**Chapter 10. Advanced Builds**

**Introduction**

In this chapter, we will look at some more advanced build job setups. We will discuss parameterized builds, which allow Jenkins to prompt the user for additional parameters that will be passed into the build job, andmulticonfiguration build jobs, which let you run a single build job through a large number of variations. We will look at how to run build jobs in parallel, and wait for the outcome of one or more build jobs before continuing. And we will see how to implement build promotion strategies and build pipelines so that Jenkins can be used not only as a build server, but also as a deployment server.

## Parameterized Build Jobs

Parameterized builds are a powerful concept that enable you to add another dimension to your build jobs.

The Parameterized Build plugin lets you configure parameters for your build job, that can be either entered by the user when the build job is triggered, or (as we will see later) from another build job.

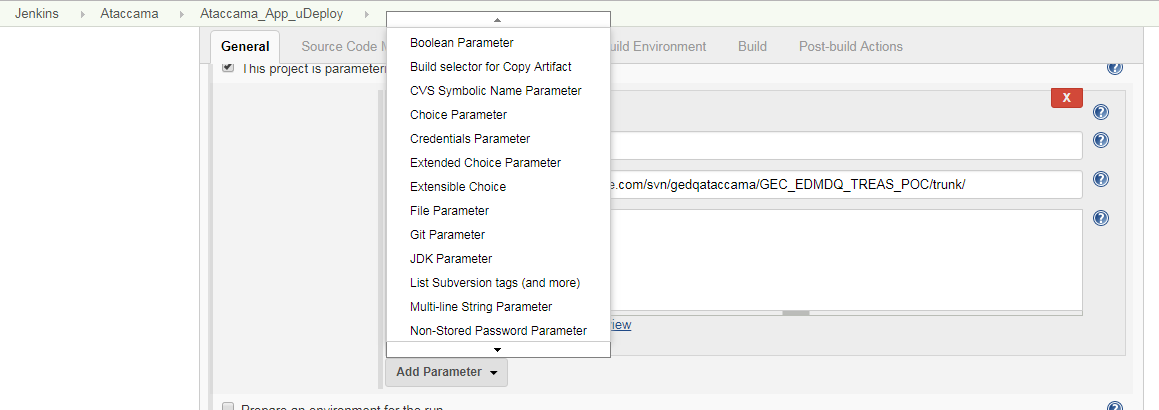
For example, you might have a deployment build job, where you want to choose the target environment in a drop-down list when you start the build job. Or you may want to specify the version of the application you want to deploy. Or, when running a build job involving web tests, you might want to specify the browser to run your Selenium or WebDriver tests in. You can even upload a file to be used by the build job.

Note that it is the job of the build script to analyze and process the parameter values correctly—Jenkins simply provides a user interface for users to enter values for the parameters, and passes these parameters to the build script.

### CREATING A PARAMETERIZED BUILD JOB

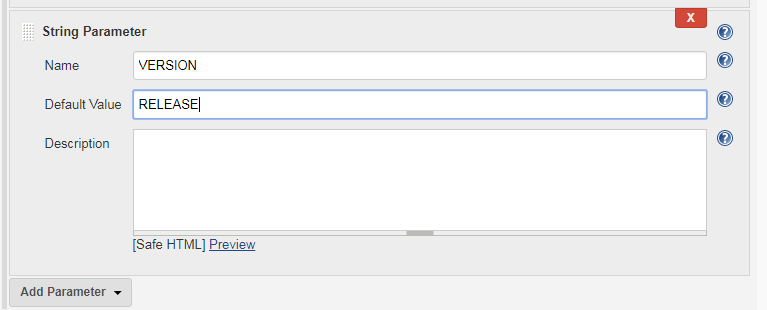
You install the Parameterized Build plugin as usual, via the Plugin Manager screen. Once you have done this, configuring a parameterized build job is straightforward. Just tick the “This build is parameterized” option and click Add Parameter to add a new build job parameter .You can add parameters to any sort of build, and you can add as many parameters as you want for a given build job.

To add a parameter to your build job, just pick the parameter type in the drop-down list. This will let you configure the details of your parameter . You can choose from several different parameter types, such as Strings, Booleans, and drop-down lists.

