Jenkins

Creating and Maintaining Jobs

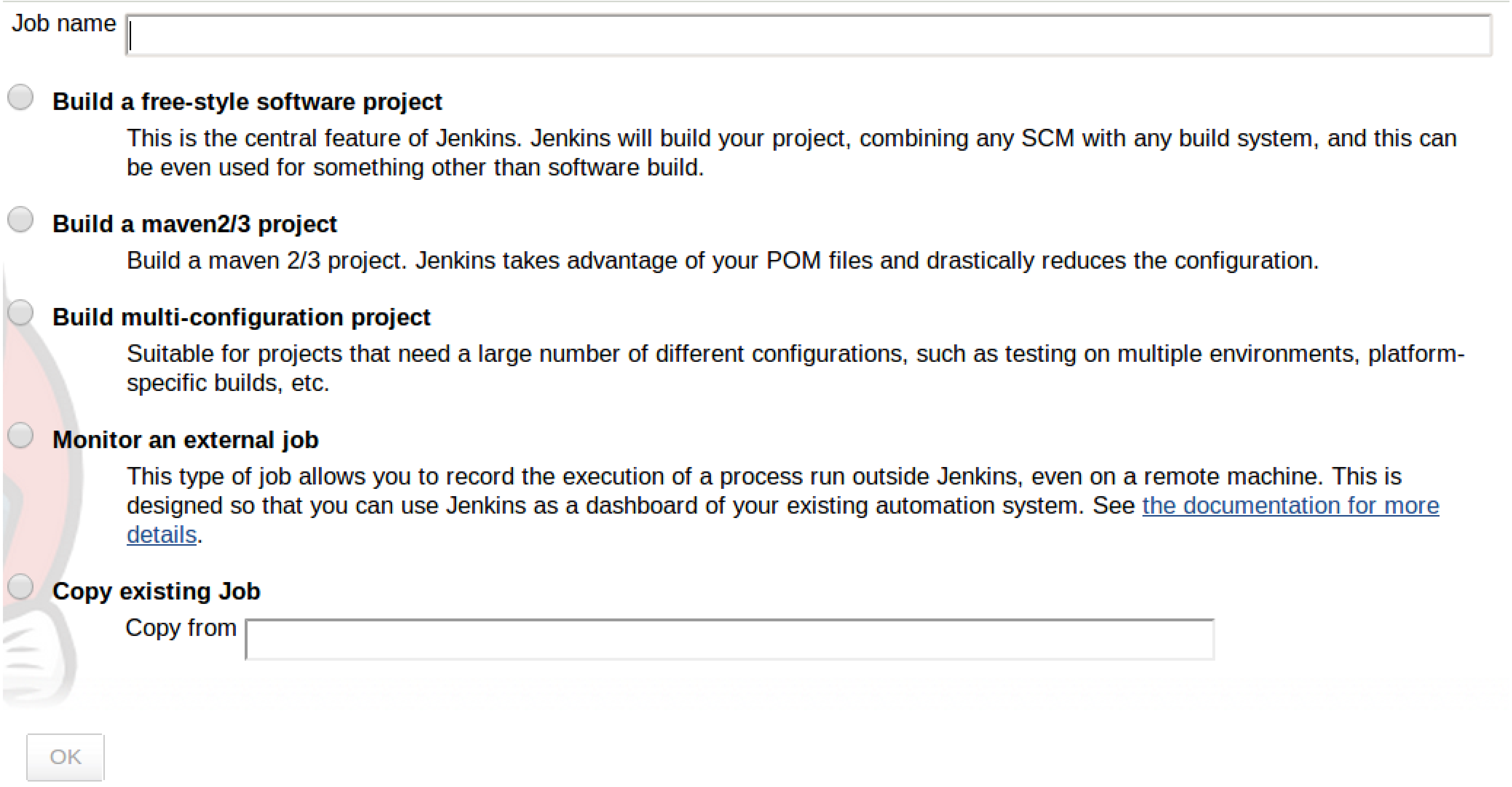
Section Contents

- Build Jobs Overview
- Creating a Freestyle Job
- Configuring Subversion
- Build
  - Triggers
  - Steps
- Post Build Actions
- Running Jobs

Build Jobs Overview

- Build Job is all/some compiling, testing, packaging, deploying or doing something with your project
- Typical scenario would be to create a unit test job, then an integration then an acceptance or functional test or documentation or other tasks
- Jenkins has 5 different Build Job Scenarios to choose from

Types of Jobs



Job name

Build a free-style software project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Build a maven2/3 project

Build a maven 2/3 project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Build multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Monitor an external job

This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

Copy existing Job

Copy from

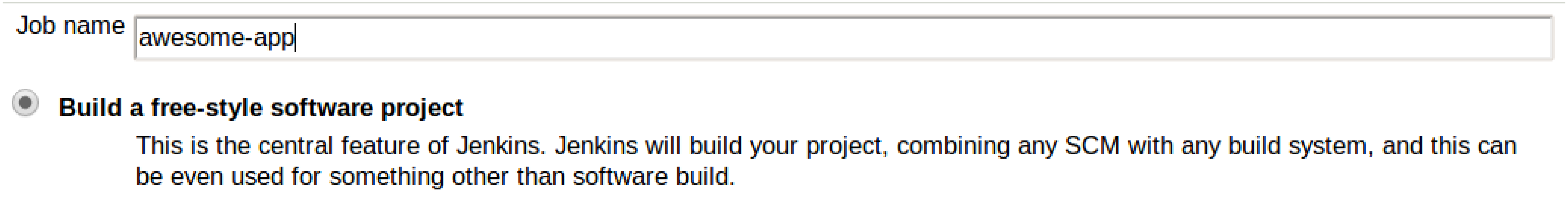
Types of Jobs

- Freestyle Software Project – General Purpose and Flexible Job
- Maven Project – Maven based project, easy setup using Apache Maven build tool
- Monitor External Job – Keep an eye on a non-interactive process
- Multiconfiguration ("Matrix") Job – Run the same job with different configurations, allows you to run jobs in different OSes, VMs, Libraries
- Copy an existing Job – Copy a pre-existing job so that you don't have to fill in all the values over again

Job Check List

- Make sure that there is enough disk space
- Have adequate memory
- Decide how many old jobs you wish to keep
- Consider backup strategy for old jobs

