TEKsystems

Our people make IT possible.

TEKsystems Education Services

presents

Continuous Integration with Jenkins Student Manual

20750 Civic Center Drive
Suite 400, Oakland Commons II
Southfield, MI 48076
800.294.9360

IN1502-SG / 5.2.16

Table of Contents

Continuous Integration with Jenkins

Housekeeping

Virtual Logistics

- Put up a green check if you can:
  - Hear instructor
  - View presentation
  - Access course materials:
    Course Manual, Lab Manual, Setup Guide and Files
- Attendance (Roster)
- Please ask questions at any time
  - Unmute and ask questions
  - Chat or "hand up"

- Class Times
  - Breaks, lunch
- Silence cell phones and all electronics

Onsite Logistics

- Facilities
- Sign-in (Roster)
- Course materials

Instructor Welcome

Instructor Introduction:

Name
Background
Qualifications / Experience
Something interesting about yourself, hobby, etc.

Course Prerequisites

- Competency with the SDLC processes including development, integration, test, etc.
- Administrative privileges on your machine
- Familiarity with your current process for integrating code and moving it through stages is helpful

- Basic familiarity with Java is helpful
- Basic understanding of Maven is helpful
  - The instructor will help to fill in any gaps in your technical background

Course Objectives

- Understand what Continuous Integration is and why it's a Best Practice
- Learn how to use Jenkins for Continuous Integration
- Understand how to define builds and deployments
- Understand how and what to trigger builds on
- Gain hands-on experience using Jenkins for tasks such as security and adding plug-ins

Participant Introductions

- Name
- Company
- Department or Team
- Projects / Role(s)
- Technical background
- Top 2 goals for this course
- Something interesting about yourself

Setting Expectations

- Introducing the Continuous Integration (CI) process
- Going in depth on how to use Jenkins for CI
- Hands-on exercises require technical abilities
- Just two days - only so much can be covered

- **If any of your goals are outside of the scope of this class, the instructor will discuss that here**

Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

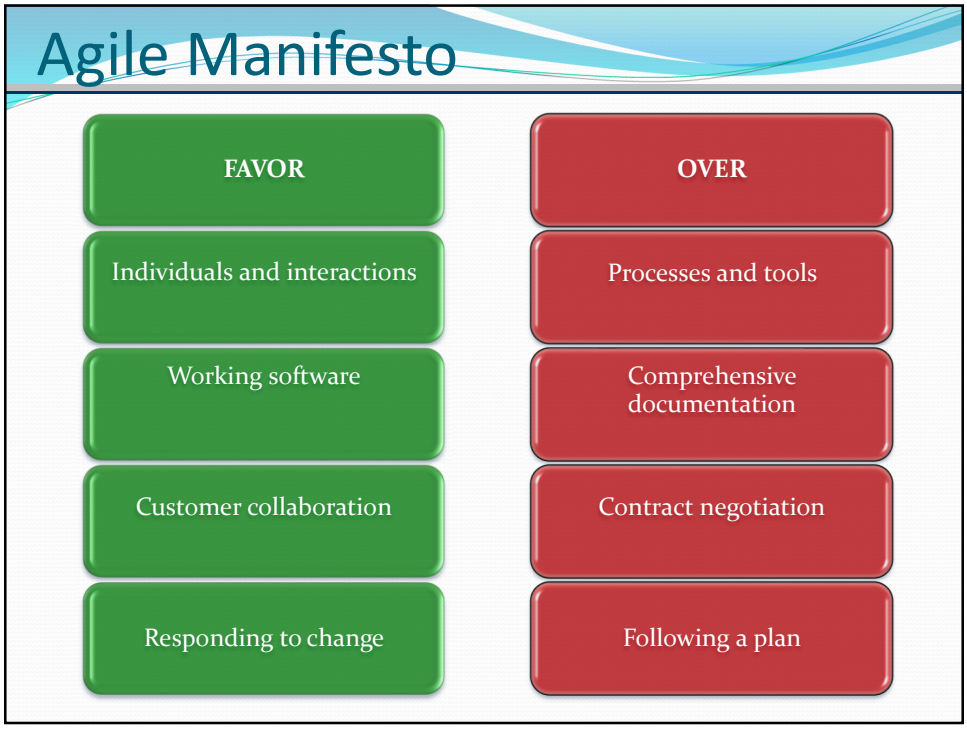
Which of these topics interest you most?
- Use Annotations to mark all that apply

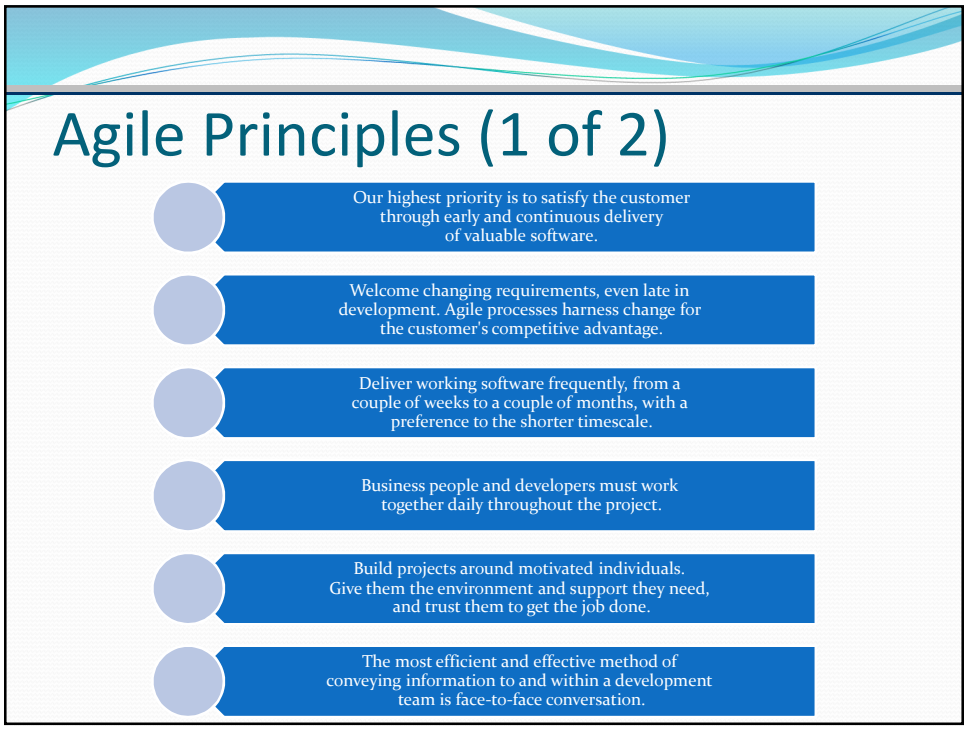
Let's Get Started

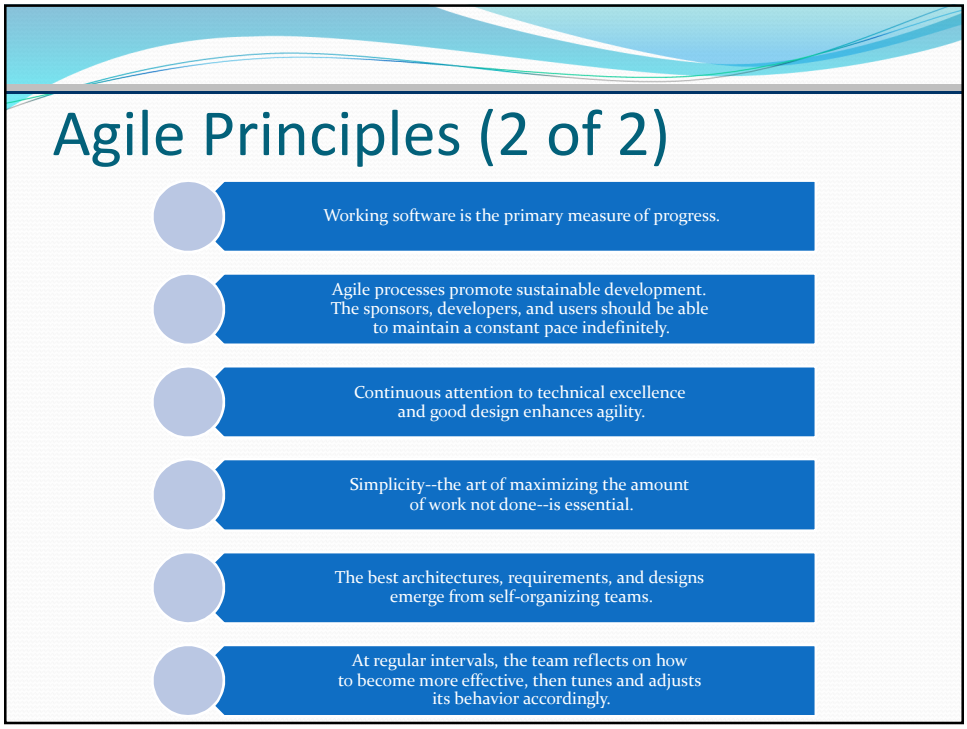
This slide has been intentionally left blank.

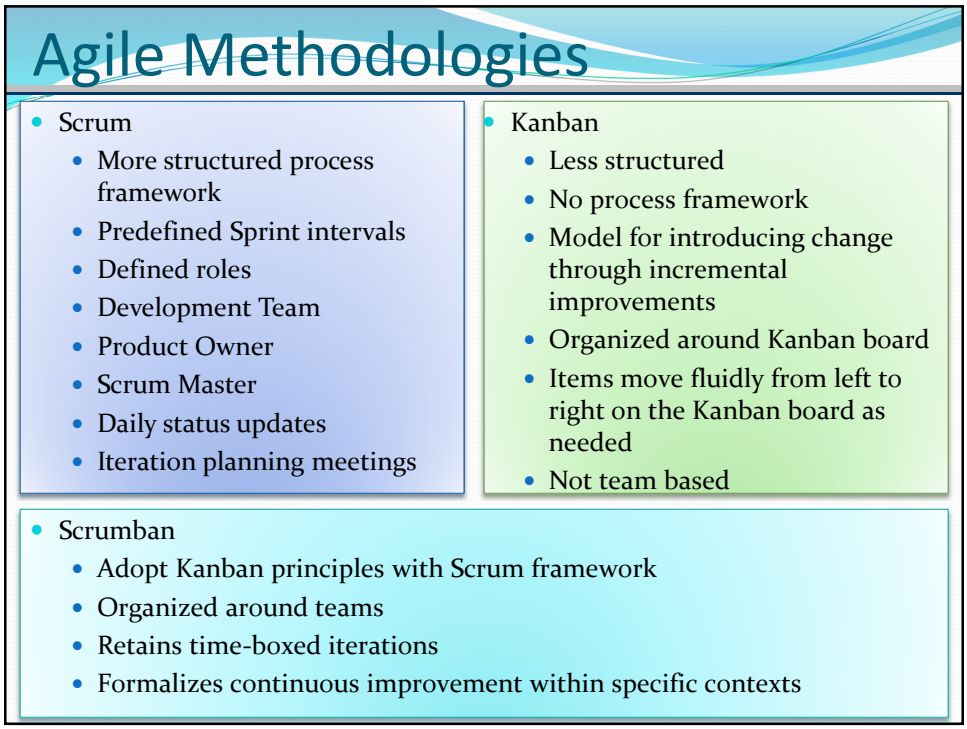
Agenda:

- **Introducing Continuous Integration and Jenkins**
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins