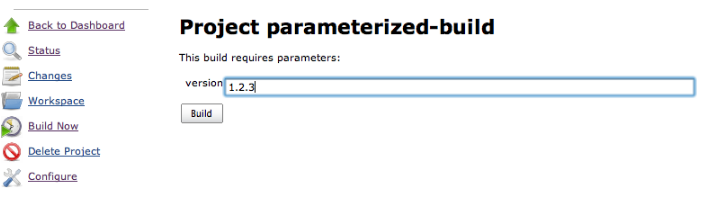


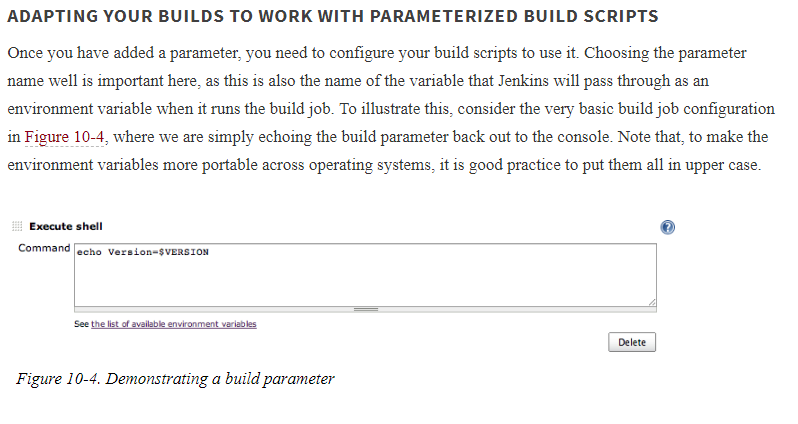
Depending on the type you choose, you will have to enter slightly different configuration values, but the basic process is identical. All parameter types, with the exception of the File parameter, have a name and a description, and most often a default value.

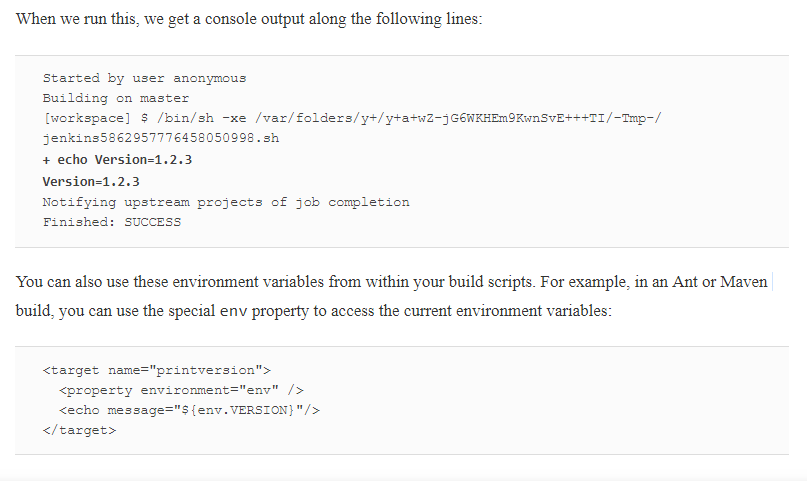
In [below](https://www.safaribooksonline.com/library/view/jenkins-the-definitive/9781449311155/ch10.html#fig-hudson-enter-string-parameter) pic, for example, we are adding a parameter called version to a deployment build job. The default value (RELEASE) will be initially displayed when Jenkins prompts the user for this parameter, so if the user doesn’t change anything, this value will be used.



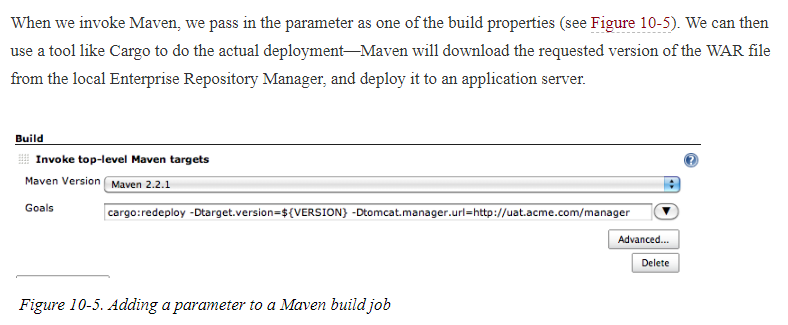
When the user starts a parameterized build job (parameterized build jobs are very often started manually), Jenkins will propose a page where the user can enter values for each of the build job’s parameters