Creating Freestyle Build Job



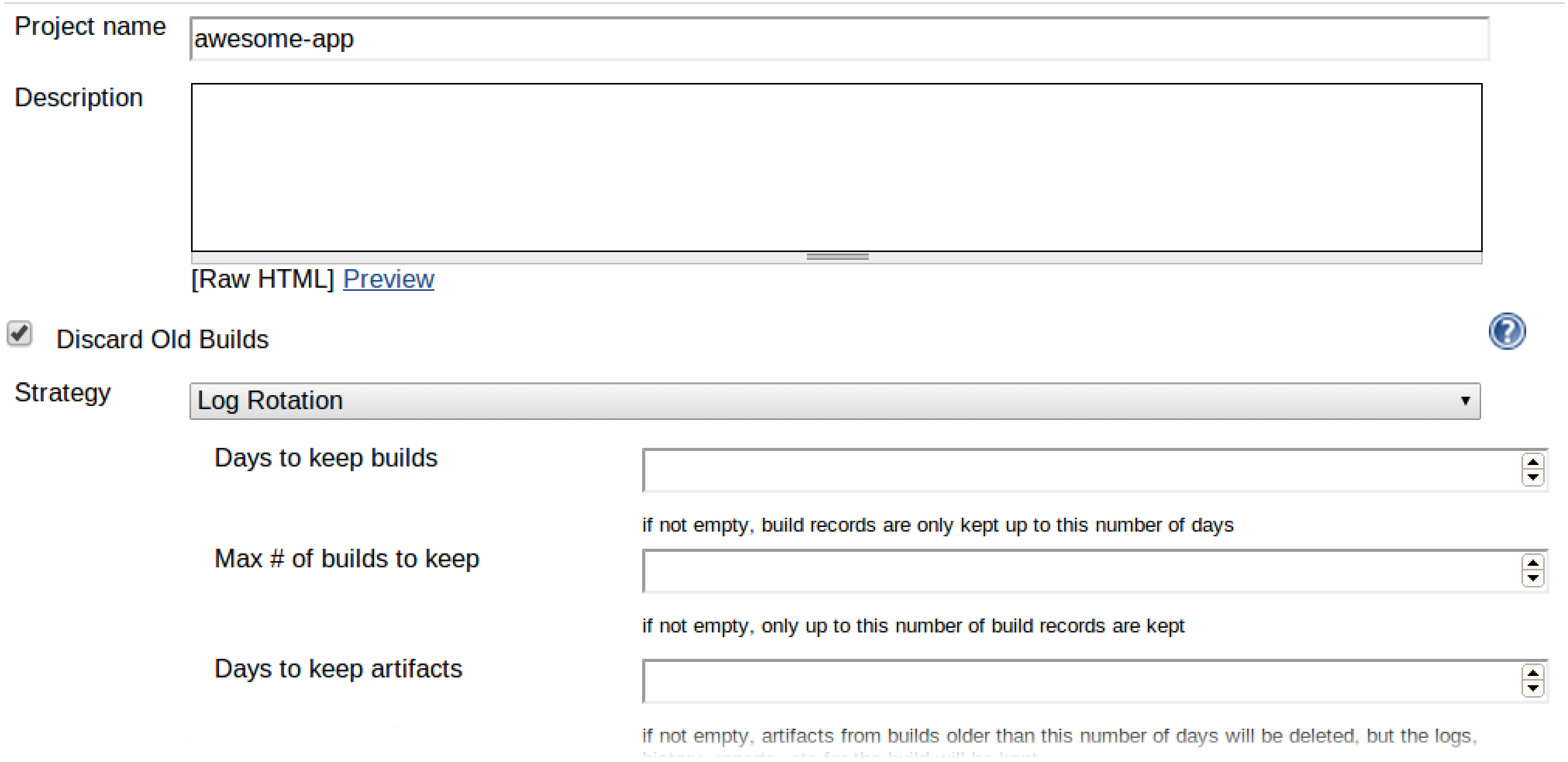
Job name: awesome-app

● **Build a free-style software project**

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

- Enter the job-name
- Job-names convert to directories so no spaces or other strange characters
- Click OK to configure your new freestyle job

Configuring Freestyle Jobs



Project name	awesome-app
Description	

[Raw HTML] Preview

[✓] Discard Old Builds

Strategy	Log Rotation
Days to keep builds	
	if not empty, build records are only kept up to this number of days
Max # of builds to keep	
	if not empty, only up to this number of build records are kept
Days to keep artifacts	
	if not empty, artifacts from builds older than this number of days will be deleted, but the logs,

Configuring Freestyle Jobs

- Job Name – At this time if you want to change your mind on the name, do it.
- Job Description – A description of the job and reason for the build and in what context will the build be taken place
- Discard Old Builds – Limit the number of builds in the build history by specifying:
  - Days to keep builds
  - Max number of builds to keep
  - How long to keep the artifacts (Advanced)

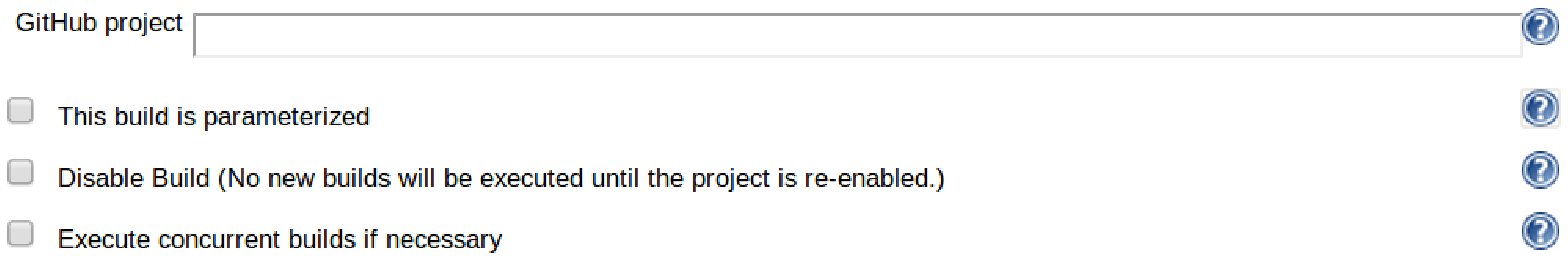
Sentimental Jenkins

- Jenkins will never get rid of the last <u>stable</u> build
- If Discard old builds is set to 10 days, and the last stable build is 20 days old, it will still be kept

Parameterized Jobs

- If you wish to run fast unit tests for example, setting up a parameter
- Fully customized
- More about Parameterized Jobs in the Advanced Section

More Freestyle Configuration

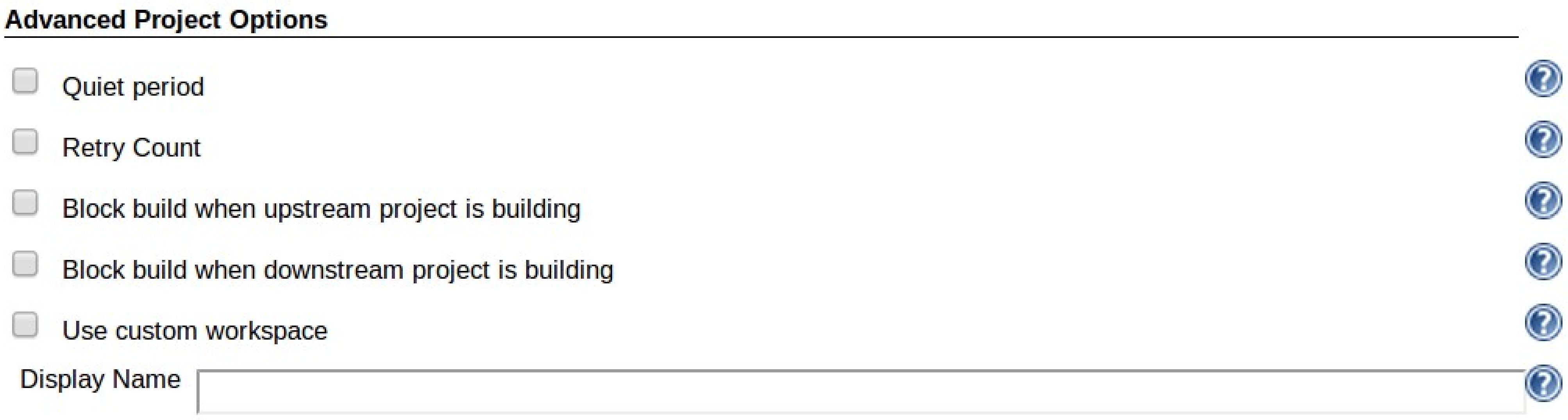

GitHub project

- [ ] This build is parameterized
- [ ] Disable Build (No new builds will be executed until the project is re-enabled.)
- [ ] Execute concurrent builds if necessary

More Freestyle Configuration

- Github Project – The repository of the project if it is housed on github, leave blank if not.
- Parameterized Jobs - Allow you to control which jobs run based on parameters
- Disable Build – Used for major refactoring so as not to plague the dev team with failures. Not useful for new jobs
- Execute Concurrent Builds – Every build get executed in a separate workspace and on a different executor

Advanced Project Options



Advanced Project Options

- ☐ Quiet period
- ☐ Retry Count
- ☐ Block build when upstream project is building
- ☐ Block build when downstream project is building
- ☐ Use custom workspace

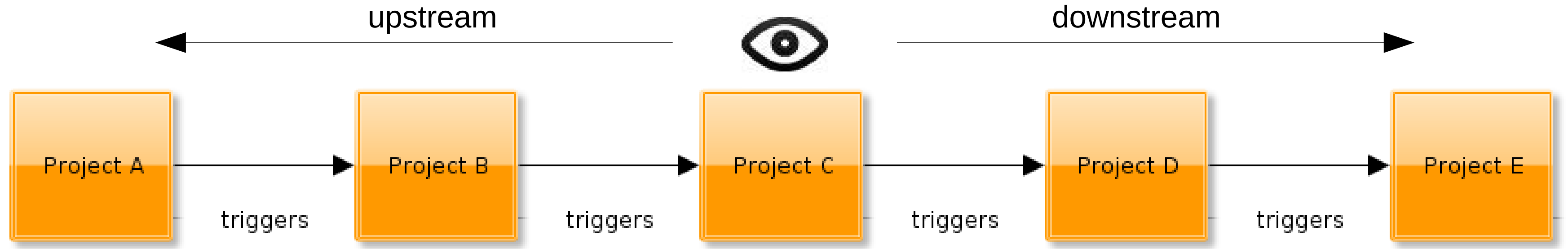
Display Name

Quiet Period & Retry Count

- Quiet Period - Override the system quiet period defined in the Jenkins
- Retry Count - If a build fails to checkout from the repository, Jenkins will retry the specified number of times before giving up

Upstream and Downstream?