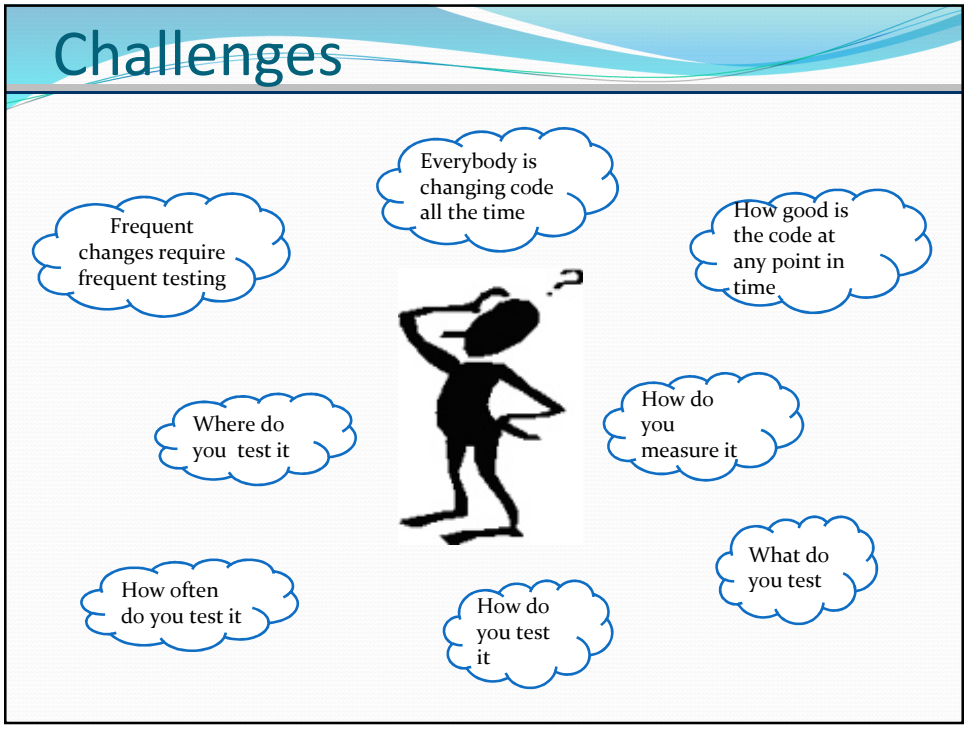
Activity: Build/Deploy Process

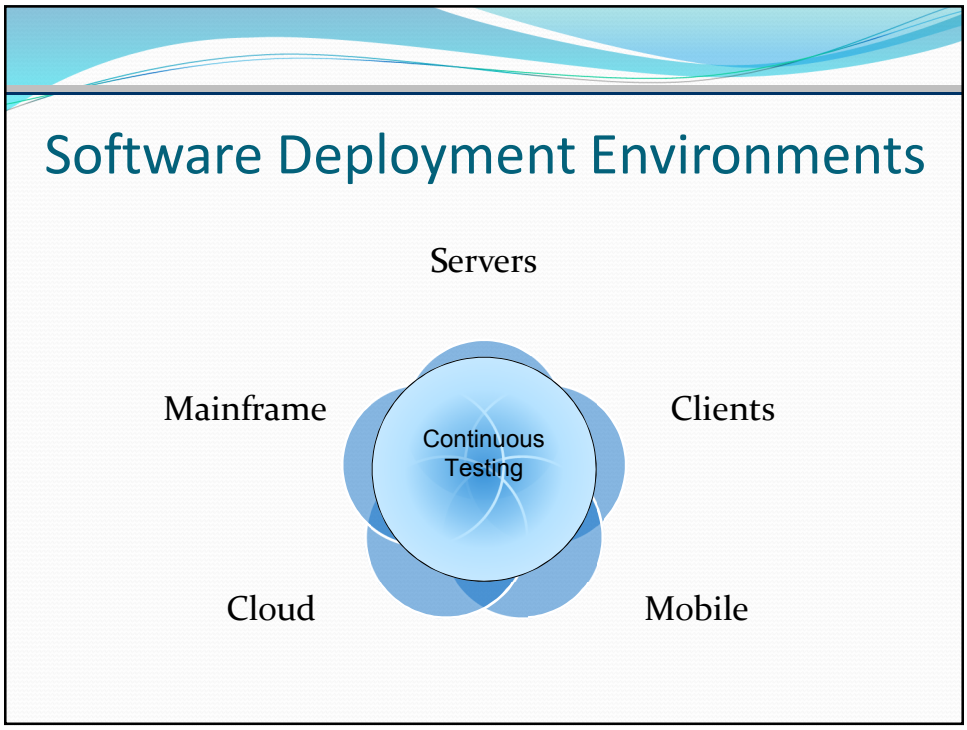
Question: How does your team currently perform its builds and deployments?

Instructions: Use Pointer tool to mark your answer below.

Manually	Using Scripts	Automated Using a Tool	Using Jenkins	CI using Jenkins

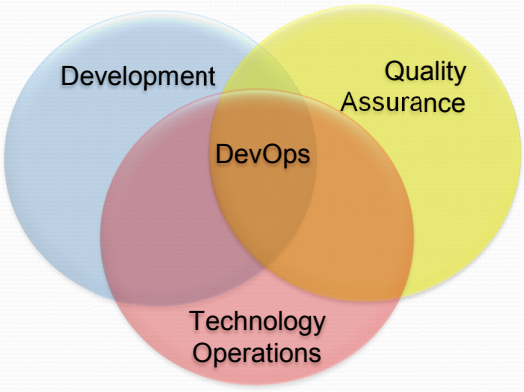
Continuous Integration Needs



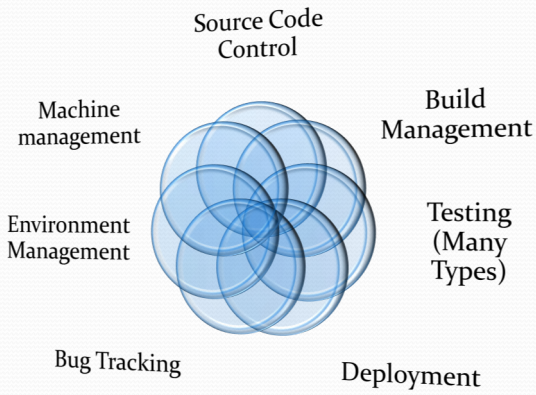


DevOps

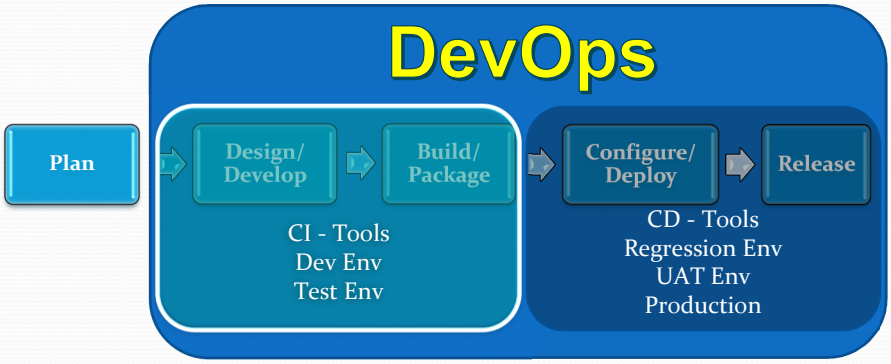
- Fusion of Development (Dev) and Operations (Ops)
- Software development method that stresses communication, collaboration, integration, automation and measurement of cooperation between software developers and IT professionals



DevOps Needs



Where Does DevOps Fit?



Activity: Jenkins Experience

Question: What is your previous experience with Jenkins?

Instructions: Use Pointer tool to mark your answer below.

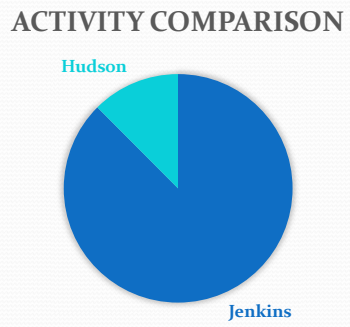
New to Jenkins	Played Around with It	Using Jenkins	Advanced User

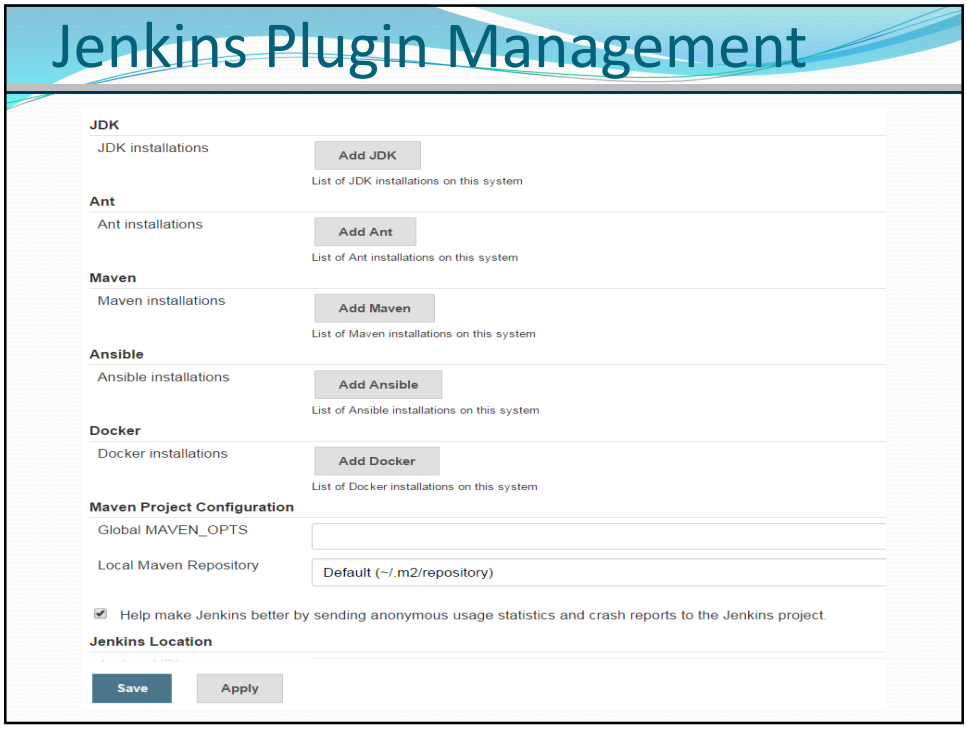
History of Jenkins

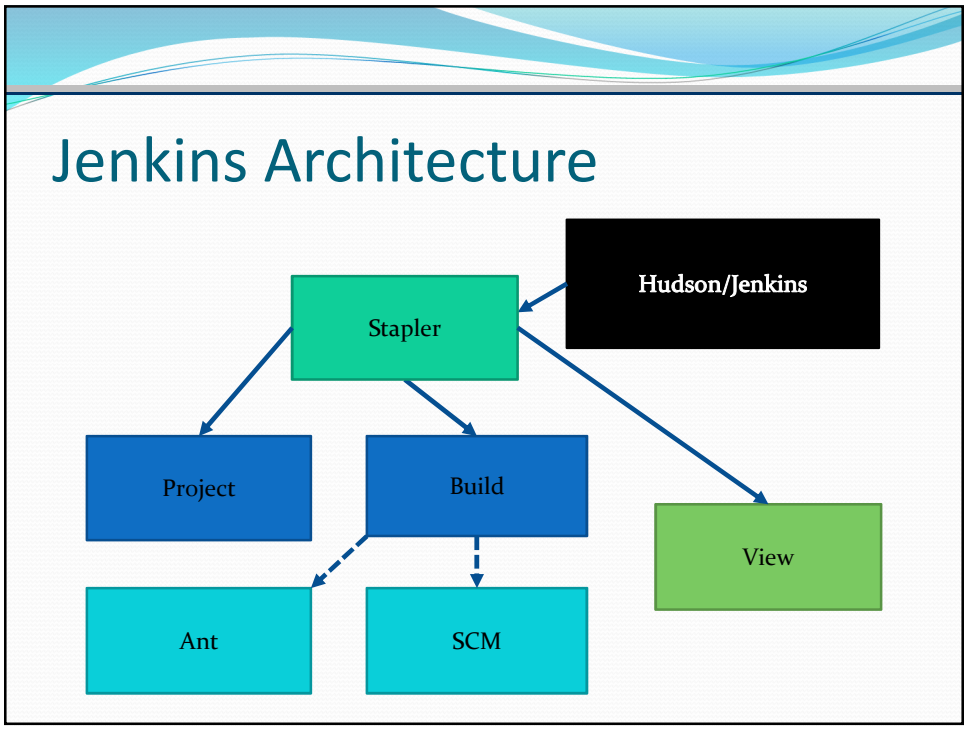
- Overview
  - Started as Hudson project 2004 at Sun Microsystems
  - Open Source Continuous Integration tool
  - Plugin-based architecture supporting many VCS and Maven
  - Also supports Unix and Windows shells
- Evolution
  - 2007 Hudson became preferred to CruiseControl
  - 2010 Oracle/Hudson had a falling out
  - Jenkins became a separate project
  - Most of developers went to Jenkins

State of Jenkins Community

- 1100 public repositories
- 567 project members
- 7 times the activity over Hudson
- Over 1200 plugins
- Current stable version > 1.640
- Jenkins 2.0 released April 2016
  - Not production-ready yet

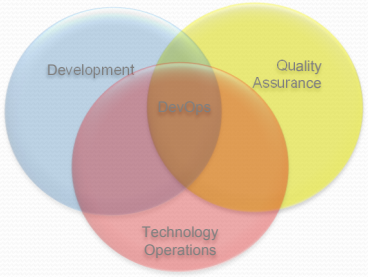






Summary

- Agile requires a new way of thinking about software delivery
- Change is good – how do we do it safely?
- Jenkins automates Continuous Integration



This slide has been intentionally left blank.

Agenda:

- Introducing Continuous Integration and Jenkins
- **Installing and Running Jenkins**
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

Running Jenkins From a WAR

- Download from Jenkins
- **Java -jar Jenkins.war**
  - Creates all the folder structures
  - Populates the initial configuration.xml
  - Registers all the default plugins
- Set **JENKINS_HOME** before to control where to put it

Installing In a Servlet Container

- Set **JENKINS_HOME** before starting the first time
  - Most servlet containers are started with a shell or bat command
- Runs in most servlet containers
- Most servlet containers have their own way of installing a web application
- Usually enough to just install it
- Most functionality works without much customization
- Upgrades are where it can get tricky
- Also versions of JVMs supported by the servlet container can affect Jenkins
- For specific servlet containers:
  - See https://wiki.jenkins-ci.org/display/JENKINS/Containers

Setup Security

- Jenkins running from the war – simple security
  - Use internal database
  - Use single admin account
- Running from servlet container
  - Use internal database
  - Use servlet container security
  - Windows – Active Directory plugin
  - LDAP plugin

- Authentication – Who are you?
- Authorization – What can you do?
- **More on this later…**

Activity: Security Practices

Question: What types of security does your team currently use for its projects?

Instructions: Use Annotation tool to place a mark below on all answers that apply.

