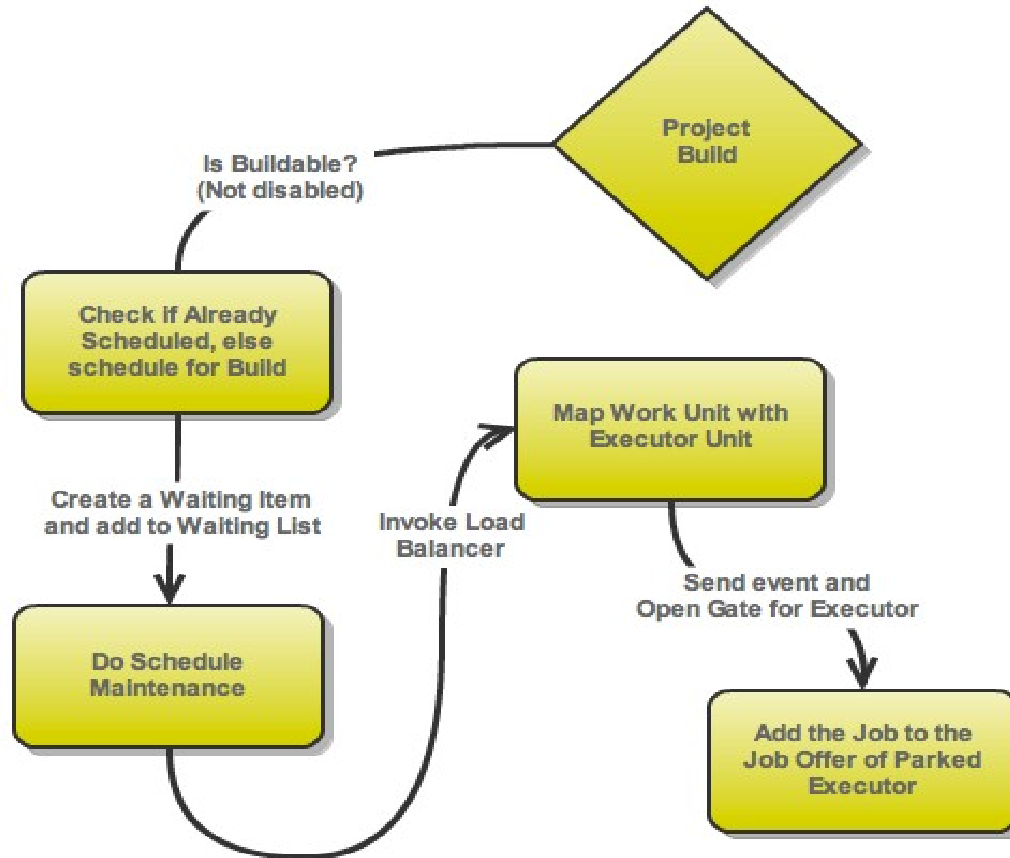
- The set of Tasks that need to be co-located as a single Work-Chunk.
- Set of all Executors from the same node as Executor-Chunk.

Solution:

Find a weighted matching between Work-Chunk and Executor-chunk using a Mapping-Worksheet. Sort the map based on Task execution order (Up stream → Down Stream project)



Project Build (Put it all together)



Steps of a Build

Following steps happen based on project configuration

SCM checkout

Based on SCM type source code is checked out

Pre-build

Invoked to indicate that the build is starting

Build wrapper

Prepare an environment for the build.

Builder runs

Actual building like calling Ant, Make, etc. happen.

Recording

Record the output from the build, such as test results.

Notification

Send out notifications, based on the results determined so far.