Upstream? Downstream?



Blocking Builds Upstream

- Jenkins will prevent the project from building when a dependency of this project is in the queue, or building.
- The dependencies include the direct as well as the transitive dependencies.

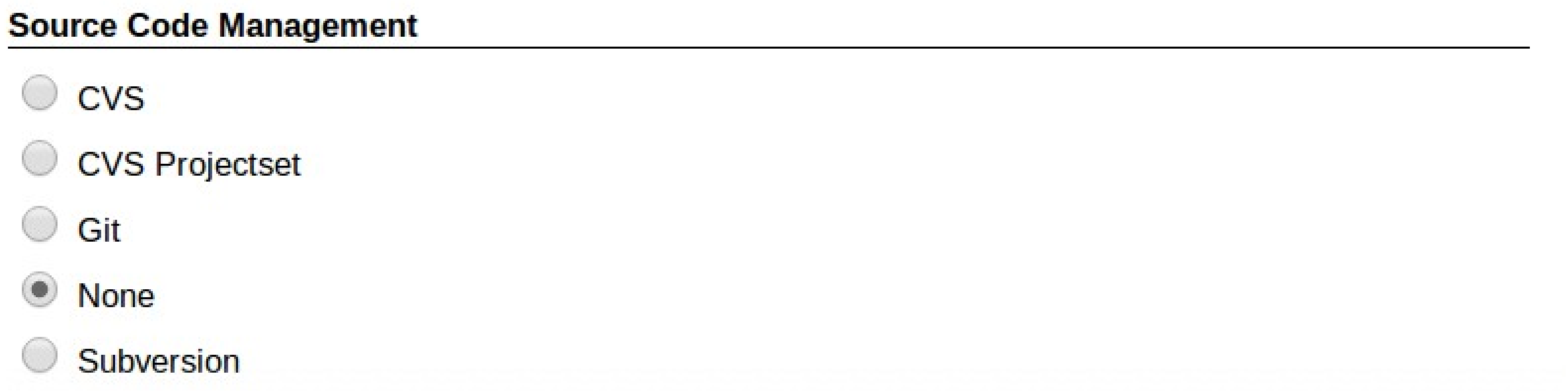
Blocking Builds Downstream

- Jenkins will prevent the project from building when a child of this project is in the queue, or building.
- The dependencies include the direct as well as the transitive dependencies.

Use Custom Workspace

- For any extraneous circumstances where a different workspace is required for a build.
- Reasons:
  - You do not want to share workspace so that you can analyze code and test results in "clean room" environment
  - Build is required to be built in a certain directory to perform function level testing
  - Need in a directory for an operating system procedure like having it read from a `cron`

Source Code Management



Source Code Management

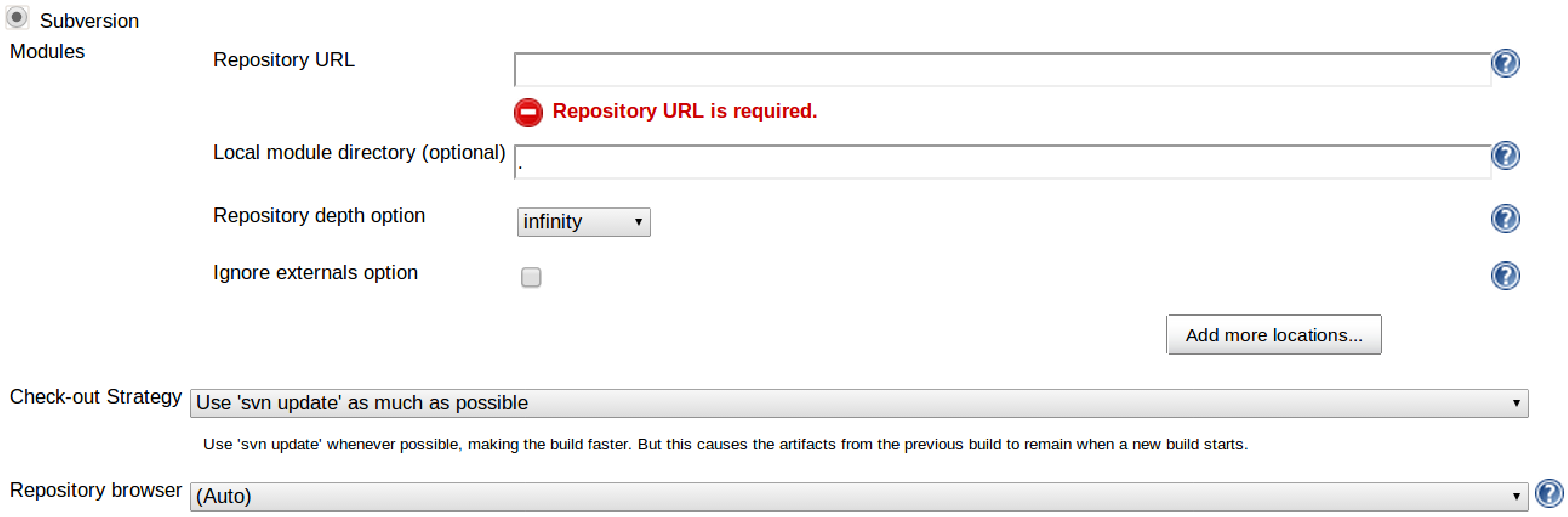
- ○ CVS
- ○ CVS Projectset
- ○ Git
- ● None
- ○ Subversion

Source Code Management

- The point of Jenkins is to monitor source control
- Jenkins will then compile and test the most recent change
- Jenkins supports a wide range of version control systems (VCS)

Configuring Subversion

Configuring Subversion



● Subversion

Modules

Repository URL	
	Repository URL is required.
Local module directory (optional)	.
Repository depth option	infinity
Ignore externals option	☐

Check-out Strategy: Use 'svn update' as much as possible

Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Repository browser: (Auto)

Configuring Subversion

- Repository URL – URL Location of Subversion. Subversion Version Control is accessed using an address
- Local Module Directory –
  - Relative directory inside of the workspace where the repository will be checked out.
  - A period can be used to represent the workspace directory

Subversion Depth

- Depth is how you can tell Subversion how far deep do you wish to perform a process.

```
$ svn checkout file:///var/svn/repos mom
A    mom/son
A    mom/son/grandson
A    mom/daughter
A    mom/daughter/granddaughter1
A    mom/daughter/granddaughter1/bunny1.txt
A    mom/daughter/granddaughter1/bunny2.txt
A    mom/daughter/granddaughter2
A    mom/daughter/fishie.txt
A    mom/kitty1.txt
A    mom/doggie1.txt
Checked out revision 1.
```

```
$ svn checkout file:///var/svn/repos mom-empty
--depth empty
Checked out revision 1
```

Source: http://svnbook.red-bean.com/en/1.7/svn-book.html

Subversion Depth Values

- "empty" includes only the immediate target of the operation, not any of its file or directory children.
- "files" includes the immediate target of the operation and any of its immediate file children.
- "immediates" includes the immediate target of the operation and any of its immediate file or directory children. The directory children will themselves be empty.
- "infinity" (Default) includes the immediate target, its file and directory children, its children's children, and so on to full recursion.

Source: http://svnbook.red-bean.com/en/1.7/svn-book.html

Subversion Externals

- Subversion maps an external "project" inside of the current checkout.
- All externals are located in the `svn:externals` property
- Ignore Externals Option – ignores any processing of externals

Subversion Externals Example

```
$ svn propget svn:externals calc
third-party/sounds             http://svn.example.com/repos/sounds
third-party/skins -r148        http://svn.example.com/skinproj
third-party/skins/toolkit -r21 http://svn.example.com/skin-maker
```

```
$ svn checkout http://svn.example.com/repos/calc
A    calc
A    calc/Makefile
A    calc/integer.c
A    calc/button.c
Checked out revision 148.
Fetching external item into calc/third-party/sounds
A    calc/third-party/sounds/ding.ogg
A    calc/third-party/sounds/dong.ogg
A    calc/third-party/sounds/clang.ogg
…
A    calc/third-party/sounds/bang.ogg
A    calc/third-party/sounds/twang.ogg
Checked out revision 14.
Fetching external item into calc/third-party/skins
```

Source: http://svnbook.red-bean.com/en/1.7/svn.advanced.externals.html

Subversion Check Out Strategy

- 'Svn Update' as much as possible - Fastest option, but leaves artifacts from previous build
- 'Always checkout a fresh copy' – Slowest option, Erases and downloads the entire repository
- 'Emulate clean checkout by first deleting unversioned/ignored files, then 'svn update'' – Guts needless files before updating
- 'Use svn update as much as possible, with 'svn revert' before update' – Guts needless files by performing revert then update.