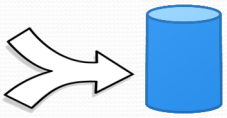
Simple Security	Plug-in Security

User Authentication	Authorization by Role

E-Mail and Version Control

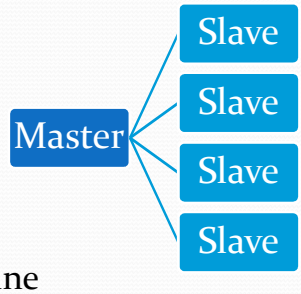
- Notification of Job progress can be sent via E-Mail
- Configure the SMTP sending information
  - Host
  - Authentication information
  - Sender email address

- Configure the Version Control System
  - CVS – by default
    - Key information and authentication contexts
  - SVN – by default
  - Many more available



Master/Slave Configuration

- Distributed builds need slaves defined to execute on
- Master defined on install
- Slaves must be added
  - Windows slaves
  - Unix slaves
  - Mac slaves
  - Define executors to run
  - Set environment variables
- JDK must be on the slave machine
- Jenkins can install the other things needed



Exercise: Install Jenkins

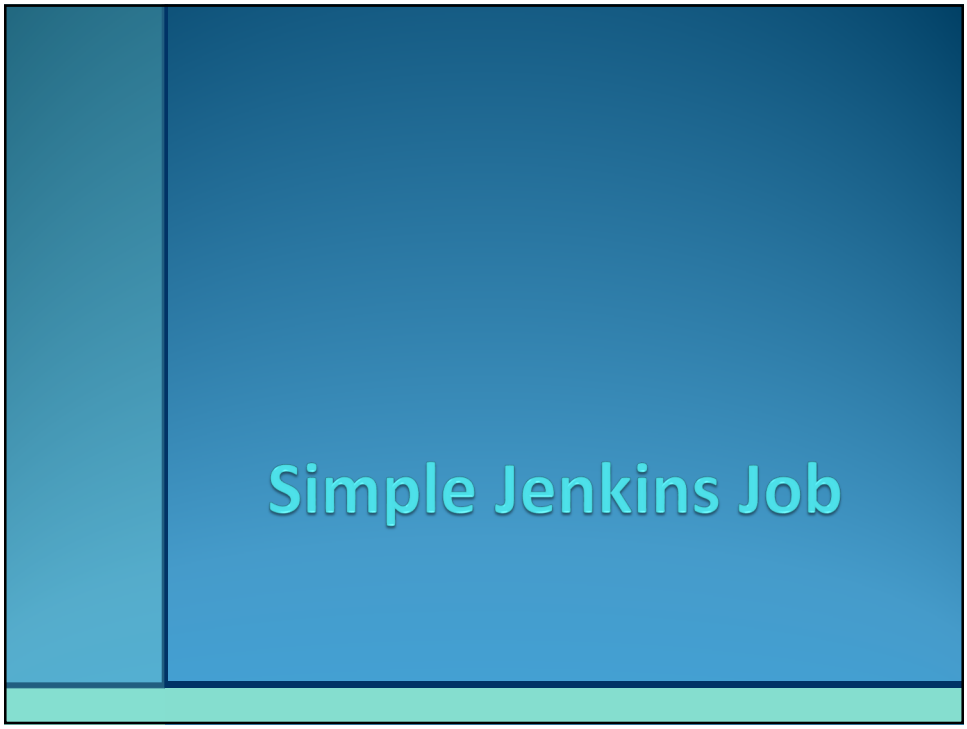
- Install Jenkins
- Customize the Jenkins environment
- Explore the broad range of options available in Jenkins
- Configure Jenkins to run in Tomcat
- Test the installation

- **Ask or chat questions at any time**
- Afterwards, we'll regroup to review the exercise

Summary

- Installing of Jenkins is really easy
- Self-installing with working defaults
- Best to secure right away
- Notification just needs access to an SMTP host to send mail
- Should always run distributed build
  - Define nodes before any jobs

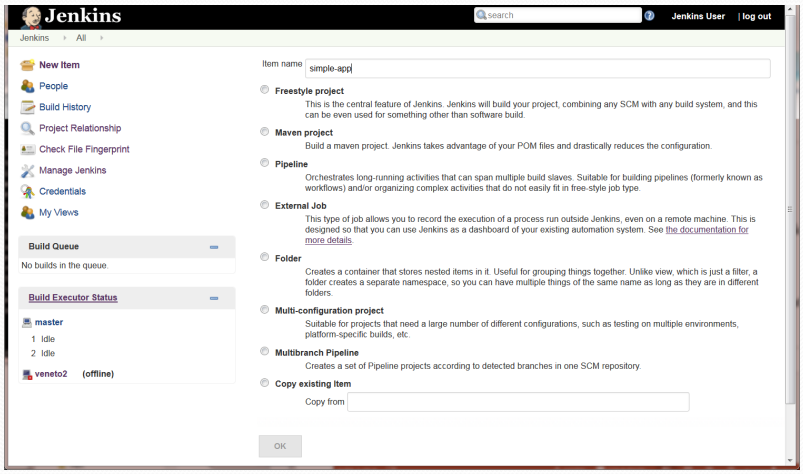
- Any questions?



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- **Simple Jenkins Job**
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

Create a Job



Configuring a Job

- Jobs do nothing without some configuration
- Configuration depends on plugins available
- Configuration depends on project type
- Configuring one property on a plugin may provide additional configuration steps
- We'll run some of the items later

Run a Job Manually

- Click the **Build Now** button

- From main screen
- From a project
- From a pipeline view

- Prompted for parameter if defined

Exercise: Create a Job

The simple job:
- copies a set of Java classes and **JUnit** tests to the Jenkins workspace
- builds the project using **Maven**

To accomplish this, you will perform these tasks:
- Configure and install Maven and its dependencies
- Define and configure a new Maven Job
- Copy the provided **simple-app** project (from **setup.zip**)
- Run the Build and view the results
  - Compiles the Java classes and the Test classes
  - Executes the unit tests
  - Installs the artifact into the Maven local repository

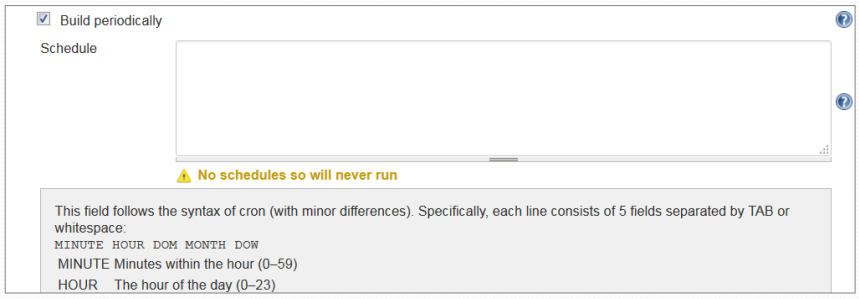
Afterwards, we'll regroup to review the exercise

Running Jobs Automatically

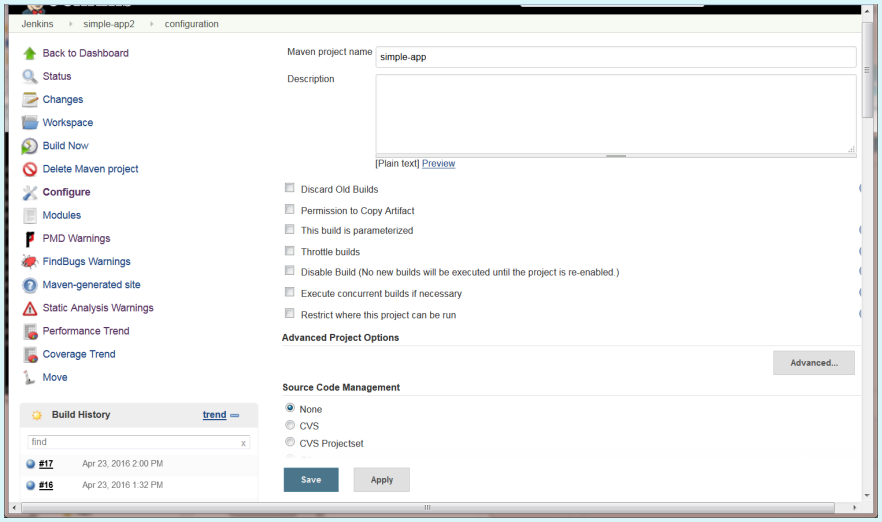
- **Configure** the job to run automatically
- **When source changes**
  - SCM must be configured
  - Specify how often to check the version control system
- **On a schedule**

Run a Job on a Schedule

- Uses **cron** type schedule



Demo: Jenkins Job *simple-app*



Exercise: Schedule a Job

- Import a Cucumber Unit Test job into Eclipse from Git
- Create a job in Jenkins
  - Configure the Git repository
  - Configure the schedule to run it

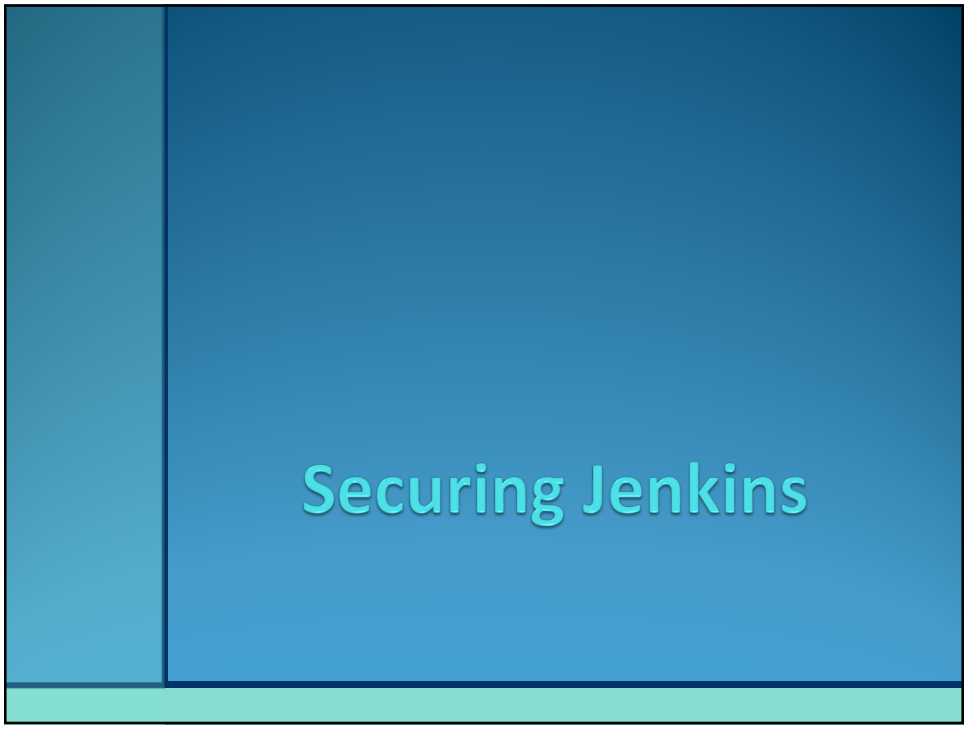
- Save the job and wait until the build is triggered by schedule
- Update the code in Eclipse and commit
- Observer the triggering of the job by the source commit

- **Ask questions at any time. Debrief/review after**

Summary

- Jenkins provides so many options to build projects
- No project can really run without some configuration
- Connect to SCM
- Connect to other projects
- Connect to build application

This slide has been intentionally left blank.



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- **Securing Jenkins**
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

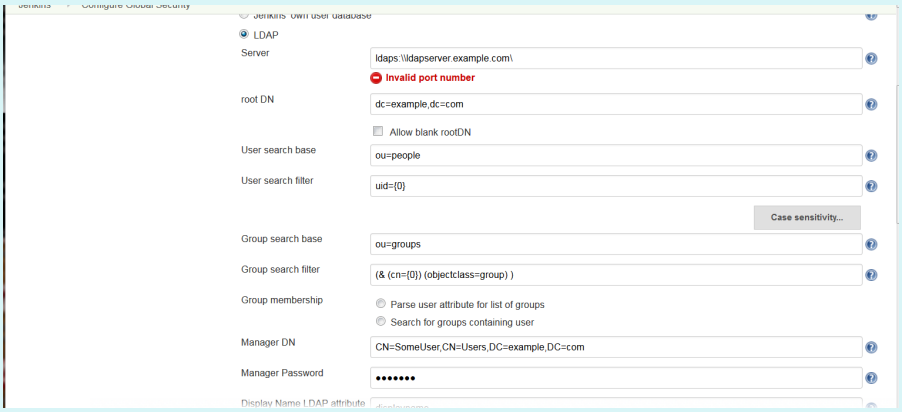
Authentication and Authorization

- Authentication – Who are you again?
  - Identities
- Authorization – What are you allowed to do?
  - Permissions
- Master/Slave security

Setting up Jenkins Security

- Use container authentication on application server
  - Tomcat, WebSphere, JBoss, etc.
  - Don't protect URLs, cli, git, jnlpJars, subversion, whoAmI
- Use Jenkins own database
  - Allow self-registration
  - Preconfigure users
- LDAP

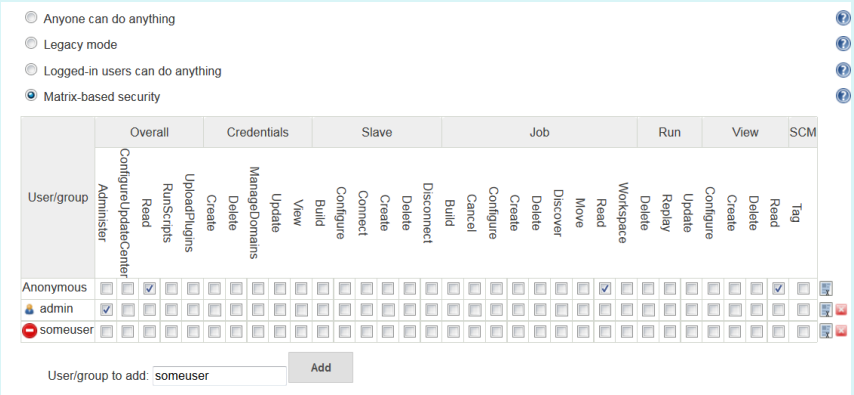
LDAP



Put up a green check ✓ if your team uses or will use LDAP.
Otherwise put up a red **X**.

