Jenkins CI

A hands-on CI workshop

What is Continuous Integration/Delivery

- **Continuous integration** (**CI**) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day.
- **Continuous Delivery** (**CD**) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time. It is used in software development to automate and improve the process of software delivery.

(Wikipedia)

Why Jenkins?

- Ease of Use
- 1185 plugins
- Reporting
- Distributed builds

Jenkins in a Nutshell

- Open-source CI Server
- Originally written by Kohsuke Kawaguchi (as Hudson)
- Released in 2008
- Renamed to Jenkins in 2011
- 1185 plugins available on official site
- http://jenkins.io

Installing Jenkins (on a RedHat distro)

- https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Red+Hat+distributions

- sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo (for LTS versions)
- sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
- sudo yum install java-1.8.0-openjdk-devel
- sudo yum install jenkins

Installing Jenkins - additional options

- On an existing Tomcat web server:

To install Jenkins on Tomcat, simply copy `jenkins.war` to `$TOMCAT_HOME/webapps`, then access http://yourhost/jenkins

Tomcat Settings:

export CATALINA_OPTS="-DJENKINS_HOME=/path/to/jenkins_home/ -Xmx512m"

Installing Jenkins - additional options

- As a docker container:

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

- Or - with a data volume container:

```
docker create -v /var/jenkins_home --name jenkins-data jenkins
docker run -d -p 8080:8080 --volumes-from jenkins-dv jenkins
```

Starting Jenkins

- The old way:

  The easiest way to execute Jenkins is through the built in Jetty servlet container. You can execute Jenkins like this:

  $ java -jar jenkins.war

  Of course, you probably want to send the output of Jenkins to a log file, and if you're on Unix, you probably want to use nohup:

  $ nohup java -jar jenkins.war > $LOGFILE 2>&1

- Now simply:

  $ sudo systemctl start jenkins

- Jenkins init script:
  - *cat /etc/rc.d/init.d/jenkins*

Accessing Jenkins

- Jenkins runs on port 8080 by default.
- Open port 8080 in firewall:
  - firewall-cmd --zone=public --add-port=8080/tcp --permanent
  - firewall-cmd --zone=public --add-service=http --permanent
  - firewall-cmd --reload

- From host machine web browser:
  - http://<jenkins.host.ip>:8080

Administering Jenkins

- JENKINS_HOME directory:

```
+- config.xml      (jenkins root configuration)
+- *.xml           (other site-wide configuration files)
+- userContent     (files in this directory will be served under your http://server/userContent/)
+- fingerprints    (stores fingerprint records)
+- plugins         (stores plugins)
+- secrets (certificates)
+- updates (temporary storage for plugin and system updates)
+- jobs
   +- [JOBNAME]      (sub directory for each job)
      +- config.xml      (job configuration file)
      +- workspace       (working directory for the version control system)
      +- latest          (symbolic link to the last successful build)
      +- builds
         +- [BUILD_ID]      (for each build)
            +- build.xml       (build result summary)
            +- log             (log file)
            +- changelog.xml   (change log)
```

Administering Jenkins

- On Web UI:
  - Manage Jenkins - > Configure System

Securing Jenkins

- On Web UI:
  - Manage Jenkins - > Configure Global Security
    - Jenkins own user database
    - LDAP
    - Unix users
  - Authorization:
    - Project-based Matrix Authorization Strategy
    - Remember to make yourself admin before locking access.

Exercise

- Secure Jenkins with it's own user database
- Create user 'dummy'
- Create a job that user 'dummy' can build but not configure
- Change the permissions so that dummy can configure the job but not build it. (Does this make any sense?)

Managing Plugins

- Manage Jenkins -> Manage Plugins
- Install Plugins:
  - Parameterized Trigger Plugin
  - Rebuilder

Building with Jenkins

- New Item:
- We are presented with project type choice:
  - Freestyle Software Project
  - Pipeline Project
  - Maven project
  - External Project
  - Multi-configuration project
- Note: in Jenkins ‘Project’ = ‘Job’

Building with Jenkins

- Freestyle software project
  - Parameters
  - SCM
  - Triggers
  - Build
  - Post-build

Building with Jenkins

- Maven project
  - Parameters
  - SCM
  - Triggers
  - **Maven** Build
  - Post-build
    - Artefact Archiving

Before building Maven project

- Note: you'll need Maven installed on your Jenkins master
- back to the command shell:
  - sudo yum install maven
- or:
- Manage Jenkins->Configure System->Add Maven installation
- Check 'install automatically'

Maven project example

- Compile Jenkins:
- Create a Maven project
- SCM:
  - git : https://github.com/otomato-gh/mvnJunitExample.git
- Build Step:
  - Maven Goals and Options: ‘install’
- Save
- Build Now 

Examining the build job

- Build page
- Module list
- Progress/duration
- Console output

Polling SCM

- Exercise 1:
  - clone https://github.com/antweiss/jenkins-examples.git
  - add git credentials with private key from id_rsa file
  - create a Maven project to poll repository every 5 minutes
  - and run ‘mvn install’
  - archive artefact target/*.war

Adding Parameters

- Optionally changing the flow and type of build/test
- Built-in parameter types:
  - String
  - Boolean
  - Choice
  - Text
  - File
  - Run (a specific run of a Jenkins project)
  - Subversion tags
  - Credentials
  - Password

Exercise 2

- Create a freestyle job with 3 parameters
  - STRING1 (string), STRING2 (string), FLAG (boolean)
  - Add an ‘execute shell’ build step which creates a file with the name which equals STRING1 if FLAG is checked or STRING2 if FLAG is unchecked.