Repository Browser

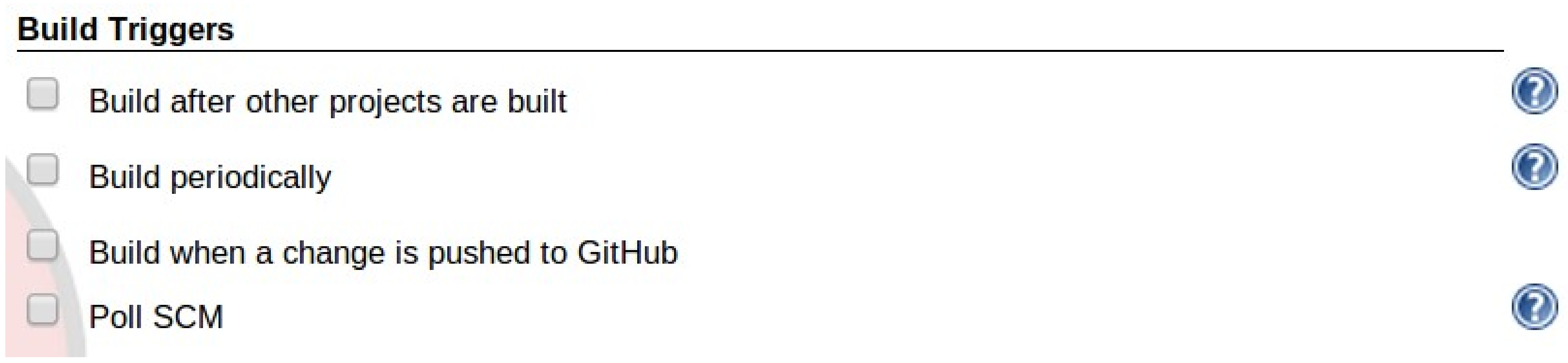
- Subversion usually has other specialized aesthetic browsers to view source code and changes
- "Auto" selection attempts to infer whatever repository browser from other created jobs
- WebSvn is the standard repository browser

Lab: Setup A Subversion Instance

- Create a free-style project named `simple-app`
- Set Subversion Repository URL to `http://http://50.112.49.6/svn/simple-msbuild-project/` or `http://50.112.49.6/svn/simple-mvn-project/` or `http://50.112.49.6/svn/simple-ant-project/`
- Meanwhile, in a separate location `svn co http://50.112.49.6/svn/simple-msbuild-project/` or `http://50.112.49.6/svn/simple-mvn-project/` or `http://50.112.49.6/svn/simple-ant-project/` in another folder to change development

Build Triggers

Build Triggers



Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] Build when a change is pushed to GitHub
- [ ] Poll SCM

Build Triggers

- Establishes how to start the build
- Four Triggers Available
  - Build after other projects are built
  - Build periodically
  - Build when a change is pushed to Github (Git only)
  - Poll the SCM
- Leave it blank if you wish to build manually

Build after other projects are built

- Easiest way to create a build pipeline
- Can contain one or more jobs, delimited by a comma
- Effective Strategy would be to perform longer difficult tests as part of another job

Build Periodically

- Cron replacement to execute the project
- Not ideal for Continuous Integration
- Uses cron scheduling to process external job (More on cron timing next)

Build When a Change is pushed to Github

- If the SCM was given 'Github' as a choice with the address to the repository
- Triggers on the last commit

Poll SCM

- Preferred method for Continuous Integration
- Uses Cron-Like Scheduling to set when Jenkins will poll the SCM and detected changes
- If Changes are discovered on the SCM, then Jenkins will build the project according to 'Build' (next section)

Cron Scheduling

- Preferred method for Continuous Integration
- Uses Cron Scheduling to set when Jenkins will poll the SCM and detected changes
- If Changes are discovered on the SCM, then Jenkins will build the project according to 'Build' (next section)

Cron Scheduling Syntax

- Scheduling has 5 field separated by whitespace
- MINUTE HOUR DOM MONTH DOW
- MINUTE – `0 - 59`
- HOUR – `0 - 23`
- DOM – `1 - 31`
- MONTH – `1 - 12`
- DOW – `0 - 6` (`0` is Sunday)

Cron Syntax Tricks

- Asterisk (*) represents every value
- Dashes (-) can represent range
- Slashes (/) represent a step
  - e.g `*/10` in the MINUTE represents every 10 minutes
- Comma (,) separated list for noncontinuous values
  - e.g. `15, 35, 50` in the MINUTE represents 15 minutes, 35 minutes, and 50 minutes
- Shorthand Values - `@yearly @annually @monthly @weekly @daily @midnight @hourly`

Ignore Post Commit Hooks

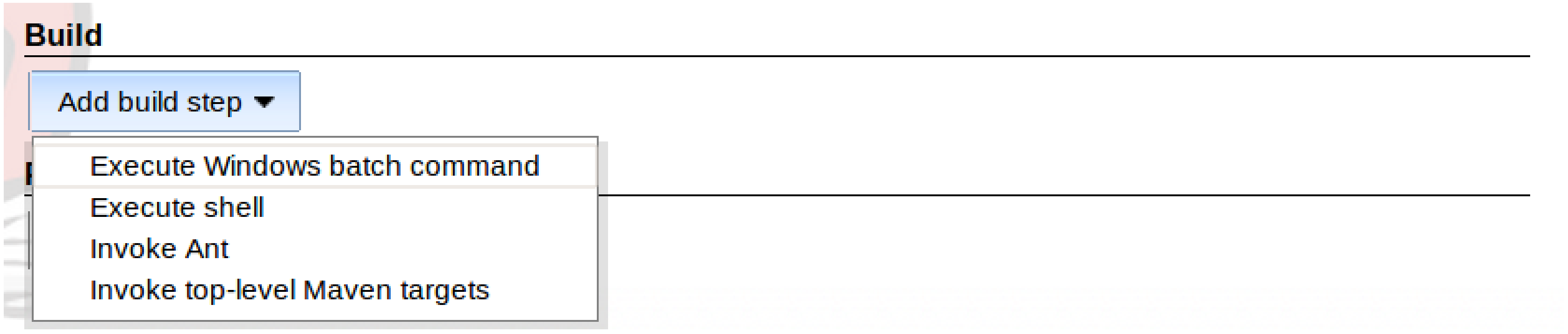
- Ignores Post Commit Hooks
- Post commit hooks are scripts that tell the SCM to tickle Jenkins or other CI servers to build
- Based on the plugin used

Lab: Build Triggers

- Poll the SCM for every minute on job `simplejob`
- `* * * * * = every minute`
- `H/5 * * * * = every 5 minutes`