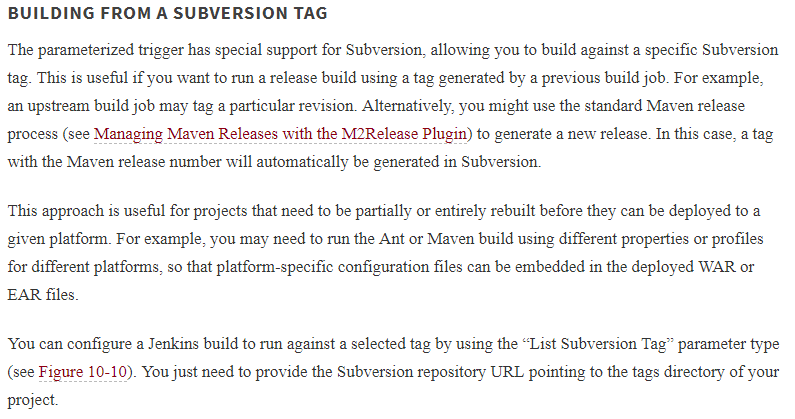


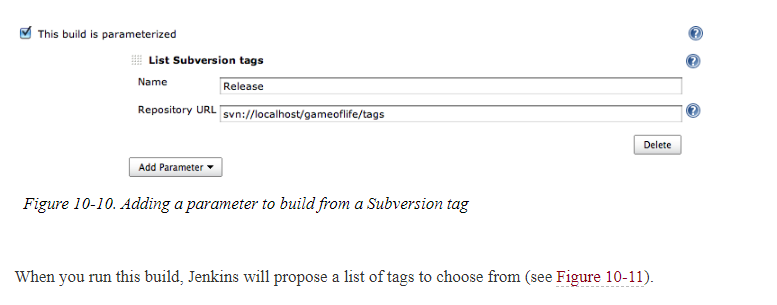




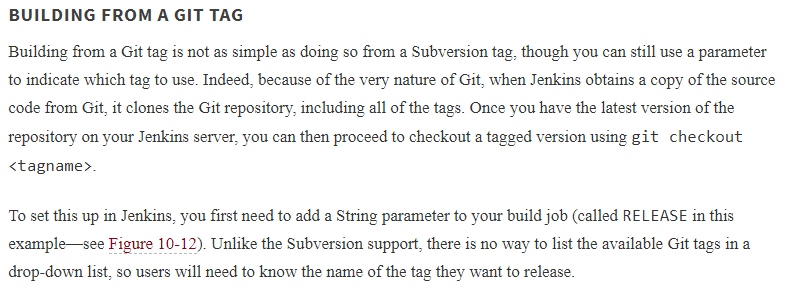


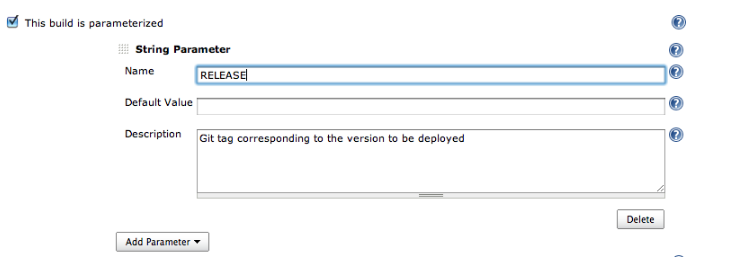
That, in a nutshell, is how you can integrate build job parameters into your build. In addition to plain old String parameters, however, there are a few more sophisticated parameter types, that we will look at in the following paragraphs ( I’m skipping few paragraphs which are part of the Jenkins 2 up & running tutorial files)