Email Notifications

- Setting up email
  - For gmail:
    - smtp.gmail.com
    - Use SMTP Authentication
    - Use SSL
    - Port : 465
    - Test configuration
    - Requires ‘less secure apps’ access on Google
    - Requires Jenkins admin email to be configured.

Email Notifications

- Setting up email
  - For Otomato SMTP server:
    - mail.otomato.link
    - Use SMTP Authentication (credentials provided in class)
    - Don’t use SSL
    - Port : 26
    - Test configuration
- Simple Email notifications
- Email-ext plugin

Exercise 3

- Send a test email to: ant.weiss@gmail.com
- Send a success email to: ant.weiss@gmail.com
- Send a failure email to ant.weiss@gmail.com

Creating Pipelines

- Simple build trigger
- Parameterized trigger plugin
- Build other projects

Visualizing pipelines

- Build Pipeline Plugin
- Delivery Pipeline Plugin

Distributed builds

- Adding a node
  - Manage Jenkins -> Manage Nodes - > New Node
- Defining ssh credentials
- Defining JDK

Exercise 4

- Create a maven project which builds Jenkins from source on the ‘slave’ node:
  - Job name ‘build-jenkins'
  - git url: https://github.com/jenkinsci/jenkins.git
  - Maven build:
    - root pom: pom.xml
    - Goals: -Plight-test install (add -DskipTests to skip tests)
- The result of maven build is 'jenkins.war' - find it in build workspace (war/target/jenkins.war)
  - browse the workspace through Jenkins web UI
- add Post-build Action:
- Trigger Parameterized Build on project 'execute-jenkins'
  - with Predefined Parameters:
    - WAR_PATH=${WORKSPACE}/war/target

Exercise 4 - continued

- Create a freestyle project
  - Job name : execute-jenkins
  - With parameter of type 'String', named 'WAR_PATH'
  - Add a build step 'execute shell':

    export BUILD_ID=dontkill

    echo JENKINS_IP=ip a | awk '/192/ {print $2}' | cut -d/ -f1 > ip.properties

    nohup java -jar ${WAR_PATH}/jenkins.war > my.out &
- add a Post-build Step:
- Trigger Parameterized Build on project 'test-jenkins'
  - with Predefined Parameters:
    - JENKINS_IP=<your_slave_ip>

Exercise 4 - continued

- Disable firewall on the slave node
- Create a third freestyle project
  - Job name : test-jenkins (set to run on master)
  - With parameter of type 'String', named ‘JENKINS_IP’
  - Add a build step: execute shell

    curl http://<USE_JENKINS_IP_VARIABLE>:8080/api/xml > out.xml
  - archive the resulting xml file in Jenkins master
    - (Use post-build step : Archive artefacts)

Aborting builds

- Just click on ‘x’

 http://otomato.link

Build timeout

- Install ‘Build Timeout’ Plugin
  - After installing the plugin, go to the configure page for your job and select "**Abort the build if it's stuck**".
  - From plugin documentation:

> Because Java only allows threads to be interrupted at a set of fixed locations, depending on how a build hangs, the abort operation might not take effect. For example,
>
> - if Jenkins is waiting for child processes to complete, it can abort right away.
> - if Jenkins is stuck in an infinite loop, it can never be aborted.
> - if Jenkins is doing a network or file I/O within the Java VM (such as lengthy file copy or SVN update), it cannot be aborted.
>
> So if you think the build time out isn't taking effect, our default assumption is that the build is hanging at the place that cannot be interrupted.

Exercise 5 - build timeout

- create freestyle project ‘endless’
- make it accept a string parameter SLEEP_TIME
- build step: shell
  - python -c "import time; time.sleep(${SLEEP_TIME})"
- define build timeout at 30 seconds and make the job abort
- if the job is aborted - make it trigger itself with SLEEP_TIME-10 (use ‘echo $((${SLEEP_TIME}-10))’)

Exercise 6

- On the slave: install tomcat server: *sudo yum install tomcat*
- Create a maven project 'build_app' to build (on slave) code from :
  - https://github.com/antweiss/courseProject.git
  - Maven goal: package
  - Archive the resulting .war file to Jenkins master
  - Find the war file name and write its name to a properties file in job workspace :
    - export WAR_FILE_NAME=`find . -name *.war | rev | cut -d/ -f1 | rev`
    - echo WAR_FILE_NAME=${WAR_FILE_NAME} > my.properties
  - Inject environment variables from my.properties (need environment injection plugin)
  - In a separate build step: execute shell to echo the name of the war file

Exercise 6 - continued

- Create a freestyle project ‘deploy_app’:
  - String Parameter - URL of the archived .war file on Jenkins server
  - Execute shell to:
    - wget <url_to_war>
    - copy the .war file to /var/lib/tomcat/webapps
    - sudo systemctl restart tomcat
  - Note:(you will need to grant sudo access to jenkins user if not running jenkins slave with root)
  - Make ‘build_app’ trigger ‘deploy_app’ one with the correct parameter (hint -use the built in JOB_URL env variable)

Exercise 6 - continued

- Create a freestyle project ‘test_app’:
  - restrict to run on master
  - test the running app with curl to http://slave_ip
  - write curl output to file
  - trigger this job from deploy_app

Monitoring Jenkins

- Monitoring plugin
- Disk Usage Plugin
- Load Statistics

 http://otomato.link

Managing change

- Audit Trail plugin - keeps a record of user changes
- JobConfigHistory plugin - records config version history
- SCM Sync configuration plugin - save config changes to SCM

 http://otomato.link

Remote Access API

- XML
- JSON

 http://otomato.link