



Getting started with Jenkins





Course Agenda

- ◇ Understanding Jenkins
- ◇ Getting jenkins up and running
- ◇ Plugins
- ◇ The big picture

Understanding jenkins

- ◇ Why jenkins
- ◇ History of jenkins
- ◇ Continuous integration
- ◇ Installing and running jenkins
- ◇ Initial setup and the data directory
- ◇ Default security
- ◇ Anonymous read access
- ◇ Running jenkins with docker

Why Jenkins

- ◇ CRON on steroids ?
- ◇ Open source
- ◇  Can collect feedback
- ◇ Graphical interface
- ◇ Large plugin ecosystem
- ◇ Doesn't need a lot of coding or configuration
- ◇ Automate regular tasks
- ◇  Automate entire deployment flow
- ◇ Very reliable
- ◇ Increases confidence in our build and deployment flow
- ◇ Immediate feedback

History of jenkins

- ◇ 2001 - Cruisecontrol
 - Mostly for java apps but also .NET
 - Needed manual xml configurations to set up jobs
- ◇ 2004 - Hudson - Sun Microsystem
 - Written Kohuske Kawaguchi
 - 2009 - Oracle acquires sun
- ◇ 2010 - Trademark dispute causes the community to fork hudson to Jenkins
- ◇ 2014 - Kohuske joins cloudbees as CTO
- ◇ 2016 - Jenkins 2 is released

Continuous integration

- ◇ Integration is painful
- ◇ Different devs working on multiple features might do large changes in code which aren't tested together
- ◇ Continuous integration allows us to combine and check work from multiple developers on a regular basis
- 🗨️ Jenkins is the tool which allows us to automate the process of building and testing the code and provide immediate feedback to the results of the integration

Installing and running Jenkins

- ◇ Download jenkins
 - <https://jenkins.io>
 - LTS vs Weekly
- ◇ Can be downloaded as:
 - War
 - Service for multiple platforms
 - Docker container
- ◇ `$ docker pull jenkins`



`$ docker run -p 8080:8080 -d jenkins`

- ◇ Get password :
 - `$ docker logs <Jenkins docker id>`

- ◇ Jenkins home is located at:



- `/var/jenkins_home`
- ◇ All configuration and plugin files saved here so we don't need a database

Default security

- ◇ Create an admin user for yourself
- ◇ This will be the user we'll be using from here on
- ◇ By default all logged in users can do anything
- ◇ To setup additional authentication and authorization methods go to "Manage Jenkins" -> "Configure global security"
- ◇ In production we need to setup additional security methods to limit what users can do
- ◇ To add users to the internal Jenkins DB go to "Manage users" -> "Create user"

Anonymous read access

- ◇ By allowing anonymous read access we allow users to view the status of jobs without being logged in
- ◇ This can be convenient when using jenkins internally to display information

Running Jenkins in Docker

- ◇ Need to setup Volume to persist data across docker container runs
- ◇ Very easy to setup jenkins inside docker and ideal for testing different versions of plugins or even jenkins

Creating build applications

- ◇ Anatomy of a build
- ◇ Cloning the sample project
- ◇ Manual compilation with Maven
- ◇ Manually testing, Packaging and running the app
- ◇ Creating a jenkins job and configuring a Git repo
- ◇ Compiling in jenkins
- ◇ Peeking into the jenkins workspace
- ◇ Browsing the workspace in jenkins
- ◇ App packaging in jenkins
- ◇ Archiving artifacts
- ◇ Cleaning up past builds
- ◇ Build time trend
- ◇ The Jenkins dashboard
- ◇ Troubleshooting build failures
- ◇ importing job config.xml files
- ◇ Anatomy of the job

Anatomy of a build

- ◇ Clone the test project from git
- ◇ After cloning we'll manually test the project to ensure we can:
 - Compile
 - Test
 - Package
- ◇ After manually cloning we'll migrate the process into jenkins

Cloning the sample project

-
- ◇ \$ git clone <https://github.com/omrisiri/spring-boot.git>
 - \$ cd spring-boot
 - \$ cd spring-boot-samples/spring-boot-sample-atmosphere/