Authorization

- Roles to permission mappings
- Users/groups to allow



Do you think your team will use Jenkins authorization? ✓ or **X**

Master/Slave Security

- Slaves run on different machine
- Need to get jobs and content from Jenkins master
- Some access is necessary
- Security across the machines is important
- Currently only command based whitelisting
  - What commands can run on slaves
  - What files can be retrieved
- Slaves can be restricted by which user they run under

Exercise: Securing Jenkins

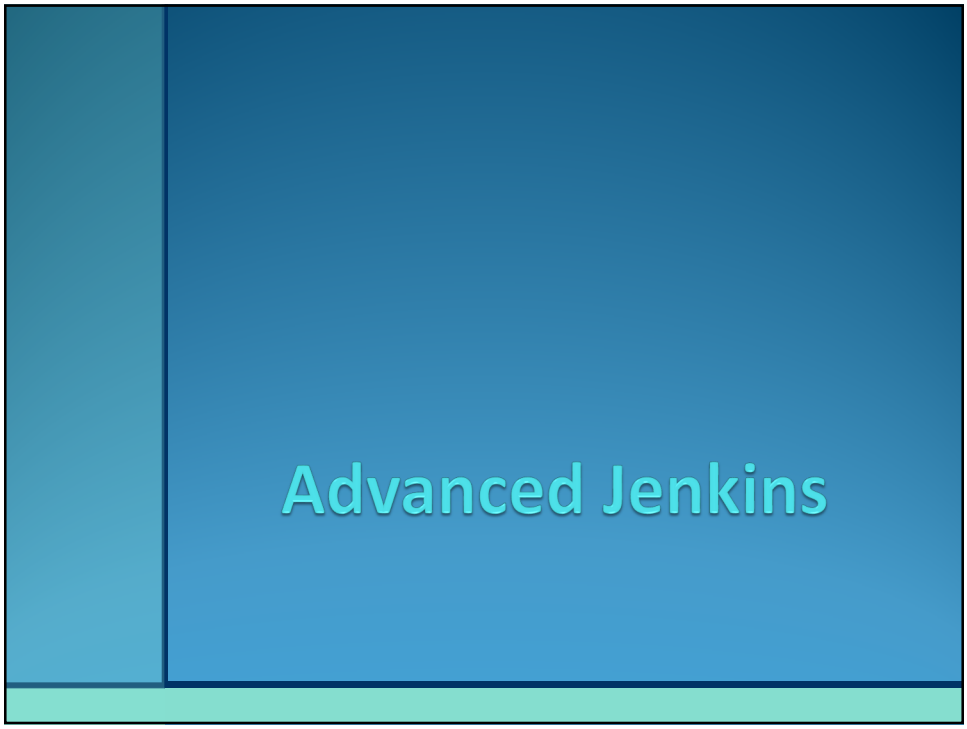
- Enable security using the built-in Jenkins database
- First, just authentication. Later, add authorization
  - Self-register a user
  - Assign user privileges
  - Self-register a different user
  - Give users different privileges
  - Switch users and see what is permitted

- **Ask questions at any time**
- **Exercise Review/Debrief afterwards**

Summary

- Always secure Jenkins!!!!
- First thing to do is secure Jenkins
- Configure how Jenkins knows who the user is
- Configure what Jenkins allows the user to see/do
- Supports limited security in standalone mode
- More security options in container base security

This slide has been intentionally left blank.



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- **Advanced Jenkins**
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

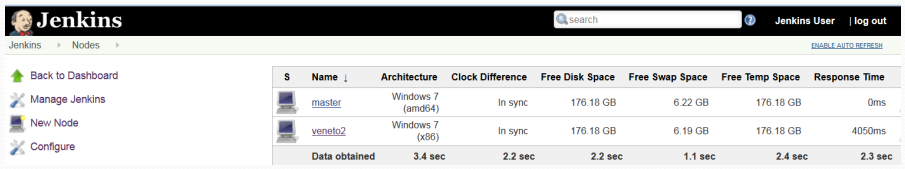
Monitoring External Jobs

- Create a **Monitor external job** project
- On the machine to be monitored
  - Run a Jenkins Java job that launches the job to monitor
  - Job finishes and triggers the results in Jenkins master
- Security can be defined
  - Better to use anonymous jobs if possible
- Alternatively any XML sent to Jenkins can trigger a run
  - Example:

```
curl -X POST -d '<run><log encoding="hexBinary">4142430A</log>
          <result>0</result><duration>2000</duration></run>' \
http://user:pass@myhost/jenkins/job/_jobName_/postBuildResult
```

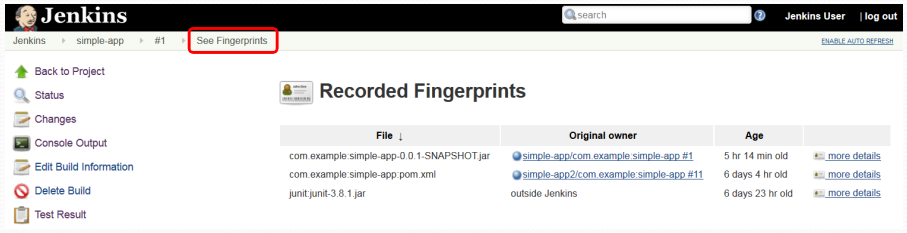
Distributed Builds

- Create new **Nodes**
  - Based on machine name
  - Can be started with JNLP
  - Can be started as services
- Jobs can be configured to only run on specific nodes



File Fingerprint Tracking

- Configure all jar jobs to record fingerprints
- Click **See fingerprints**

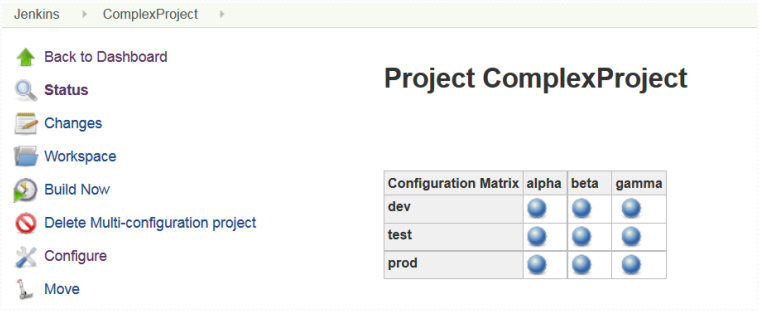


Using Jenkins for Non-Java Projects

- Create like any other project
- On Linux/Mac, jobs can **execute shell** commands
- On Windows, jobs can **execute windows batch commands**
- Checkout builds as with Java projects
- Set parameters and environment variables
- Publish plugins work for any artifact type

Matrix Projects

- Same steps but different implementations or parameters
- Matrix of parameters (Release X Environment)



Activity: Types of Jobs

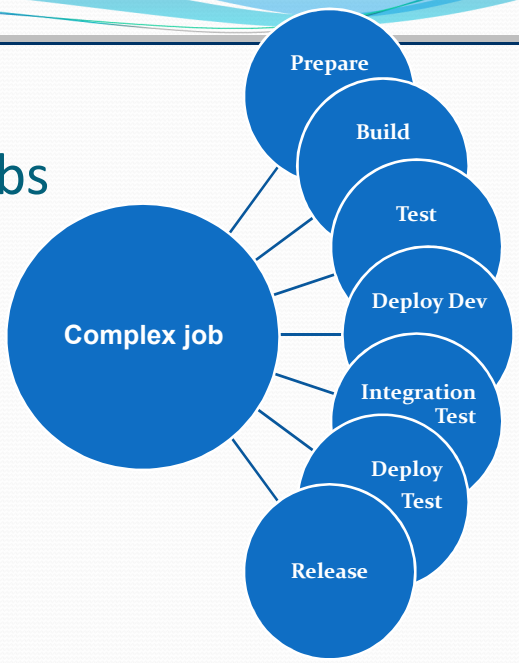
Question: What types of jobs other than Maven and Freestyle have you used?

Instructions: Annotate your answer below.

External	Matrix	Non-Java	Other

Splitting Jobs into Smaller Jobs

- Complex jobs can be simplified by decomposition



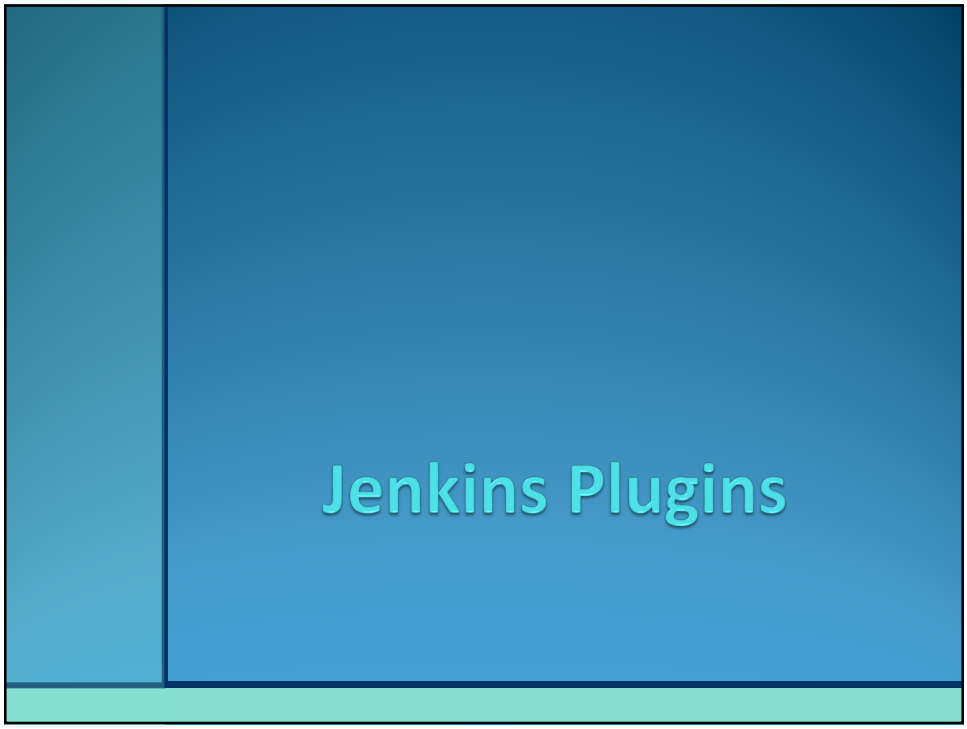
Exercise: Complex Projects

- Create a Multi-configuration project
- Define the configuration matrix
- Add a build step to execute a command
  - Copies a file from one folder to another
- Build the project
- View the results
- **Ask questions at any time**
- **Exercise Review/Debrief afterwards**

Summary

- Jenkins orchestrates a sequence of steps to accomplish an automated process reliably and predictably
- Jobs are more than just Maven builds
- Many jobs require complex relationships
- Jenkins allows for very complex job configurations

This slide has been intentionally left blank.



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- **Jenkins Plugins**
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

Jenkins Plugins

- Plugins provide almost all Jenkins functionality
  - Core provides a very limited set of functionality
    - Read the configuration
    - Load plugins
    - Build views
- Plugins do everything else
  - Authenticate
  - Authorize
  - Build
  - Configure
  - Check out from SCM
  - Scheduling

Change Reporting

- Important to know what changed
- Comments describe why it changed
- Stored with build
- VCS tools can provide change information
  - CVS
  - Subversion
  - Git
  - Team Foundation Server
  - Rational Team Concert
  - Perforce

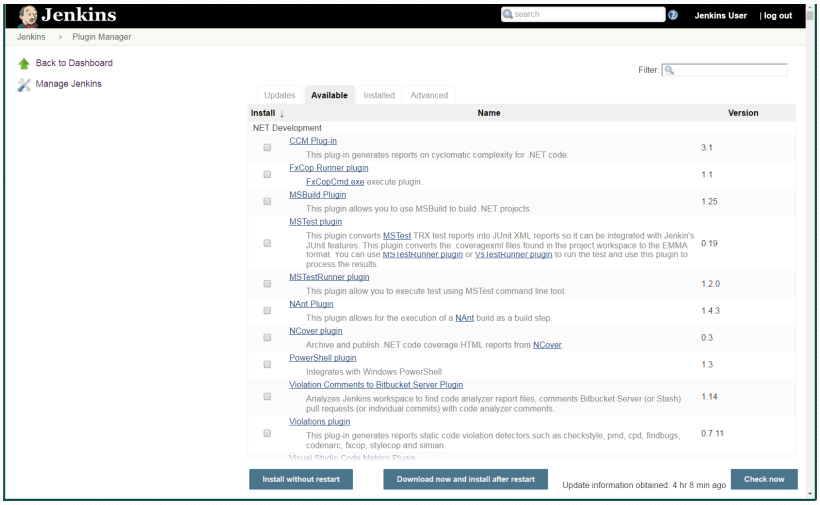
Core Plugins

- Scripts management
- Credentials management
- Job monitoring
- Authentication and authorization
- Email
- Maven
- Ant
- JUnit
- LDAP
- Master/slave support