Building

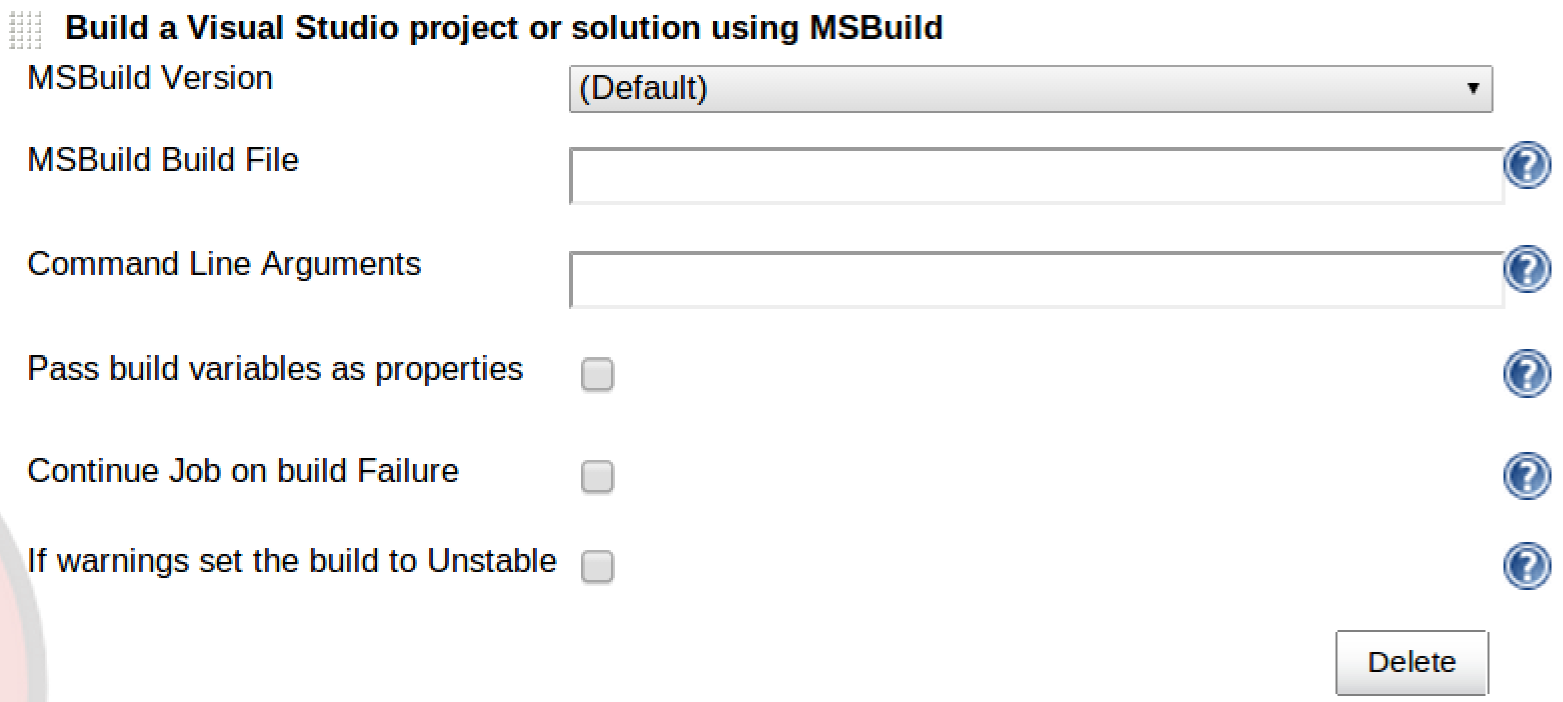
Building


Build
Add build step
Execute Windows batch command
Execute shell
Invoke Ant
Invoke top-level Maven targets

Building

- Specifies how to build the project, after Jenkins finds a change to the SCM
- Click on 'Add Build Step' to add the build type of your choice
- If your preferred build tool is not found, a plugin may need to be installed
- All builds can be deleted using 'Delete' next to each build section

Build a Visual Studio Project



Build a Visual Studio project or solution using MSBuild

MSBuild Version	(Default)
MSBuild Build File	
Command Line Arguments	
Pass build variables as properties	☐
Continue Job on build Failure	☐
If warnings set the build to Unstable	☐

Build a Visual Studio Project

- Use MSBuild to build .NET projects.
- First configure MSBuild.exe on Jenkin's configuration page.
  - Usually situated in a subfolder of `C:\WINDOWS\Microsoft.NET\Framework`
- If you have multiple MSBuild versions installed, you can configure multiple executables.

Build a Visual Studio Project

- MSBuild Version – What Version of MSBuild is required
- MSBuild File – What MSBuild file should run (`.proj` or `.sln` file)
- Command Line Arguments – Any arguments required for the run
- Pass Build Variables As Properties – Checkbox to Pass Jenkins Build Variables as Properties

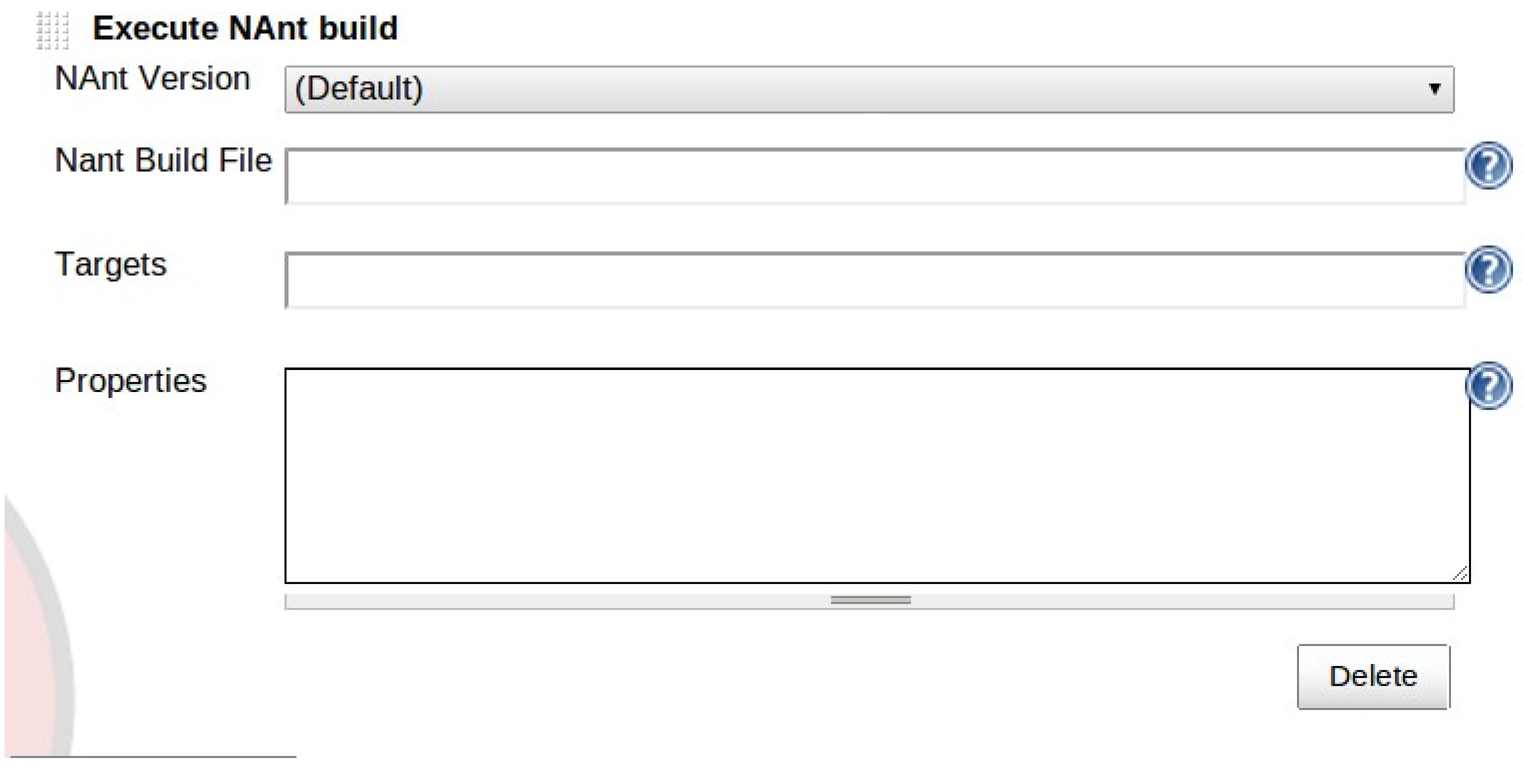
Build a Visual Studio Project (Advanced)

- Continue Job On Build Failure – If Build fails continue with the rest of the job
- If Warnings Set the Build to Unstable – Causes the build to become unstable if any warnings were posted from the build

Lab: Install MSBuild Plugin

- Install MSBuild Plugin
- Create an MSBuild version in 'Manage Jenkins' pointing to current version of MSBuild (e.g. MSBuild-4.0)
- Create more MSBuilds if you have them and which to compile with different versions