Manual compilation with Maven

- ◇ pom file defines how to do the build, test and package the code
- ◇ Need to have Maven and java installed
 - yum install maven
 - apt-get install maven
- ◇ Command line tool is mvn
- ◇ \$ mvn compile

Manually testing, Packaging and running the app

-
- ◇ \$ mvn test
 - Runs all tests and shows results
 - ◇ \$ mvn package
 - Packages the project into Jar or War file
 - This should create a jar file we will use during the examples here called:
 - **target/spring-boot-sample-atmosphere-1.4.0.BUILD-SNAPSHOT.jar**

Creating a jenkins job and configuring a Git repo

- ◇ Now we'll migrate the command line to jenkins
- ◇ Create a new job (also called item)
- ◇ Give it a name and select the type of job
 - We'll start with a freestyle job
- ◇ Now we'll configure the git repo in the source code management tab
 - Copy it from the link previously given

Compiling in jenkins

- ◇ Since our example is a MVN job we'll select "Invoke top-level maven targets"
 - Write the targets we've previously run:
 - Compile
 - If the POM is not in the TLD we need to tell jenkins where to find the pom in the "Advanced" section
- ◇ After saving we can now build the application
 - "Build now" is the manual way of running a build
- ◇ Now lets explore the build log
 - Select "console output" on the left dashboard
 - Notice the workspace we're working in

Jenkins workspace

- ◇ Workspace is the location where are the files are located during build
- ◇ Full path is in the logs
- ◇ Should contain all the build files and output artifacts

Jenkins workspace from jenkins

- ◇ Is available from the jenkins GUI
 - More convenient than sshing to machines
- ◇ Inside the build view
 - Select the workspace directory

App packaging in jenkins

- ◇ When using jenkins with maven we can also package the app inside the build
 - Change the maven target to package
- ◇ After building the job we'll see the artifact in the target directory

Archiving artifacts

- ◇ Jenkins allows us archive the artifacts to an external source such as artifactory
- ◇ Set up as a “post build action”
 - Select “Archive the artifact”
 - Add “target/*.jar” to the list of files to archive
- ◇ Run build again
- ◇ You’ll now see the artifacts in the build log

Cleaning up past build

- ◇ The build directories aren't cleaned after every build
- ◇ We need to ask Maven to clean the directory even if nothing has changed
- ◇ Add “Clean” target to the maven build targets
 - Will now contain “clean package”
- ◇ Can also be done externally using Jenkins delete workspace before build
 - This will cause the build to slow down

Build time trend

- ◇ Allows us to view the trend of the build time
- ◇ Can help us detect changes which affected the build time

The jenkins dashboard

- ◇ The main screen is also called the dashboard
 - Can be used to have a higher level view of jobs
 - Has shortcuts to build and change the jobs
 - We can see the status of the build

Troubleshooting build failures

- ◇ Looking at the console output and understanding where our build failed is essential
- ◇ Post build steps will still run on build failure unless otherwise told

Importing job config.xml files

- ◇ We can go directly to the jobs directory in jenkins and add a new directory to jenkins and copy config.xml file to it
- ◇ We need to tell jenkins to reload configuration from disk following this change.
- ◇ Go to: Manage jenkins -> Reload configuration from disk

Anatomy of the job

- ◇ A job is a definition of how a build should run
- ◇ Defined in a config.xml file
- ◇ Each job may contain many different builds

Finding and managing plugins

- ◇ The need for plugins
- ◇ Useful plugins overview

The need for plugins

- ◇ Jenkins installs some plugins out of the box but a lot of we might need to add additional plugins to extend functionality

Useful plugins

- ◇ Source control plugins
 - Git,svn,tfs etc
- ◇ Trigger plugins
 - Github pull request trigger
- ◇ Build tools
 - Copy artifacts to s3/artifactory
 - ant/mvn
 - Even powershell extensions
- ◇ Wrappers
 - Virtualbox, docker, ec2
 - Selenium
- ◇ Notifiers
 - Hipchat, slack, twitter
- ◇ Reporting
 - Findbugs
 - Static code analysis

Summary