Common Plugins

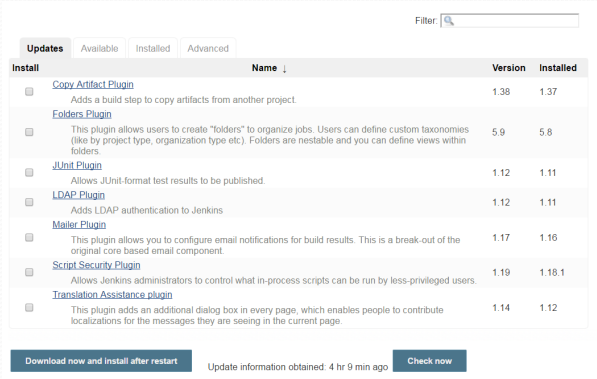
- Source Control Management
  - Git, CVS, Subversion, etc.
- Analysis collector
  - PMD, Checkstyle, FindBugs, etc.
- Authentication
  - LDAP, Active Directory, OpenAuth
- Pipeline
  - Build pipeline, Stage view, Pipeline Steps, etc.
- Parameterized builds
- Release promotions
- Green Balls – makes the balls green instead of blue

Manage Plugins - Available



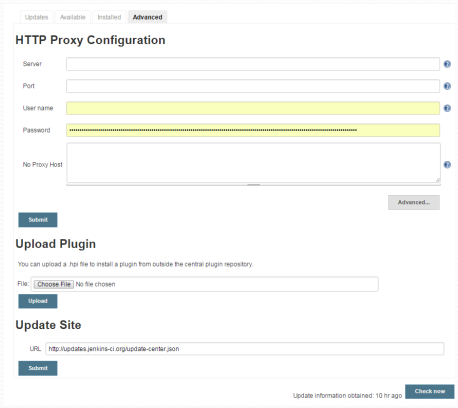
Updating Plugins

- List of updated plugins is shown
- Restart required
- Check to update



Advanced Tab

- Define proxy setting
- Install plugin manually from .hpi
- Add a new update site

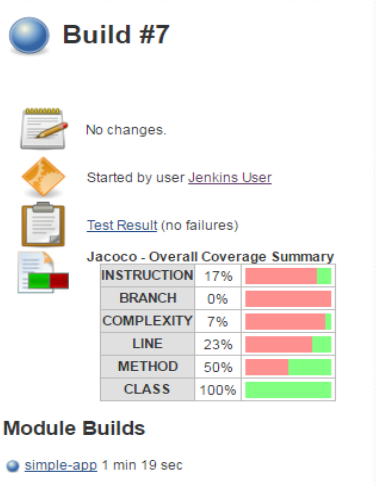


Activity: Explore Plugins

- Open Jenkins
  - Click **Manage Jenkins**
  - Click **Manage Plugins**
  - Click **Available** tab
- Explore the available plugins to consider how they might be useful
- **Be prepared to suggest some plugins and why you think they would be good in your processes**

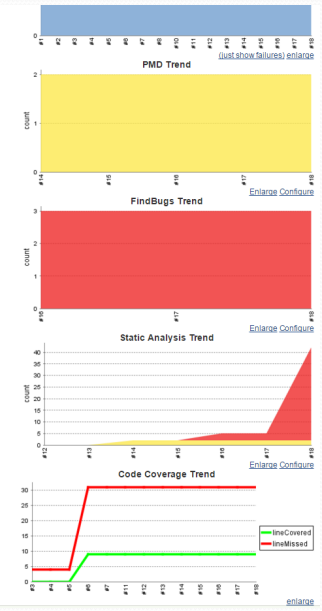
Code Coverage

- Important to assess overall code quality
- Poor code coverage increases risks of defects
- Supports many code coverage plugins
  - Clover
  - SonarQube
  - Emma
  - JaCoCo
  - Cobertura
  - Ncover
  - Serenity



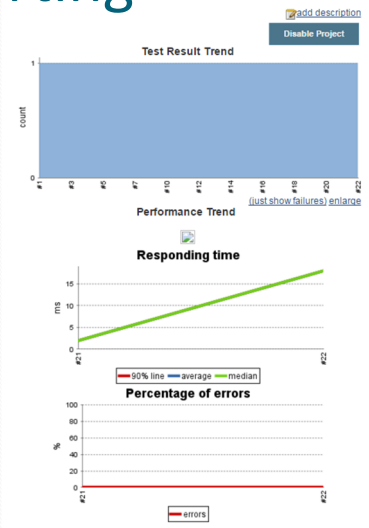
Static Analysis

- Combines output from various plugins
  - PMD
  - FindBugs
  - Checkstyle
- Another option is SonarQube plugin



Performance Reporting

- Graphed over time
- Performance Plugin – used to be JMeter plugin
- Now supports JUnit test



Style Checking

- Some tools are both static analysis and style checkers
- Concern with readability and maintainability
- Lint has been around for years
- Check for adhering to coding styles
  - Curly braces
  - End of line
  - Existence of comments
  - Length of comments

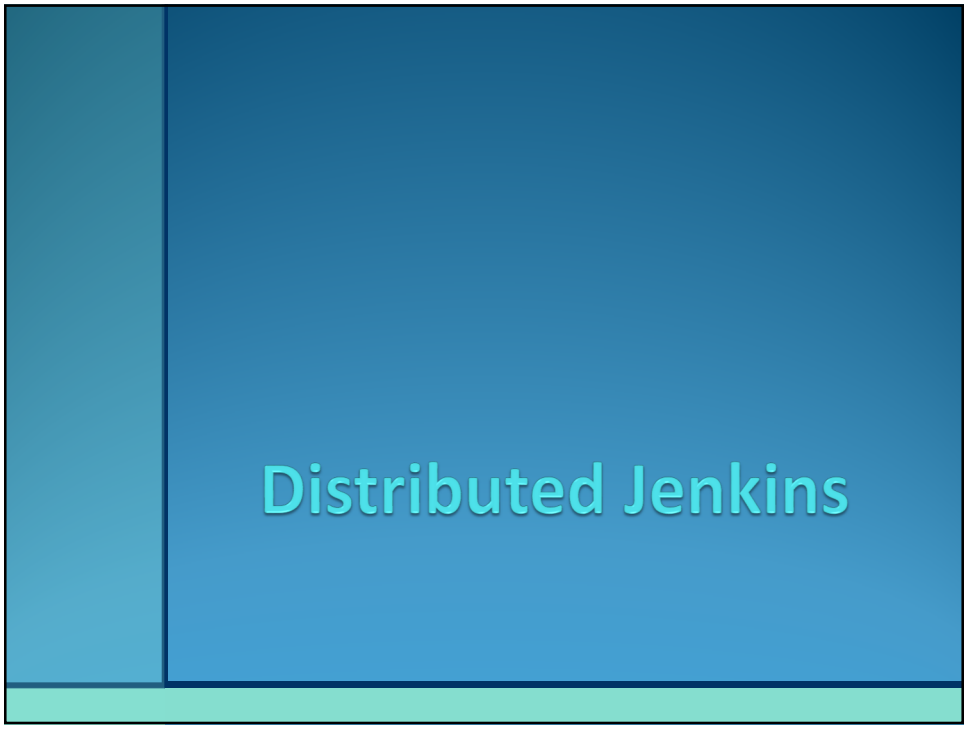
Exercise: Plugins

- Install several plugins for code quality
- Create a new Maven project named **simple-app2**
  - Add a pre-build step to copy the provided application from the setup folder
  - Configure the build to use the plugins
  - Add post-build steps to produce the code quality reports
- Build the project and view the code quality reports

- **Questions welcome**
- **Exercise Review/Debrief afterwards**

Summary

- Plugins do almost all the work for Jenkins
- Hundreds of plugins available
- Many updates every day
- Easy to install the plugin
- Some configured at system level
- Some configurations are per job
- Some perform a function on a job status



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- **Distributed Jenkins**
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

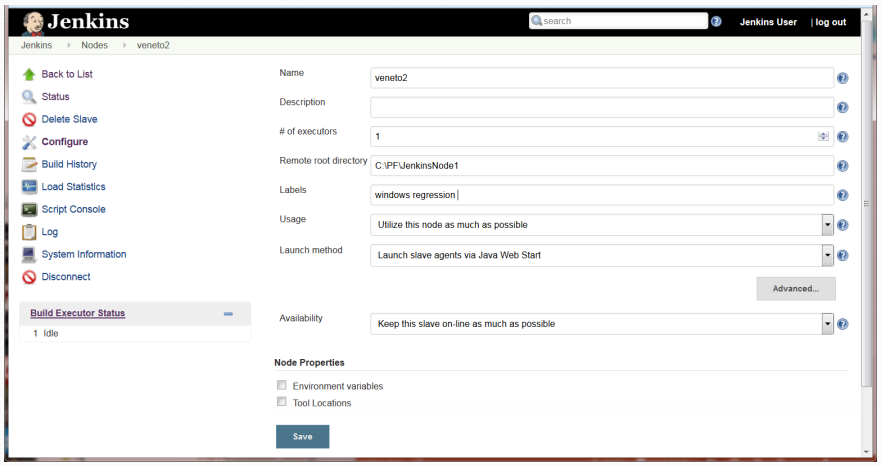
Why Distribute Workload?

- Example from real world
  - Release build
    - Automated build
      - Runs in 1/4 hour
    - Automated Unit tests
      - Runs in 1/2 hour
    - Automated Integration tests
      - Runs in 3 hours
    - Automated Load and Stress test
      - Runs in 6 hours
    - Released if passes
- Run 4 different applications daily
- Developer check-ins trigger build

Setting up a Distributed System

- Install Jenkins as central build master
- Create Nodes on machines to perform builds
- Assign labels to the nodes as descriptions that can be searched during runtime execution
- Can also create nodes with Docker containers
  - Needs SSH, a JDK and a user
  - Create the image and configure the Docker plugin

Node Configuration



Running Jobs on Nodes

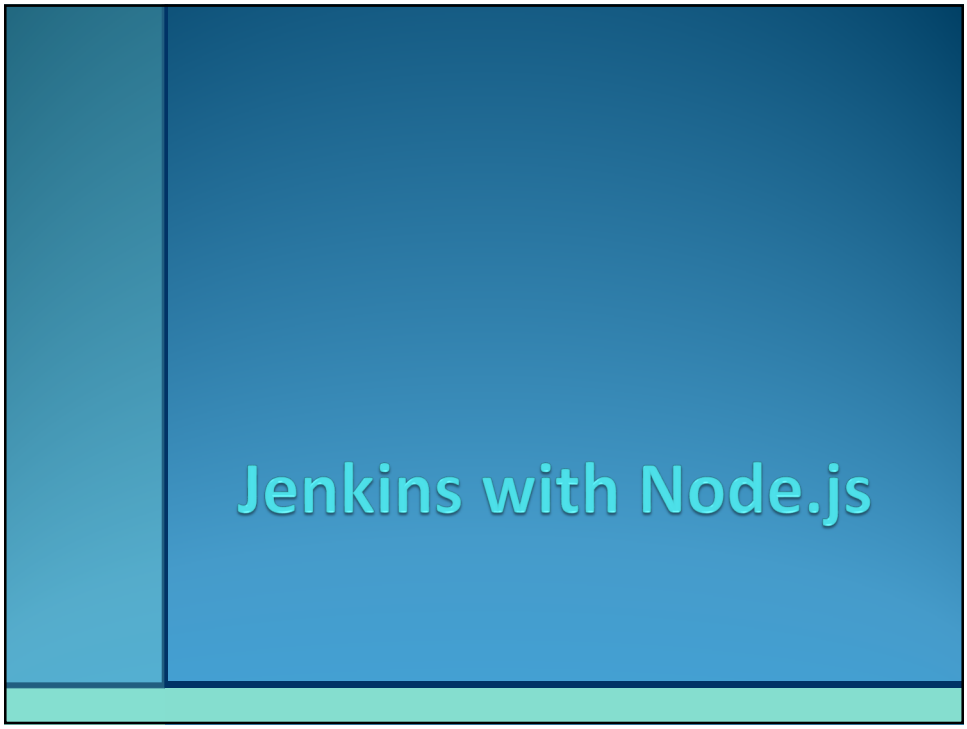
- Configure a job
- Specify **Restrict where job can run**
- Provide the label expression
  - Node1
  - "jenkins-solaris (Solaris)" || "Windows 2008"
  - Regression
- Use quotes if label includes spaces

Exercise: Distributed Build

- Ensure Tomcat is running as administrator
- Define a **Dumb Slave** node
- Define a new Maven job named **distributed-app**
- Configure the build **schedule** and **repository location**
- **Restrict** the job to run only on the node you defined
- Configure the other job settings
- Run the job and review the results
- **Exercise Review/Debrief**

Summary

- Distributed systems greatly enhance Jenkins throughput
- Nodes should always be labeled
- Jobs should be restricted to nodes with the right context
- Some jobs run better on Windows
- Some jobs run better on Linux
- Some jobs can only run on Macs



Agenda:

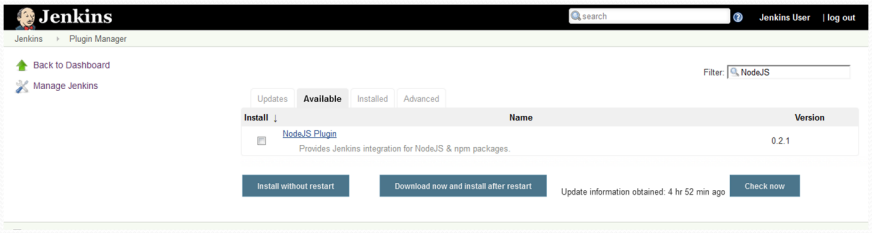
- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- **Jenkins with Node.js**
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

Node.js or NodeJS?