Build a NAnt Project



Execute NAnt build

NAnt Version	(Default)
Nant Build File	
Targets	
Properties	

Build a NAnt Project

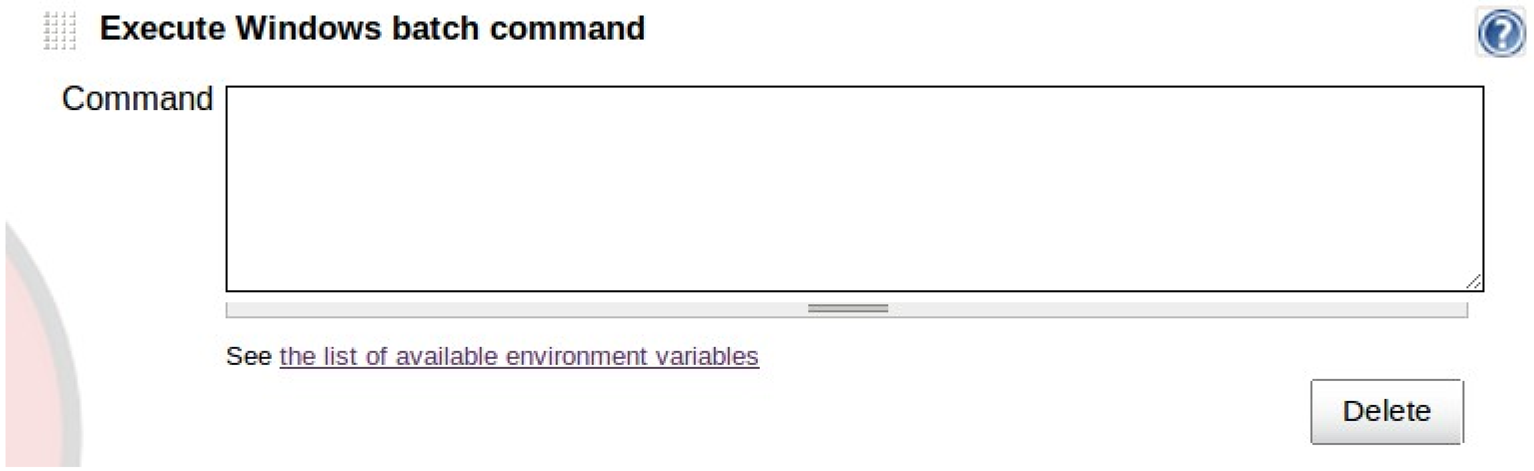
- NAnt is analog to the Popular Ant Tool on the JVM
- NAnt contains targets that contains tasks
- NAnt will invoke the targets directly
- First configure NAnt installation directory
- If you have multiple NAnt versions installed, you can configure multiple executables.

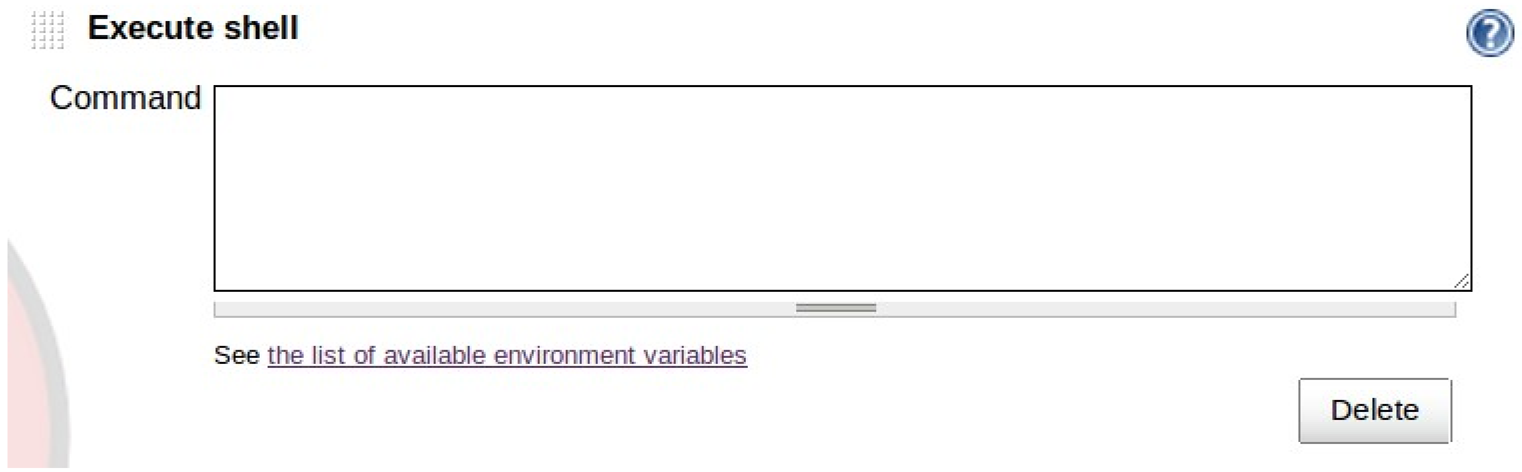
Build a NAnt Project

- NAnt Version – Version of the NAnt to run the build, default will run the latest
- Nant Build File – Location of the `.build` file to build the application
- Targets – List of targets that the build will execute
- Properties (Advanced) -Define properties that will be passed to the NAnt build (one property per line)

```
user.name = Abe Lincoln
user.email = abe@logcabin.org
```

Windows Batch Or Shell Command


Execute Windows batch command
Command
See the list of available environment variables
Delete



Windows Batch Or Shell Command

- Execute any OS Specific Command
- Script location is relative to the Workspace directory
- Build is specific to the OS and is brittle

Environment Variables

- Jenkins provides a set of environment variables that can be used and reference in your build scripts
- For a full reference see: `http://<jenkins-address>:<port>/env-vars.html`

Lab: Setup Job and Submit

- View the current environment variable on your Jenkins instance
- Add an 'MSBuild' Build to `simpleproject`
- Set the 'MSBuild' to use the version that you specified
- Set the 'MSBuild' file to `main-app.sln`
- Add the configuration parameter: `/p:configuration=release`
- Save the Job and run it manually one time only

Lab 2: Use Windows Batch to Test

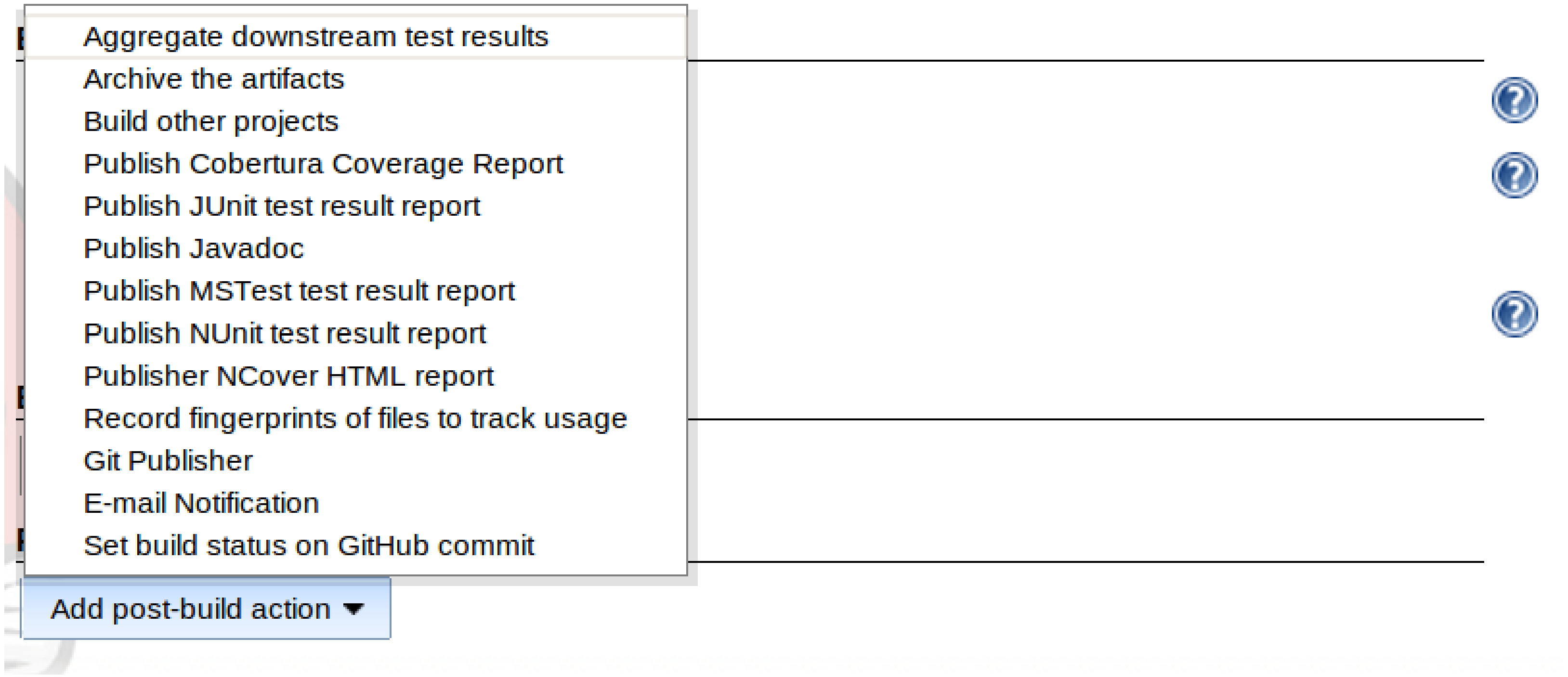
- To test in a Microsoft Environment use the MSTest.exe to test the project.
- MSTest.exe is found on different installations

```
rmdir /s /q TestResults
mkdir TestResults

"C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\MSTest.exe"
   /testcontainer:main-app-tests\bin\debug\main-app-tests.dll
   /category:"unit"
   /resultsfile:TestResults\testResults.trx
```