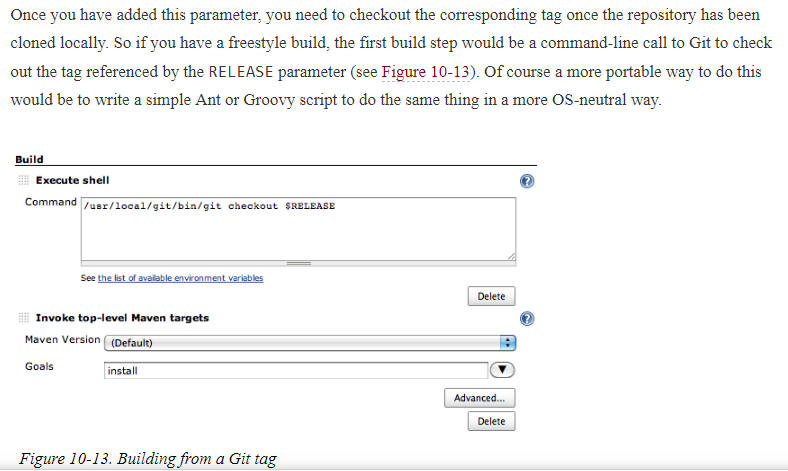






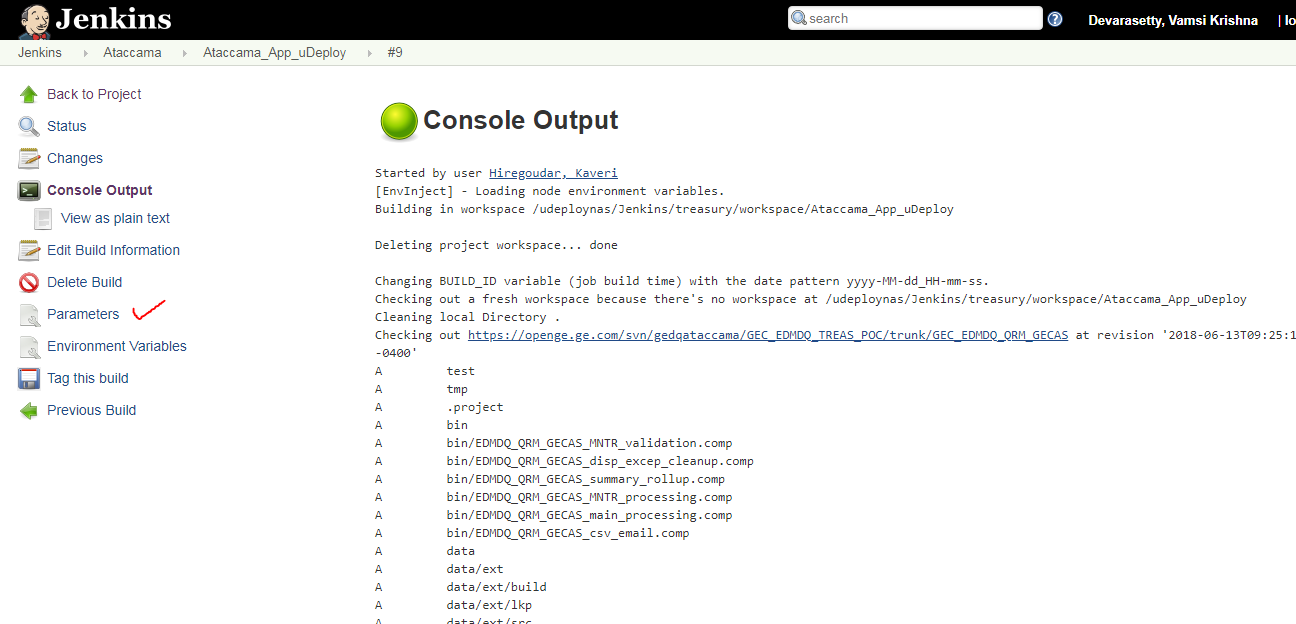


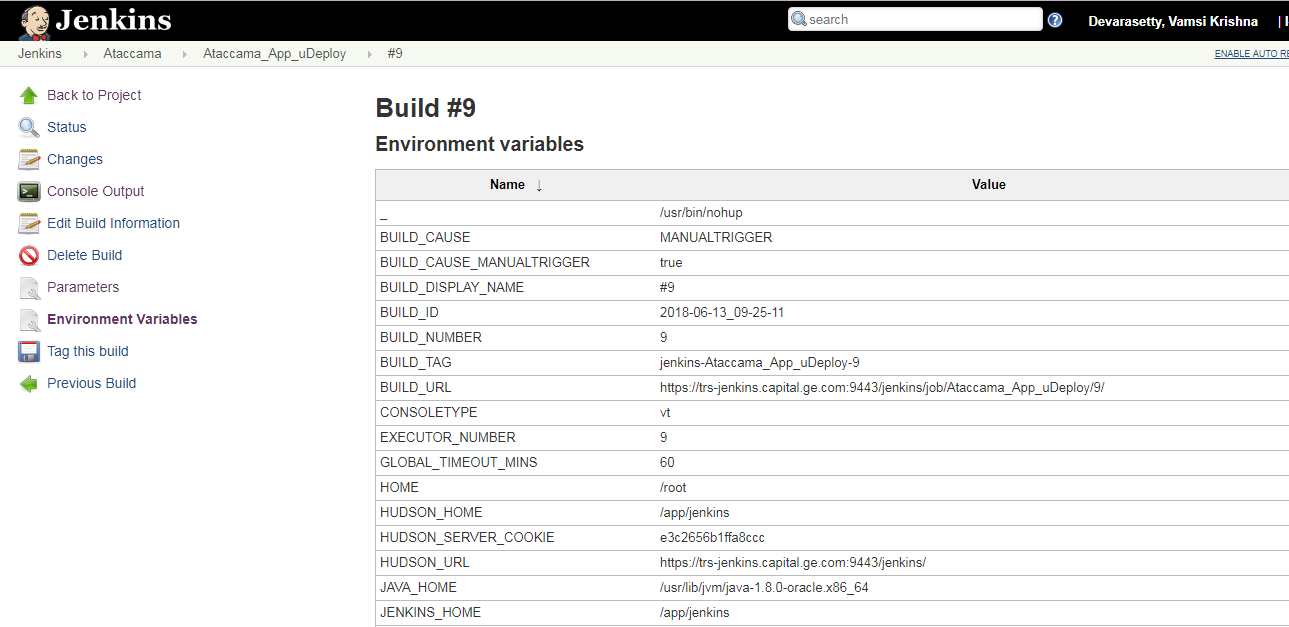
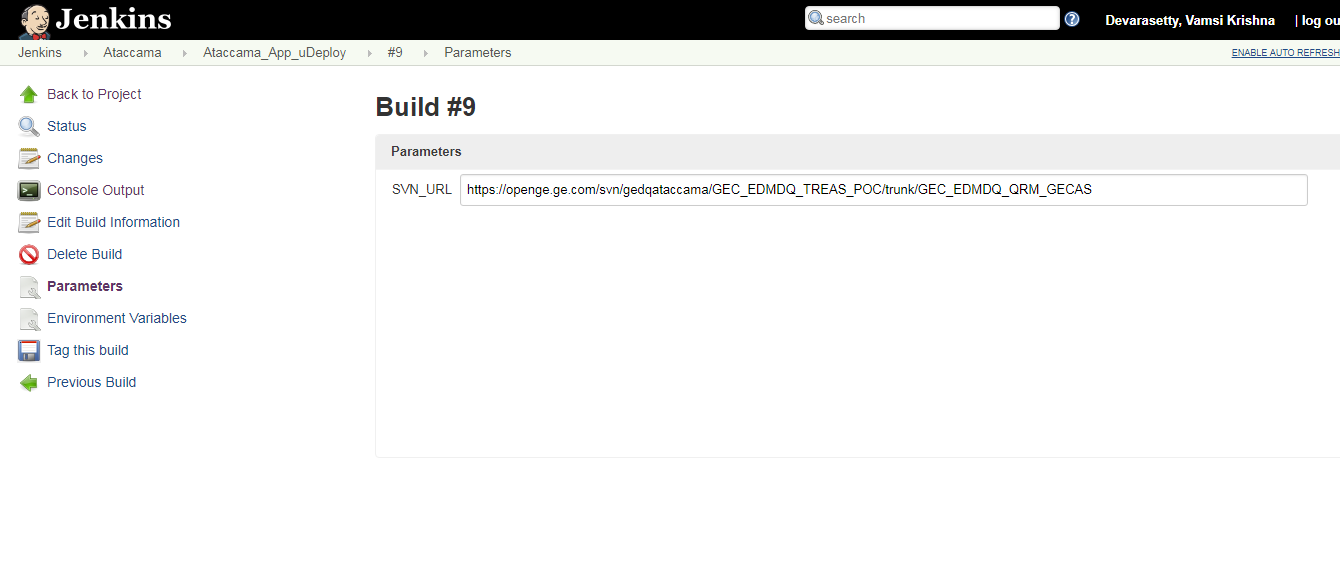