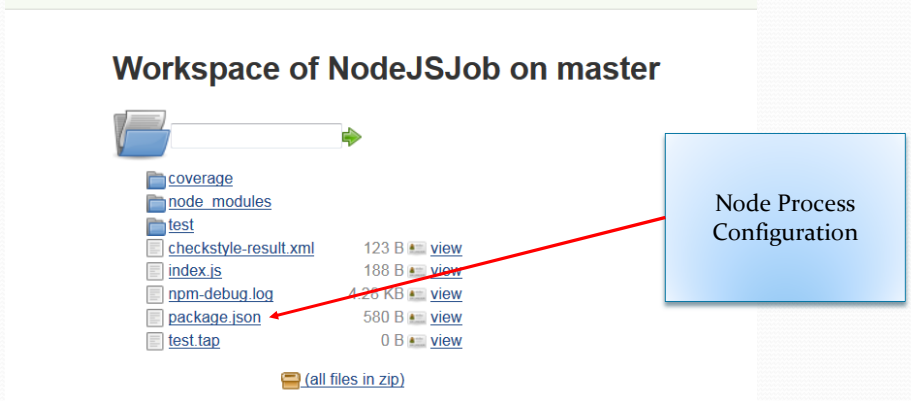
- Correct naming convention?
- **Node.js** - as written on the **nodejs.org** official website
  - This is how we refer to the *framework*
- **NodeJS** - name of the plugin available within Jenkins
  - This is how we refer to the *plugin*
- **NodeJSJob** - name of a Jenkins job you will create during an exercise
  - This is an arbitrary *job* name

Node.js Support in Jenkins

- Install the NodeJS Plugin
- Configure the NodeJS Installation
- Add build steps to the job that run **NodeJS scripts**
  - You'll do this later during an exercise



Structure of JavaScript Projects



package.json

```
{
  "name": "nodejs",  "version": "1.0.0",  "private": true,  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "istanbul cover tape \"test/*-test.js\"",
    "lint": "eslint index.js",
    "ci-test": "istanbul cover tape \"test/*-test.js\" > test.tap && istanbul
        report clover",
    "ci-lint": "eslint -f checkstyle index.js > checkstyle-result.xml"
  },
  "keywords": [],  "author": "someuser <johndow@gmail.com>",  "license":
       "ISC",
  "devDependencies": {
    "eslint": "^0.10.0",
    "istanbul": "^0.3.2",
    "tape": "^3.0.3"
  }
}
```

- Scripts defined for running **lint**, **test**, **ci-test** and **ci-lint**
- Dependencies are defined as **library** and **version**

Poll

1. What is the name of the Jenkins plugin?
   A. Node.js
   B. Node.JS
   C. Nodejs
   D. NodeJS

2. Where do the instructions for the Node process go?
   A. package.json
   B. node.config
   C. node.js
   D. package.js

Integration with TestRunner

- Many test runners for Node.js
  - Karma – unit test runner
  - Protractor – integration tests with Selenium
  - Jasmine – for those with Ruby background
  - QUnit – browser-based unit tests
  - Mocha – very common test runner with async support
- Install tool in node "**npm install mocha**"
- Include TestRunner script in the **package.json**
- Define test cases in **.js** files
- Run "**node run test**" to launch node and run tests

Style Checking JavaScript Projects

- Several JavaScript style checking options
  - **jscheckstyle**
  - **eslint**
  - **Jshint**
- Install the style checker
- Add the script in the **package.json**
- Run node providing the script name
- Reports will be output

Starting Integration Servers

- Many **Platform as a Service** (**PaaS**) node vendors
  - Digital Ocean
  - Openshift
  - Microsoft Azure
  - Heroku
  - HP - Cloud
  - IBM – BlueMix
- Host your own
- Deploy the node application from Jenkins
- Start node as a server "**node server**"
  - http://localhost:5000/ to connect to node

Example Node.js Web App

```
var http  = require('http');
http.createServer(function (request, response))
{
      response.writeHead(200,{
      'Content-Type': 'text/plain',
      'Access-Control-Allow-Origin': '*'
      });
      response.end('Hello World\n');
}) listen(5000);
```

NodeJS Plugin Exercise

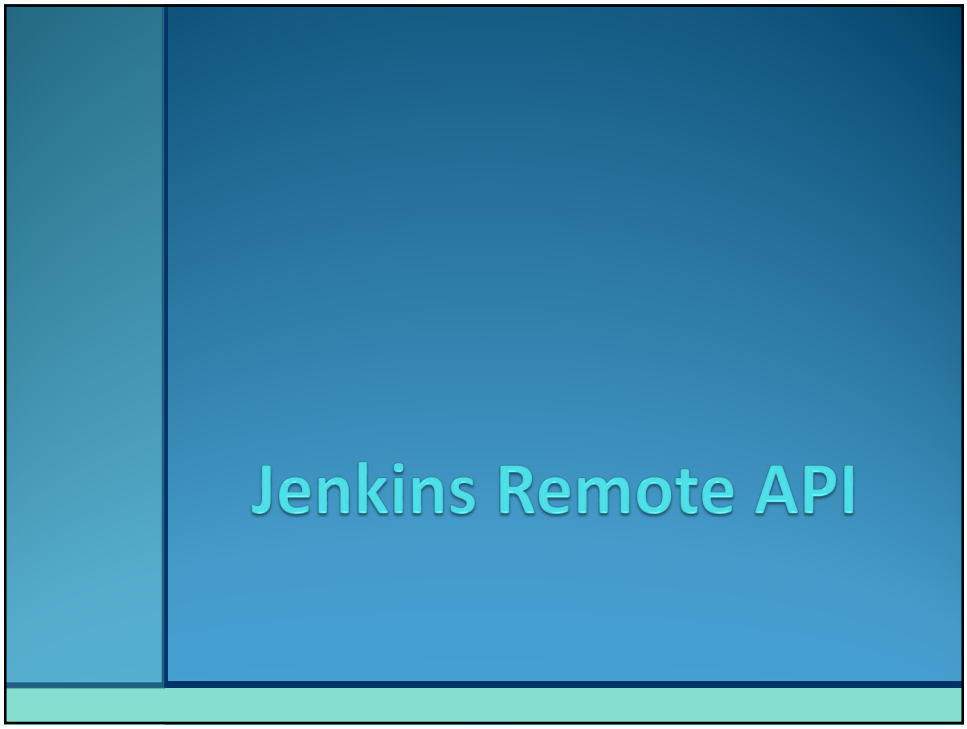
- Install **NodeJS plugin**
- Create a Freestyle job named **NodeJSJob**
- Add a build step to display the versions in use
- Add a build step to run a **NodeJS script** command
- Build the job and view the console output

- **Exercise Review**

Summary

- Node.js is a very powerful, efficient platform
- Some estimates are 40% less time to server pages
- Popularity is growing
- CI and CD teams will be expected to deploy to node
- Jenkins can handle it

This slide has been intentionally left blank.



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- **Jenkins Remote API**
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

Service Calls

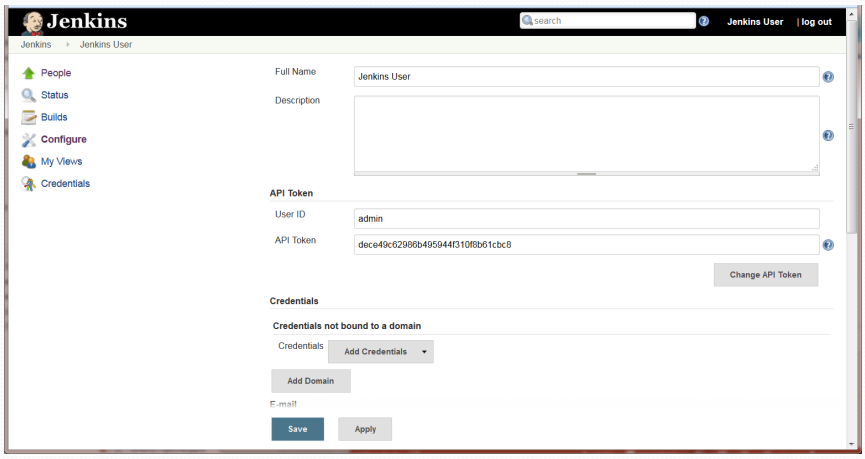
- API provides access to jobs
  - Build
  - Information
  - Create jobs
  - Delete jobs
  - Request Shutdown/Restart
- REST API
- Example:

```
curl -X POST http://localhost:8080/jenkins/job/job_name/command
```

Remote API Security

- URL can include a userid and password
  - **http**://user:pass@host/Jenkins/job/command
  - Not very secure
- URL can include token=TOKEN_VALUE
  - **https**://.../command?token=123456
  - Secured with https
- URL can include an API token assigned to a user
  - https://.../command
  - Basic authentication browser asks for userid/password
  - Provide userid and API token

Getting the API Token



Triggering a Build

- Any web request builder will work
  - **Curl**
  - **Wget**
  - Java **httpclient**
- URL = `Jenkins URL/job/JOB_NAME/build`
  - Security context if needed:
    - userid:password in Jenkins URL
    - Userid:apitoken in Jenkins URL
    - token=xxxx as query parameter
    - username and apitoken as query parameter
- Use query parameters to provide additional parameters

Retrieving Information

- Retrieve data formats
  - XML
    - http://jenkinshost:8080/jenkins/job/name/api/xml
  - JSON
    - http://jenkinshost:8080/jenkins/job/name/api/json
- Python API allows conversion to Python objects
  - `eval(url/Jenkins/job/name/api/python).read()`
- Can add depth as query parameter to limit information
- The tree query parameter can define attributes wanted
- Use **XPath** to filter attributes wanted

Activity: Result Format

Question: With which format do you have the most experience?

Instructions: Use Annotation tool or the Pointer to place a mark below on the answer(s) that apply.

XML	JSON

Discussion: Why pick XML or JSON when using the remote API to retrieve information about the jobs and configuration?

API Libraries – Python

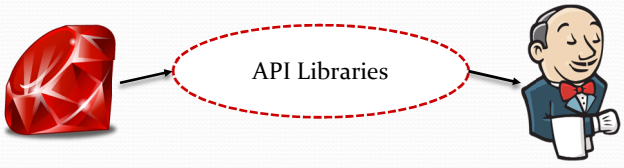
- Python-Jenkins API wrapper for Jenkins REST API
- Example Python code:

```
server = Jenkins('http://localhost:8080')
server.create_job('empty',
jenkins.EMPTY_CONFIG_XML)
jobs = server.get_jobs()
print jobs
server.build_job('empty')
```

API Libraries – Ruby

- Allow access to Jenkins from Ruby application
- Example:

```
require "jenkins_api_client"
@client = JenkinsApi::Client.new(:server_url =>
          'http://jenkinshost/jenkins/')
@client.job.list_all
```

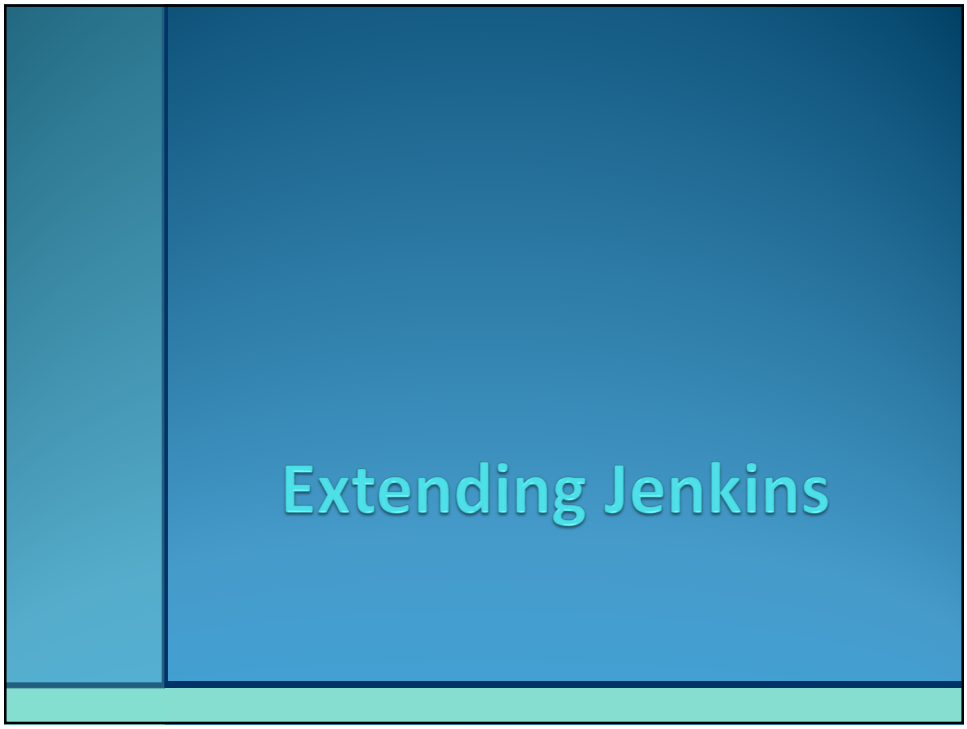


Jenkins Discovery

- Jenkins supports auto-discovery
  - UDP - Listens on port 33848 by default
    - Broadcast to 255.255.255.255
  - DNS multicast
    - Similar to how email servers or default web server are found
    - Use `_jenkins_tcp.somedomain.com`
  - All servers auto-report in their domain
    - Servers **.somedomain.com
    - http://discover-Jenkins.somedomain.com