Post Build Actions

Post Build Actions


Aggregate downstream test results
Archive the artifacts
Build other projects
Publish Cobertura Coverage Report
Publish JUnit test result report
Publish Javadoc
Publish MSTest test result report
Publish NUnit test result report
Publisher NCover HTML report
Record fingerprints of files to track usage
Git Publisher
E-mail Notification
Set build status on GitHub commit
Add post-build action

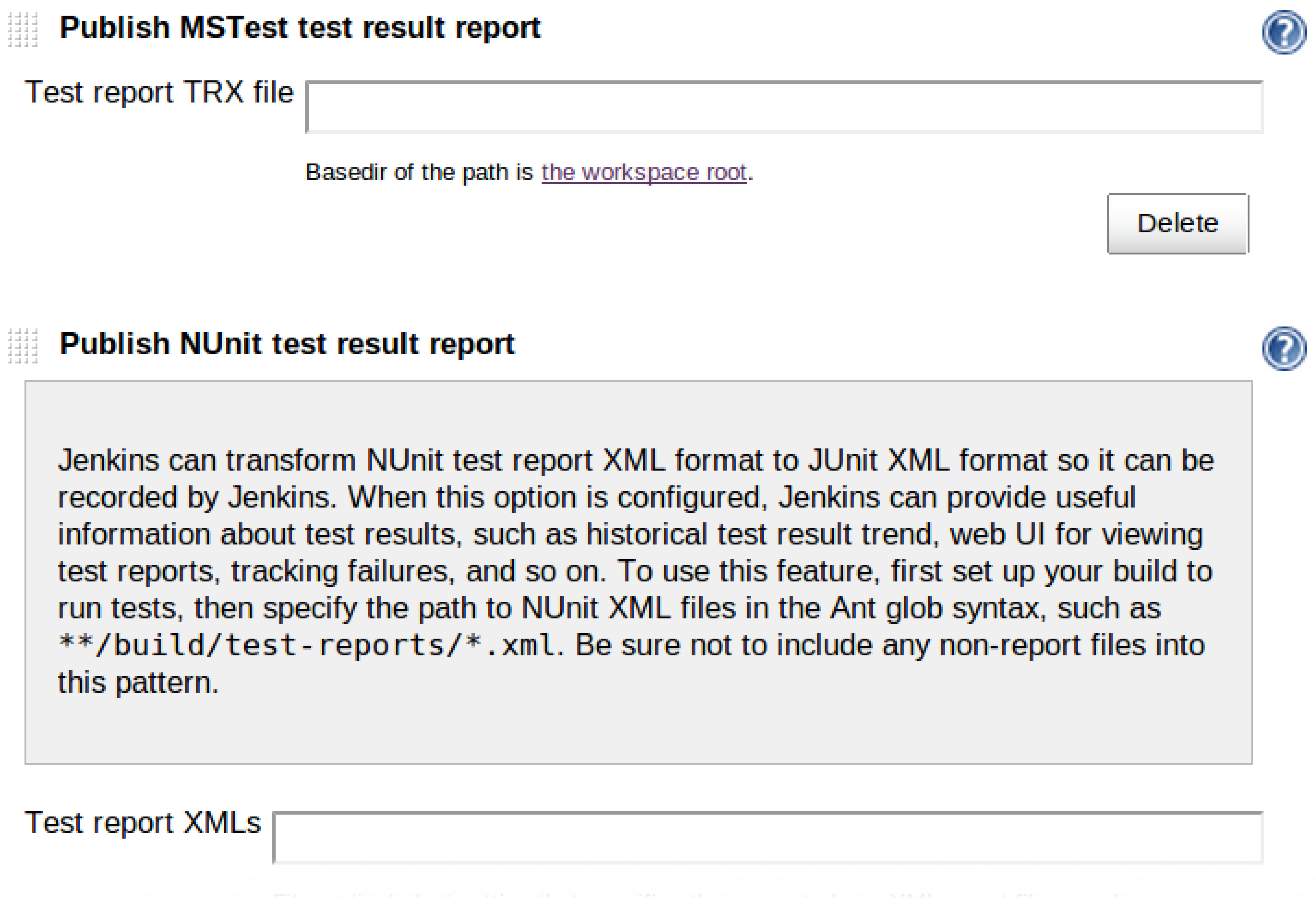
Post Build Actions

- Takes place after the build
- Opportunity to:
  - archive artifacts
  - report and aggregate test results
  - notify people about the results
  - record fingerprints
  - more

Reporting Test Results

- Reporting Test Results
- Support for Various Testing Tools
- All testing results should be in the format of a JUnit XML format.
- MSTest Test Result requires the TestReport TRX File to show results
- NUnit Test Result requires NUnit XML test result files to show results

MSTest and NUnit Setup



Publish MSTest test result report

Test report TRX file

Basedir of the path is the workspace root.

Publish NUnit test result report

Jenkins can transform NUnit test report XML format to JUnit XML format so it can be recorded by Jenkins. When this option is configured, Jenkins can provide useful information about test results, such as historical test result trend, web UI for viewing test reports, tracking failures, and so on. To use this feature, first set up your build to run tests, then specify the path to NUnit XML files in the Ant glob syntax, such as `**/build/test-reports/*.xml`. Be sure not to include any non-report files into this pattern.

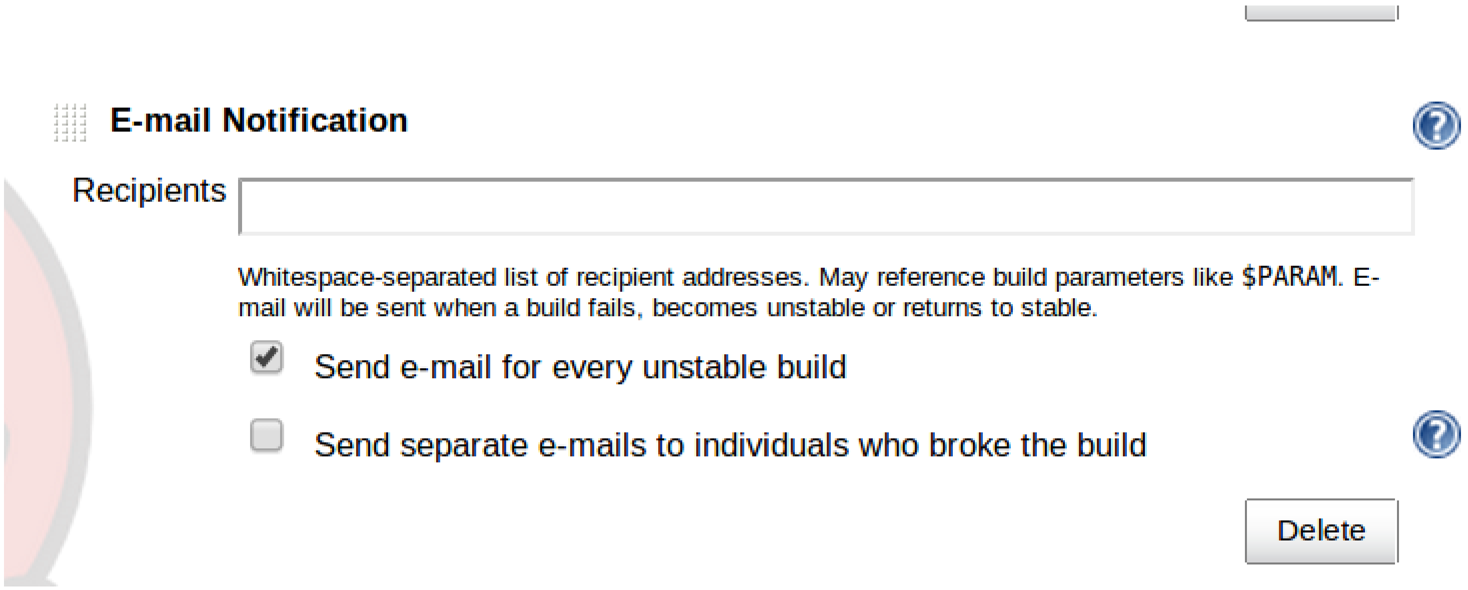
Test report XMLs

Lab: Set up a Test Result Post Build

- Install MSTest Plugin
- Go to job configuration and put in a TRX file:

```
TestResults\testResults.trx
```

Setting up an Email Notification



E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like `$PARAM`. E-mail will be sent when a build fails, becomes unstable or returns to stable.