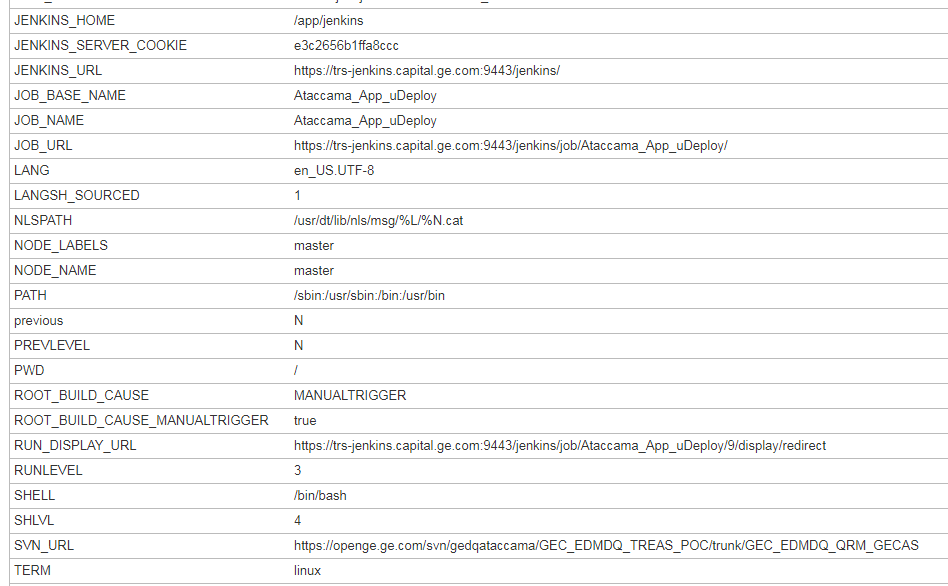


### PARAMETERIZED BUILD JOB HISTORY

Finally, it is indispensable to know what parameters were used to run a particular parameterized build. For example, in an automated deployment build job, it is useful to know exactly what version was actually deployed. Fortunately, Jenkins stores these values in the build history , so you can always go back and take a look.