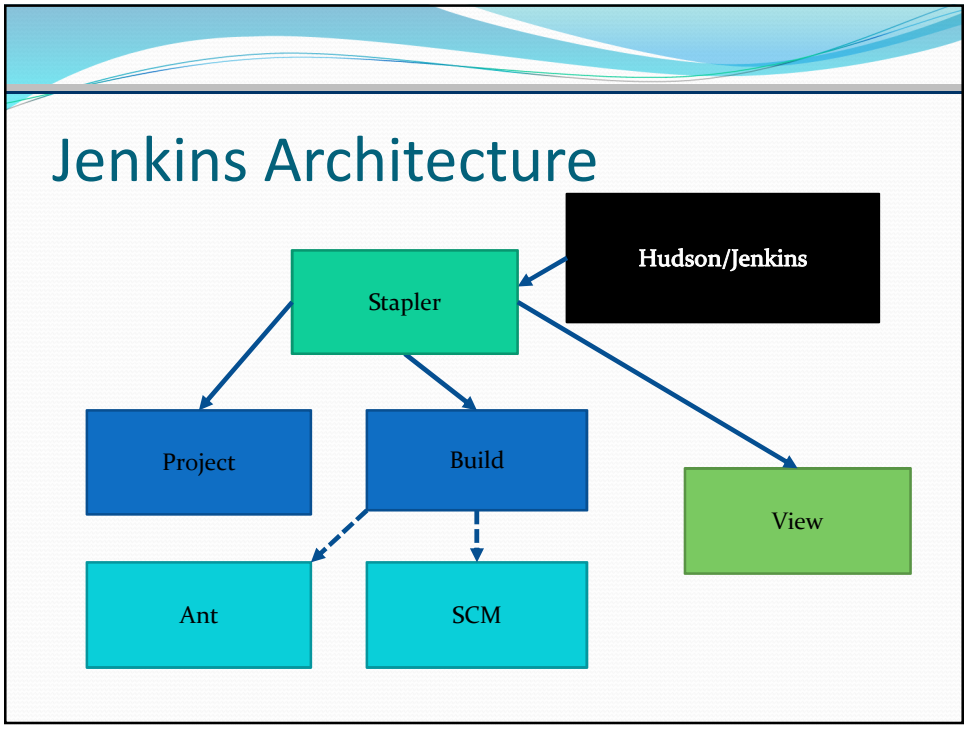
Summary

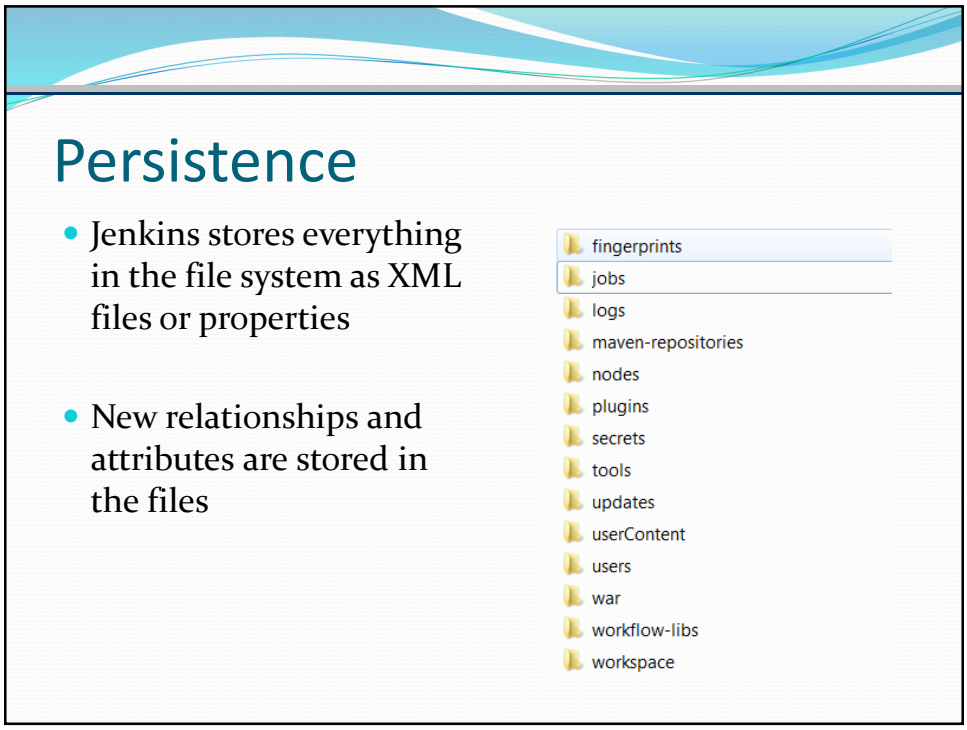
- Jenkins provides a very useful UI
- Organizations may have very standard job configurations
- Jenkins automates builds
- Builders can automate Jenkins
- Several API options
  - XML, JSON, Ruby or Python



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- **Extending Jenkins**
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins





Example Job

```
<?xml version='1.0' encoding='UTF-8'?>
<project>
  <actions/>
  <description></description>
  <keepDependencies>false</keepDependencies>
  <properties/>
  <scm class="hudson.scm.NullSCM"/>
  <canRoam>true</canRoam>
  <disabled>false</disabled>
  <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
  <authToken>abunchOfRandom123Numbers</authToken>
  <triggers/>
  <concurrentBuild>false</concurrentBuild>
  <builders/>
  <publishers/>
  <buildWrappers/>
</project>
```

Writing Command Line Utilities

- **GroovyCommand** executes a Groovy script on a master or slave
- **CLICommand** agent sends parameters to Jenkins server
  - Server creates **CLICommand**
  - Invokes **main(List, Locale, InputStream, PrintStream, PrintStream)**
- Enhance existing Jenkins modules with a CLI method
  - **@CLIMethod(name='somename')**
- Extend existing CLI commands
  - Add new methods
  - Extend behavior of existing methods
  - **@Extension** on CLI class being extended

Writing Plugins

- Seriously reconsider!
- Jenkins plugins are like Ant or Maven plugins
- Map a Java class into the context of a running Jenkins
- Access to jobs, views, configurations
- Dynamically loaded by Jenkins on startup
- Plugin exposes its attributes (also referred to as properties)
- Jenkins queries the attributes
- Jenkins dynamically builds pages to edit attributes
- Use Maven generator to build a prototype plugin

Activity: Types of Plugins

Question: What types of plugins have you developed?

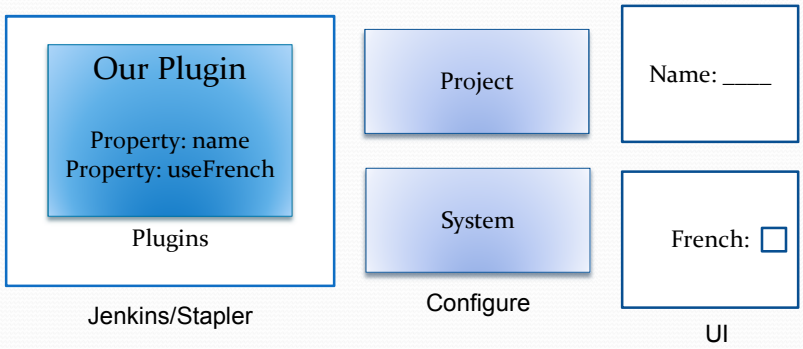
Instructions: Annotate your answer below.

Maven	Ant	Jenkins	Custom Tag

Plugins Interaction

- Map a Java class into the context of a running Jenkins
- Access to jobs, views, configurations
- Dynamically loaded by Jenkins on startup
- Plugin exposes its attributes
- Jenkins queries the attributes
- Jenkins dynamically builds pages to edit attributes

Plugin Architecture



Configure System sets the plugin property **useFrench**
Configure Project sets the plugin property **name**

Extending Existing Plugins

- Most Jenkins plugins hosted on GitHub
- Clone the repository
- Provided the enhanced behavior
- Test thoroughly
  - Plugins can be installed from **.hpi** in Jenkins using advanced options
- Push the changes back to GitHub
- Let the developer know what you changed and why

Exercise: Extending Jenkins

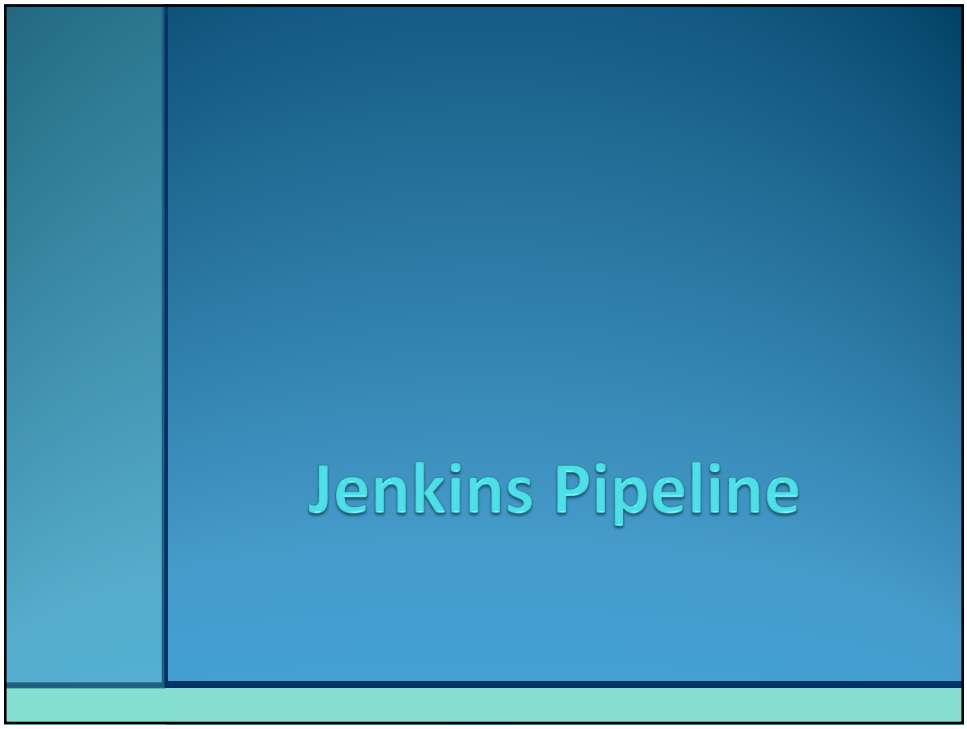
You will create a Jenkins plugin:

- Add repositories for Jenkins files to Maven
- Execute Maven command to generate plugin project
- Execute Maven install to build the plugin
- Install the plugin in Jenkins
- Restart Jenkins
- Create a new job
- Add a build step with plugin
- Perform the build and view the console output
- **Exercise Review**

Summary

- Hardly ever need to extend Jenkins
- If needed there are many ways
- Writing new or extending CLI commands
- Writing new or extending plugins

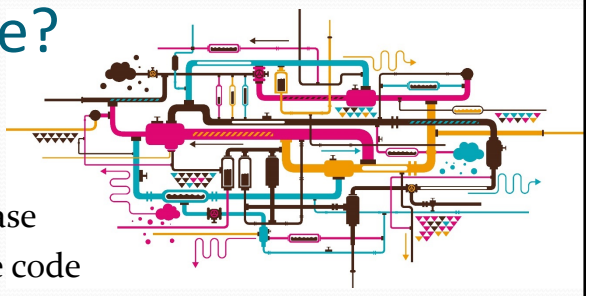
This slide has been intentionally left blank.



Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- **Jenkins Pipeline**
  - **Introduction**
  - **Quick Introduction to Apache Groovy**
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

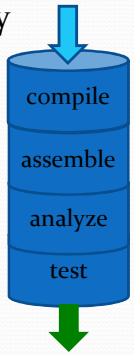
Why Pipeline?



- Build complexity continues to increase
- Developers manage code
- CI/CD folks manage jobs in Jenkins
- Builds should be in code not an application
- Builds need to be resumed after a restart
- Should be able to pause and resume a build
- Lastly, need a graphical representation of progress

What is a Pipeline?

- Formerly referred to as **Workflow** in earlier Jenkins versions
- Pipelines are a way of thinking of jobs with dependencies and viewing them graphically
- Ways to implement a pipeline
  - Write the pipeline code using Groovy
  - Use the Pipeline Builder plugin
  - Define build dependencies



Building a CD Pipeline

- Write the pipeline code using Groovy

```
node {
    git url: someurl
    def mvnHome = tool mvn
    env.PATH = ${mvnHome}/bin:${env.PATH}
    sh mvn -B verify
}
```

- Use the Pipeline Builder plugin

Activity: Groovy and Java

Question: Do you already know Java and/or Groovy?

Instructions: Use Pointer tool to mark your answer below.

