Jenkins PlugIns

Some important and useful PlugIns:

1) Copy Artifact Plugin:

This PlugIn helps to add a build step to copy artifacts from another project. The plugin lets you specify which build to copy artifacts from (e.g. the last successful/stable build, by build number, or by a build parameter).

Pipeline syntax

- To copy artifacts from the latest stable build of "sourceproject"

```
copyArtifacts(projectName: 'sourceproject');
```

- To copy artifacts from the specific build of "downstream"

```
def built = build('downstream');  // https://plugins.jenkins.io/pipeline-build-step

copyArtifacts(projectName: 'downstream', selector: specific("${built.number}"));
```

2) Disable specific Job from another Job using JOB DSL Plugin

The DSL script to use would be:

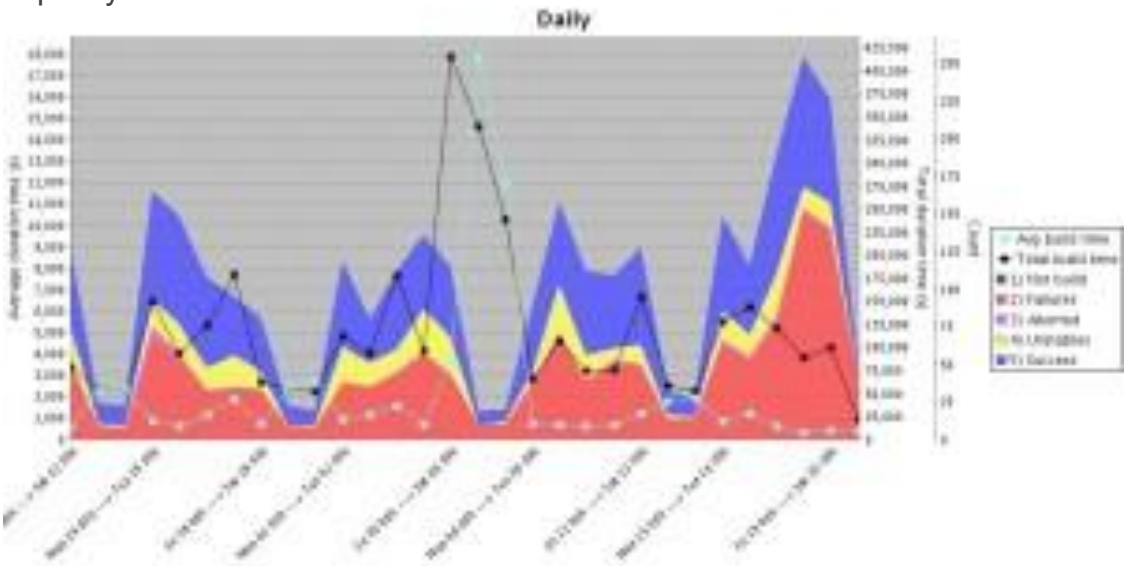
```
job("jobname"){
    using("jobname")
    disabled(true)
}
```

1. This is for Freestyle jobs. If you need to disable, for example, a Pipeline Job, use pipelineJobinstead of job.
2. Usually, "jobname" needs to include folders, using the Jenkins root as reference.
3. To reenable a job, use disabled(false).

3) Global Build Stats Plugin:

It’s essential to know your current capacity, usage and capability to prepare for capacity planning or system requirements, because the very first question is “what is your current

capacity?”



You need to know how many builds are happening on a daily and weekly basis. How much time is taken by various builds, what is the waiting period. This plugin give you enough data to answer the question. This plugin can provide you data as graph which can be further consumed.

4) **Job GeneratorPlugin–**

In big or growing organizations, its a bit difficult to maintain the jobs for a project when developers are working on various branches and releases. You want to give access to developers to create their own job and at the same time you don’t want developers to create arbitrary jobs which may not fall under company standards. This plugin gives you flexibility to define templates and developers can create new jobs with the help of job generator template. Configuration access can be disabled via roll-based authorization plugin.