## Parameterized Triggers

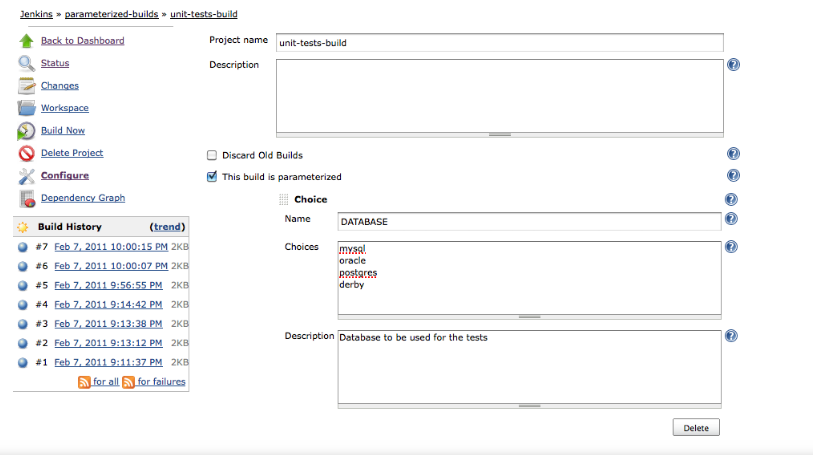
When you trigger another build job from within a parameterized build job, it is often useful to be able to pass the parameters of the current build job to the new one. Suppose, for example, that you have an application that needs to be tested against several different databases. As we have seen, you could do this by setting up a parameterized build job that accepts the target database as a parameter. You may want to kick off a series of builds, all of which will need this parameter.

If you try to do this using the conventional “Build other projects” option in the Post-Build Actions section, it won’t work. In fact, you can’t trigger a parameterized build in this way.

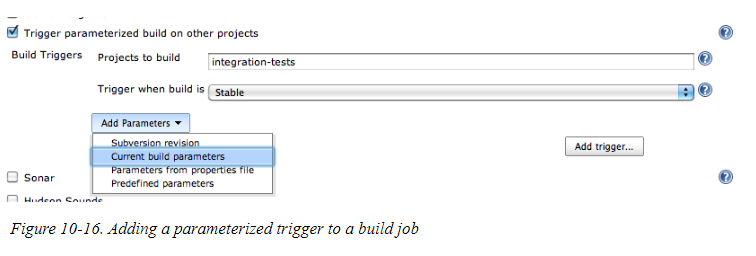
However, you can do this using the Jenkins Parameterized Trigger plugin. This plugin lets you configure your build jobs to both trigger parameterized builds, and to pass arbitrary parameters to these builds.

Once you install this plugin, you will find the option of “Triggering parameterized builds on other projects” in your build job configuration page. This lets you start another build job in a number of ways. In particular, it lets you kick off a subsequent build job, passing the current parameters to this new build job, which is impossible to do with a normal triggered build. The best way to see how this works is through an example.

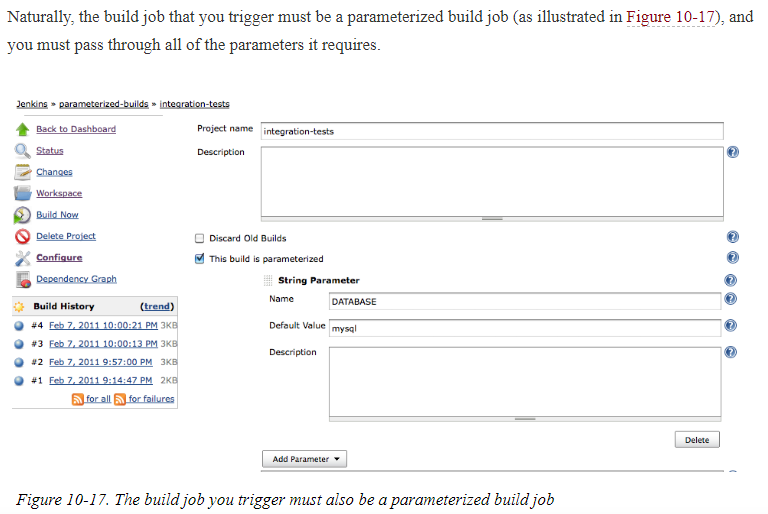
Below we have an initial build job. This build job takes a single parameter, DATABASE, which specifies the database to be used for the tests. As we have seen, the user will be prompted to enter this value whenever the build is started.

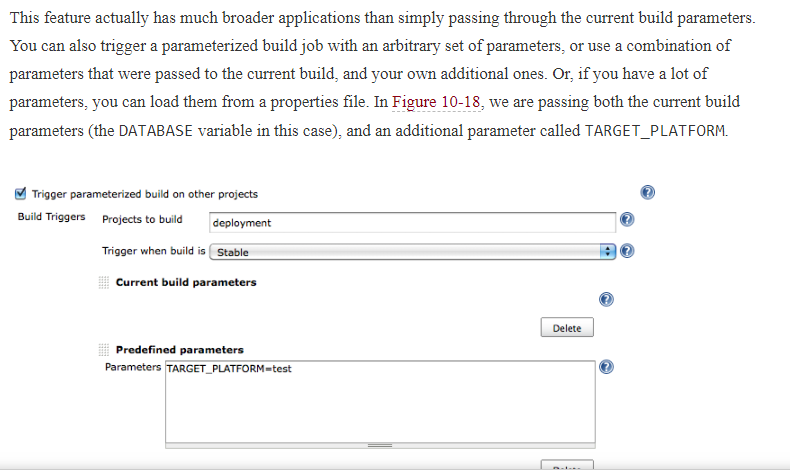


Now suppose we want to trigger a second build job to run more comprehensive integration tests once this first build job has finished. However, we need it to run the tests against the same database. We can do this by setting up a parameterized trigger to start this second build job