Java	Groovy	Both Java and Groovy	Neither

Introduction to Groovy

- General purpose scripting language that runs on JVM
- Optionally typed – dynamic with static-typing and compilation capabilities
- Applies OO principles around Java classes
- Example in Java:

```
class Hello {
String name;
void sayHello() { println("hello world"+name); }
static void main(String[] args) {
       Hello hello = new Hello();
       hello.name = "world";
       hello.sayHello();
}
```

- Same example in Groovy:

```
def sayHello(name) { println("hello $name"); }
fef name="world"
sayHello(name);
```

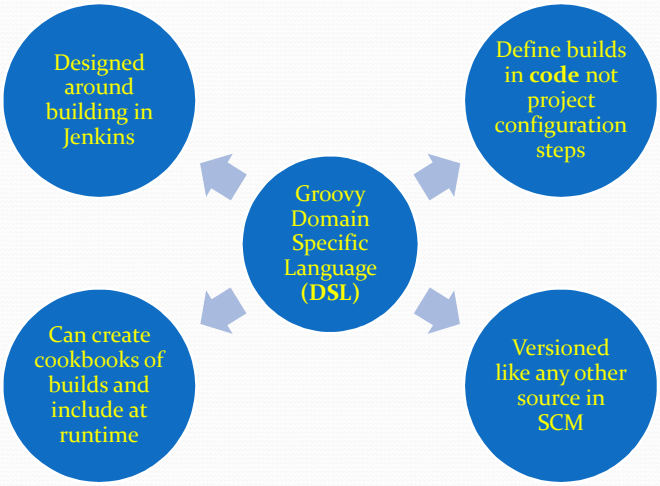
Syntax

- Comments like Java – also **shebang** line
- See www.groovy-lang.org/syntax.html
- Dot operates like Java
  - Separates **variable.keyword** or **variable.attribute**
- Can imbed Java directly
  - `def result = somejava() { return 3; }`
- **assert** some truth
- Triple quote string starts a string and continues to next
- `def lista = [1,2,3]` to define standard JDK List

Differences from Java

- Default imports more than Java
- Dynamic runtime binding to methods
- Curly braces are reserved for closure
- **Package** visibility by default vs. Java is **protected**
- Closures are similar to Lambda expression but different
- Inner classes are different from Java
  - Groovy doesn't support **y.new X()** syntax
- **GStrings** are like **Strings** but Strings with **$** will have errors in Groovy

Key Concepts for Pipeline



Activity: Pipeline Experience

Question: Have you ever made or used a pipeline?

Instruction: Put up a ✓ or X

Discussion:

- What did you use the pipeline for?
- Can you think of scenarios where pipelines would be useful?

Summary

- Builds are very complex - many small jobs
- Challenge to represent that visually
- Hard for others to understand
- Builds should be maintainable like the code being built
- Groovy DSL made build descriptions easy to read
- Can retrieve build script from SCM for versioning
- Pipeline views allow a graphical representation

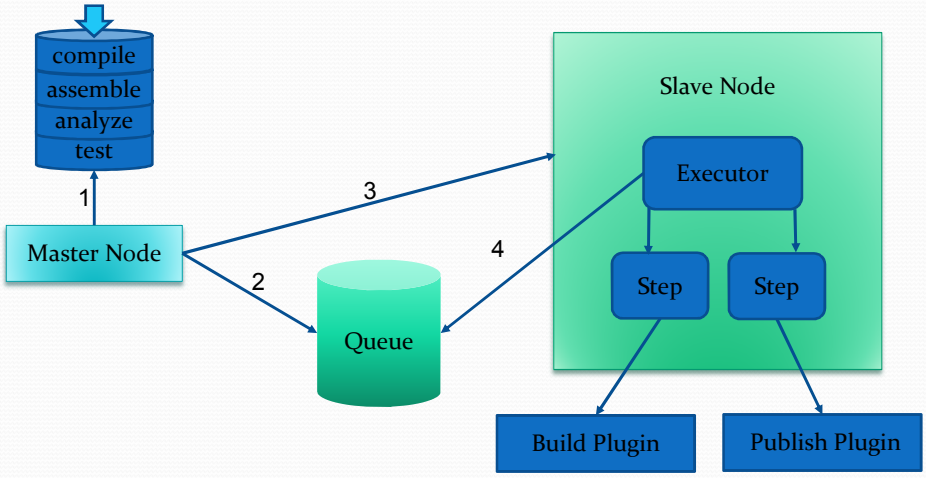
Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - **Jenkins Pipeline Continued**
  - Front-ending Jenkins Pipeline
- Best Practices for Jenkins

Build Steps

- Plugins add build or post-build functionality
- Step defines what plugin to use and what to do
- Publishers also can be told to accomplish something
- Some steps are very simple
  - `echo("some string");`
- Steps can fail or throw exceptions
- A node is a step that schedules a task in the build queue
  - Executor will execute when a slot is available

Build Steps Diagram



Example Build Script

```
node('remote') {
  git url: 'C:\\Setup\\git\\simple-maven-project-with-tests.git'
  def v = version()
  if (v) {
    echo "Building version ${v}"
  }
  def mvnHome = tool 'M3'
  bat "${mvnHome}\\bin\\mvn -B -Dmaven.test.failure.ignore verify"
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.jar',
        fingerprint: true])
  step([$class: 'JUnitResultArchiver', testResults: '**/target/surefire-
        reports/TEST-*.xml'])
}
def version() {
  def matcher = readFile('pom.xml') =~ '<version>(.+)</version>'
  matcher ? matcher[0][1] : null
}
```

Gathering Human Input

- Steps can sometimes need to request input from users
- Use the input step
  - Input waits for input from user
  - Define the message to prompt the user
  - Optionally accept something in response

- Executor pauses until the user interacts with the system

Poll

1. Strong motivators for pipeline include:
   A. It's cool
   B. Integrate with version control
   C. Graphical representation
   D. Encapsulation

2. Pipelines only run on the master.
   A. True
   B. False

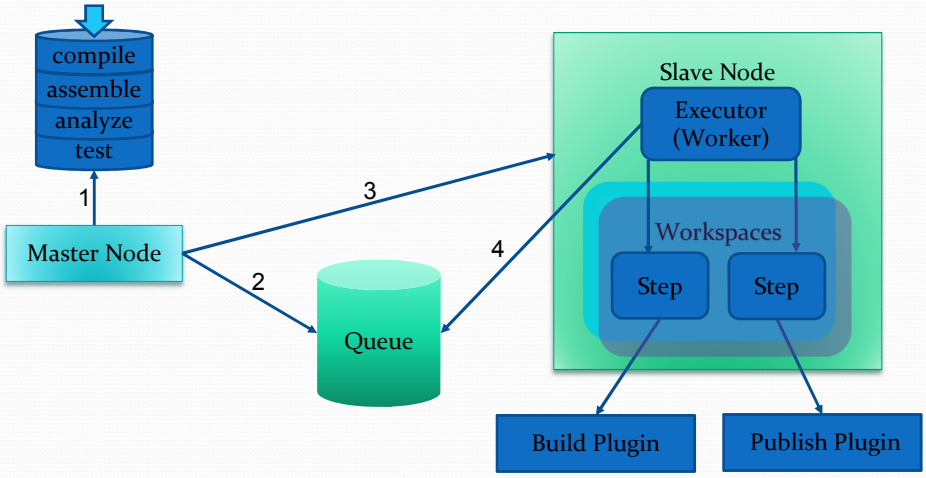
Allocating Projects and Workspaces

- Each executor processing a node will allocate a new workspace to execute in
- If a node supports multiple executors, a job can have multiple workspaces, shown as : `/slave/workspace/job@2`
- New workspaces can be allocates with a **ws** step
- The **dir** step can launch another step in a different directory

Using Distributed Workers

- Node step triggers an executor to grab a context to run
- The master runs a flyweight process to keep track of the pipeline but doesn't perform the build
- Specify node `('unix || mac') { }`
- Will launch the build in a node that matches

Workspaces and Workers Diagram



Summary

- Pipelines are a powerful CI/CD tool
- Orchestrate the build using nodes and steps
- Also take advantage of slaves for distributed builds
- Some steps may run from one repository
- Other steps may require content from a different repo
- Conditional logic is helpful to know what to do next
- Try/Catch handling can add additional functionality

Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - **Front-ending Jenkins Pipeline**
- Best Practices for Jenkins

Let's take a break before we move on

Remote API for Pipeline

- Remote API works the same for pipeline projects
- Query the job and get the results as XML or JSON
- Provide the script to run as attribute
- Configure parameters

Interacting with Pipeline API

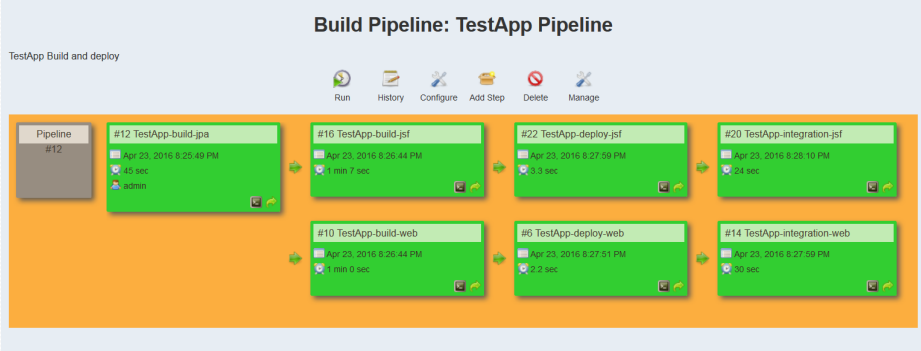
- Job **DSL** was designed for scripting job definitions
- **DSL scripts** can be stored in SCM
- Starts with job object and a closure to define the job
- If the job exists, it is updated
- If the job doesn't exist, it is created
- Usually started with a **Seed** job
- Can also execute as a Groovy Command from CLI

Sample Job DSL Script

```
def project = 'some.project'
def branchApi = new
URL("https://api.github.com/repos/${project}/branches")
def branches = new
groovy.json.JsonSlurper().parse(branchApi.newReader())
branches.each {
    def branchName = it.name
    def jobName = "${project}-${branchName}".replaceAll('/','-')
    job(jobName) {
        scm {
            git("git://github.com/${project}.git", branchName)
        }
        steps {
            maven("test -Dproject.name=${project}/${branchName}")
        }
    }
}
```

Build Pipeline Plugin

- Defines a graphical representation of jobs as pipeline
- Define the start job
- Job hierarchy reflects the jobs in the pipeline



Exercise: Pipeline

- Create a build pipeline view
- Create a job for a shared library
- Create jobs to build two web applications
- Create jobs to deploy the applications to Tomcat
- Create jobs to perform integration tests on both apps
- Add to the build pipeline view
- Run the initial job and watch the pipeline progress

- **Exercise Review**

Summary

- Jenkins has added amazing, powerful features
- Powerful APIs for creating job definitions
- Powerful tools to manipulate the jobs
- Better ways to view progress of jobs in Jenkins
- Better support for querying progress of jobs externally
- Pipeline greatly enhances the ability of Jenkins to control complex flows

This slide has been intentionally left blank.

Agenda:

- Introducing Continuous Integration and Jenkins
- Installing and Running Jenkins
- Simple Jenkins Job
- Securing Jenkins
- Advanced Jenkins
- Jenkins Plugins
- Distributed Jenkins
- Jenkins with Node.js
- Jenkins Remote API
- Extending Jenkins
- Jenkins Pipeline
  - Introduction
  - Quick Introduction to Apache Groovy
  - Jenkins Pipeline Continued
  - Front-ending Jenkins Pipeline
- **Best Practices for Jenkins**

General Best Practices

- Always Secure Jenkins
- Don't build on the Master
- Backup Jenkins Home Regularly
  - See previous best practice

"Save Early, Safe Often"

Projects and Dependencies

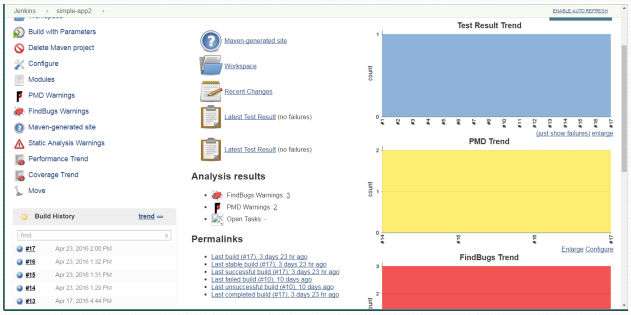
- Limit project names to alphanumeric
- Use file fingerprinting for dependencies
- Always build clean from source

SDLC Integration

- Integrate with the defect tracking system
- Integrate tightly with VCS

Traceability

- Keep the trends and the automation results
- Set up separate jobs per branch



Notification

- Enable email notification on builds to ALL developers
- Tag source on stable builds

Performance

- Give Jenkins LOTS of disk space
- Archive unused jobs before deleting them
- Avoid scheduling lots of jobs to start at the same time
- Create maintenance jobs to clean up periodically

Activity: Priorities

Question: Which of the best practices would be the highest priority for your processes?

Instructions: Annotate your answers below.

Notification	
Traceability	
Security	
Project complexity	
SDLC integration	
Performance	

Summary

- By incorporating Jenkins best practices, your CI process will produce quality build jobs:
  - Are secure and track which developers contribute failing code
  - Adhere to naming conventions
  - Integrate with version control and defect tracking systems
  - Provide visibility into the history of the project trends
  - Notify all developers on builds
  - Perform well by generous space and periodic maintenance

Course Wrap-up

Questions?

- Do you have any questions?
- Are there any topics you'd like me to review?
- Would you like more depth on any topic?
- Is there a related topic you'd like to know about?
- Official site: **https://jenkins.io**

Course Evaluations

- If you haven't already completed the course evaluation, please give us feedback regarding your impressions of this workshop:
  - Instructor
  - Materials
  - Equipment and facilities
  - or Virtual Experience

- We appreciate your feedback. Thank you!

This slide has been intentionally left blank.