- [✓] Send e-mail for every unstable build
- [ ] Send separate e-mails to individuals who broke the build

Setting up an Email notification

- Notifies all recipients given that a job on the following conditions:
  - Every failed build triggers a new e-mail.
  - A successful build after a failed (or unstable) build triggers a new e-mail, indicating that a crisis is over.
  - An unstable build after a successful build triggers a new e-mail, indicating that there's a regression.
  - Unless configured, every unstable build triggers a new e-mail, indicating that regression is still there.

Setting up an Email notification

- Set the recipients
- If Security is setup a separate email can be sent to the person that broke the build
- By default it will only send out an email for the first failed build but not subsequent builds

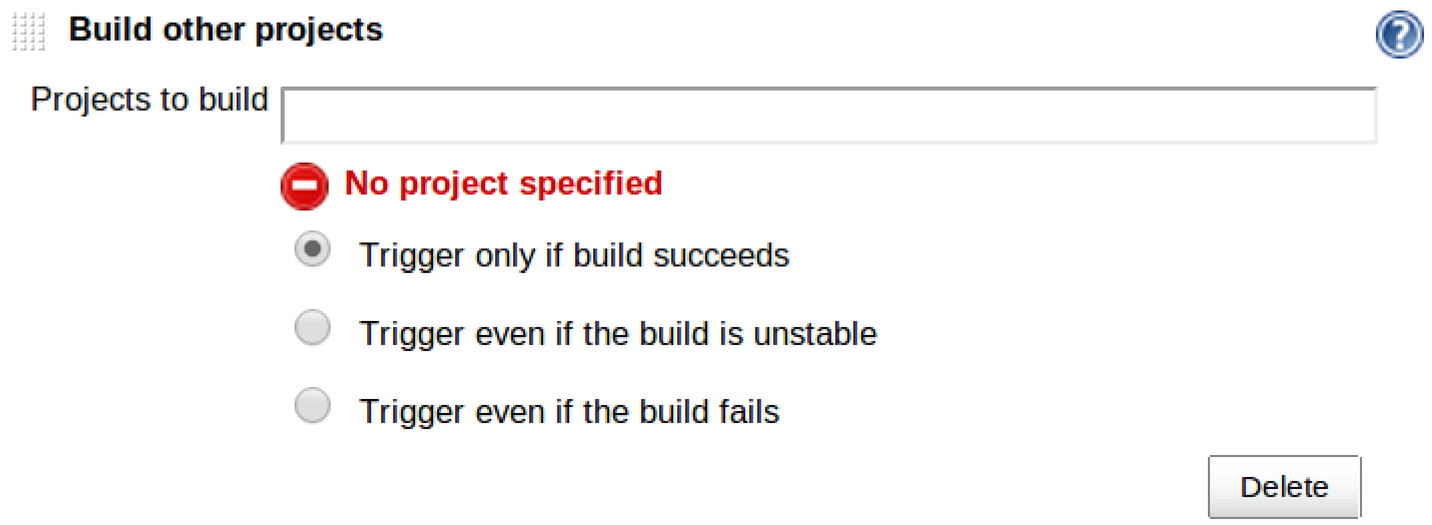
Lab: Run Testing

- Set up an Email Notification to notify some of the other members in our group
- Run a manual build to ensure that it works and view the results
- Have other commit code to the project and test the automatic builds

Build Other Jobs

- Will build other jobs on Jenkins
- Default is it will run if the current job builds successfully and is stable
- It is configurable to build other jobs if it is unstable
- Perfect for reporting only after the test has succeeded

Build Other Jobs


Build other projects

Projects to build

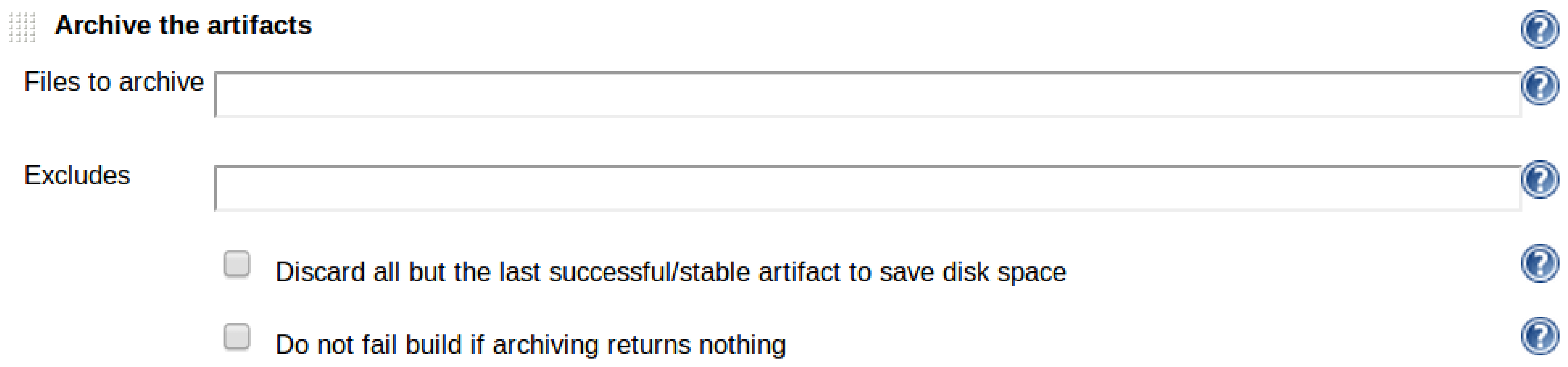
No project specified

(●) Trigger only if build succeeds

( ) Trigger even if the build is unstable

( ) Trigger even if the build fails

Archiving Artifacts



Archive the artifacts

Files to archive

Excludes

☐ Discard all but the last successful/stable artifact to save disk space

☐ Do not fail build if archiving returns nothing

Archiving Artifacts

- An artifact can be:
  - binary executable (.jar, .exe)
  - deployable artifact (.jar, .ear, .war)
  - documentation archives
  - source code archives
- Continuous Building

Archiving Artifacts

- "Files to archive field" can place the full path of the files wanting to archive
- Assumes from `workspace` directory
- You can use ant wild cards
  - `**` = recursive directory search
  - `*` = wildcard
- Comma separated for different groups

Advanced Archiving Artifacts

- "Excludes" contains patterns of files to exclude
- "Discard all but the last successful/stable artifact to save disk space" – to save space, you can discard all archives except the last stable one
- "Do not fail build if archiving returns nothing" – If an archive cannot be found, do not fail the build

Obtaining Artifacts by URL

- Three fixed URLs are available to retreive
  - the last stable build – built successfully, but failed a test
  - the last successful build - built and tested successfully
  - the last completed build – a finished build regardless of test result

Obtaining Artifacts by URL

- The last stable build
  - `<server-url>/job/<build-job>/lastStableBuild/artifact/<path-to-artifact>`
- The last successful build
  - `<server-url>/job/<build-job>/lastSuccessfulBuild/artifact/<path-to-artifact>`
- The last completed build
  - `<server-url>/job/<build-job>/lastCompletedBuild/artifact/<path-to-artifact>`

Lab: Artifacts

- Set up Archive Artifacts to show all `.exe` and `.dll` files
- Select "Do not fail build if archiving returns nothing"
- Retrieve the last stable `Project1.exe` file using a URL

Thanks