In this case, we are simply passing through the current build parameters. This second build job will automatically be started after the first one, with the DATABASE parameter value provided by the user. You can also fine-tune the triggering policy, by telling Jenkins when the build should be triggered. Typically, you would only trigger a downstream build after your build has completed successfully, but with the Parameterized Trigger plugin you can also configure builds to be triggered even if the build is unstable, only when the build fails or ask for it to be triggered no matter what the outcome of the first build. You can even set up multiple triggers for the same build job

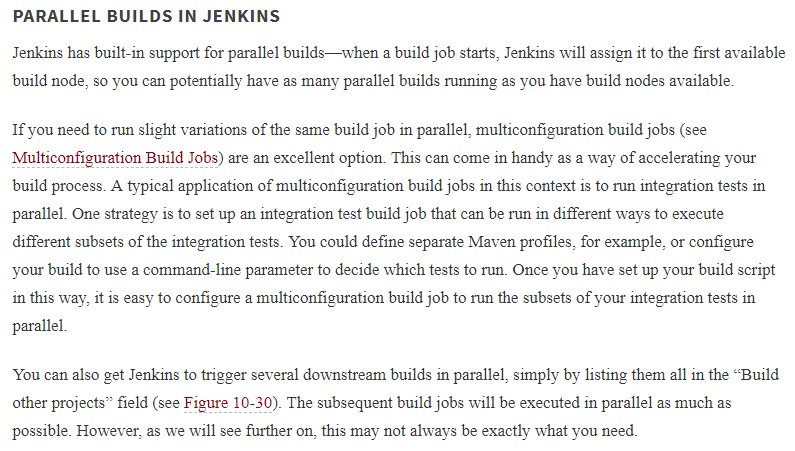


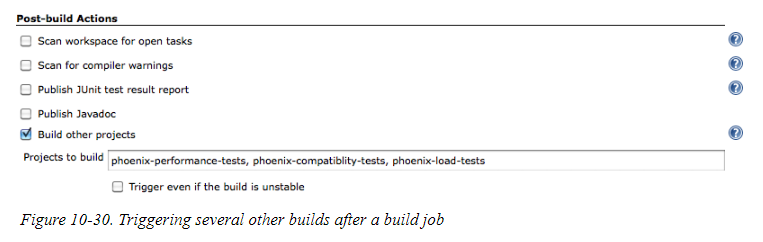


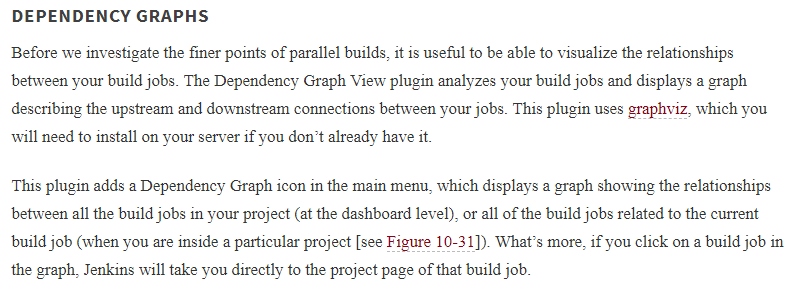
After this there is multi configuration job which is sort not an important topic