5) **Disable-failed-job –**

In rapid development environments failing jobs due to various unnecessary reasons are a pain, It’s difficult to keep track of such jobs and disable/delete them. This plugin can solve the problem by defining the upper number of consecutive failed builds and then disable automatically.

6) **Embeddable-build-status –**

This plugin can give you a link which can be pasted anywhere (example github project) to expose the status of the build and users can get the current state of the job while looking at project.

7) Exclusion –

This plugin enables you to handle conflicts between jobs. You can assign a resource (or lock) in multiple jobs, when build is executed, it will acquire the lock and other builds (if fired) will wait until lock is released.

8) GitHub/GitLab Pull Request Builder-

This is one of the best plugins I found to provide support to automate code review up to a certain extent in github/gitlab. This plugin does amazing job once defined for the project. For any new pull request this plugin does fox merge and runs the build on the code as well as gathers the necessary static analysis (if configured) or build results and comment back the status to pull request. This helps reviewers get an idea of the health of the code which is going to be merged. You can even define automatic merge if build passes.

9) Hudson Extended Read Permission Plugin-

During initial days of Jenkins setting configuration access was not available to any developer to make sure they do not change anything but developers used to ask to view configurations to know how job has been setup, they may want to see the build steps. Extended read plugin can provide option to developer to view the configuration without giving them write access.

10) Post+build+task

There may be a need for performing some actions on the basis of the results of a build, for example if build passed you may want to upload artifact(ex debian) to some repo (apt) or perform some packaging part or similar. In case of failure you may want to roll back something (like release) . This plugin helps you to define the pass/fail criteria and let’s you decide what to do after that.

11) JDK Parameter Plugin–

This plugin is useful for an organization where many projects are using different versions of java. This plugin let’s you choose the java version during run time of the build.

12) Job Configuration History Plugin-

This is one of my favorite plugins. This plugin lets you keep track of config changes in each build including who did it. You can easily revert back to any previous config if you want.

13) Multiple SCMs plugin–

Default SCM section provides only one source control tool/URL option, what if you want to check out from more than 2 repos from multiple source control tools (like svn and git). This plugin will come in handy in such scenarios. This plugin will facilitate users to add any number of SCM URLs to checkout the code.

14) Parameterized Trigger plugin–

Another one of my favorite plugins. This plugin allows you to have user input as a variable and use on run time. This is the most used plugin in dynamic environments where you have lots of options and user-defined values to be used in the build which may keep changing.

15) Pre SCM BuildStep Plugin-

Just like post build task plugin, you may have requirements to perform some action even before checkout happens for the job, for example you may want to perform merging of the branch before the build and then checkout. This plugin can come in handy in various conditions and gives you flexibility when running the job.

16) SCM Sync Configuration Plugin-

Backup, this is the most important task of any administrator. Without regular backup, the whole system cannot be reliable; this plugin provides you features to backup live Jenkins configs to any source control tool. It will keep committing the config files (including Jenkins and jobs) to SCM repository as soon as there is any change.

17) Configuration Slicing Plugin –

This plugin comes in very handy when you want to make bulk changes in multiple jobs. This plugins allows you to change values of various fields like email, timer, shell script, configurations etc at one go.

18)Determine if given Job is running using Jenkins API..

```
import hudson.model.*
def version = build.buildVariableResolver.resolve("VERSION")
println "VERSION=$version"
def nextJobName = 'MY_NEXT_JOB'
def nextJob = Hudson.instance.getItem(nextJobName)
def running = nextJob.lastBuild.building
if (running) {
   println "${nextJobName} is already running. Not launching"
} else {
   println "${nextJobName} is not running. Launching..."
   def params = [
      new StringParameterValue('VERSION', version)
   ]
   nextJob.scheduleBuild2(0, new Cause.UpstreamCause(build), new
ParametersAction(params))
}
```