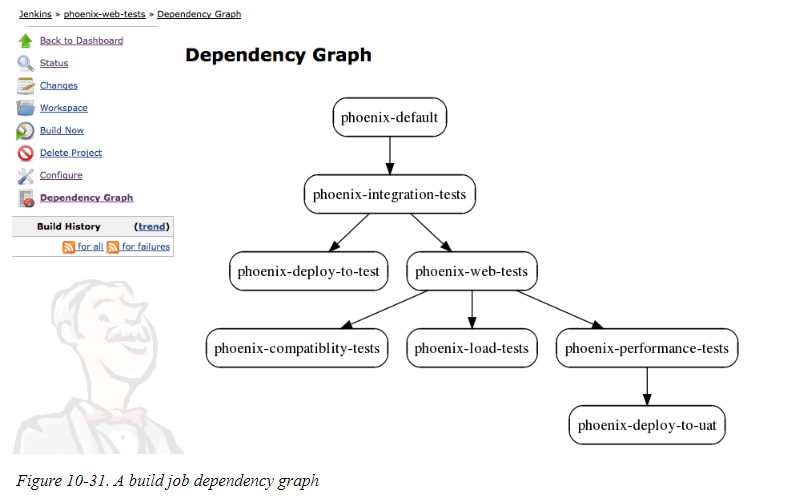
## Coordinating Your Builds

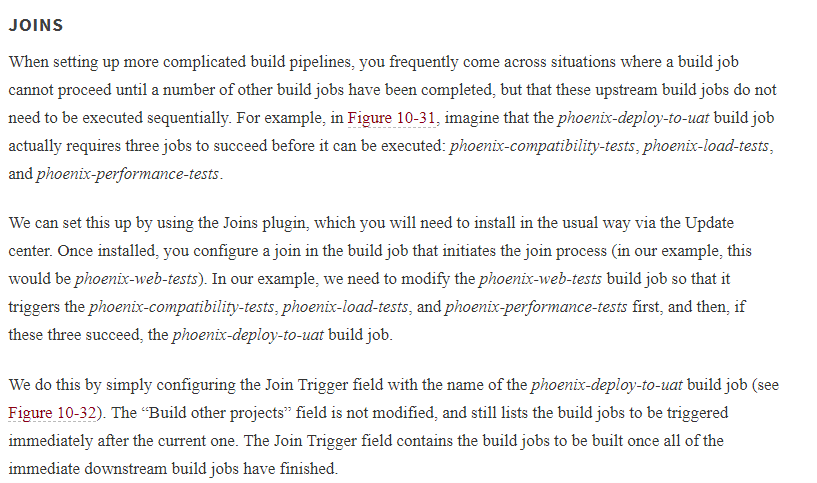
Triggering downstream build jobs is easy enough. However, when setting up larger and more complicated build job setups, you sometimes would like builds to be able to run concurrently, or possibly wait for certain build jobs to finish before proceeding. In this section, we will look at techniques and plugins that can help you do this.

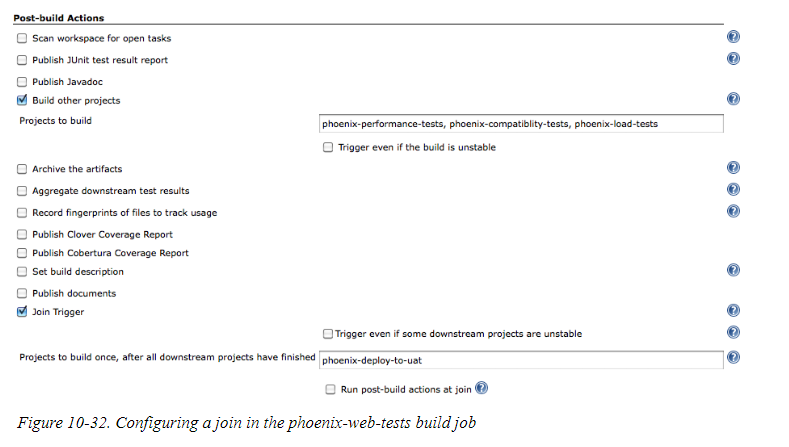


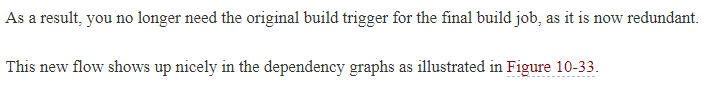


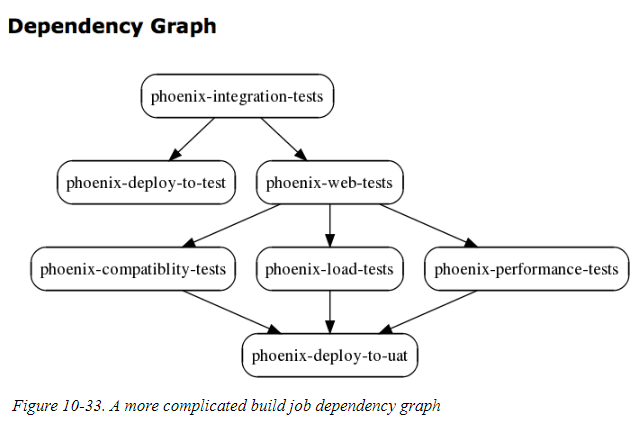












## Build Pipelines and Promotions

Continuous Integration is not just about automatically building and testing software, but can also help in the broader context of the software product development and release life cycle. In many organizations, the life of a particular version of an application or product starts out in development. When it is deemed ready, it is passed on to a QA team for testing. If they consider the version acceptable, they pass it on to selected users for more testing in a User Acceptance Testing (UAT) environment. And if the users are happy, it is shipped out into production. Of course, there are almost as many variations on this as there are software development teams, but one common principle is that specific versions of your software are selected, according to certain quality-related criteria, to be “promoted” to the next stage of the life cycle. This is known as build promotion, and the broader process is known as a build pipeline. In this section, we will look at how you can implement build pipelines using Jenkins.