Chapter 1 - Continuous Delivery and the Jenkins Pipeline

Objectives

Key objectives of this chapter

- Continuous Delivery
- The Jenkins Pipeline
- A Brief Introduction to Groovy
- The JenkinsFile
- Pipeline Jobs

1.1 Continuous Delivery

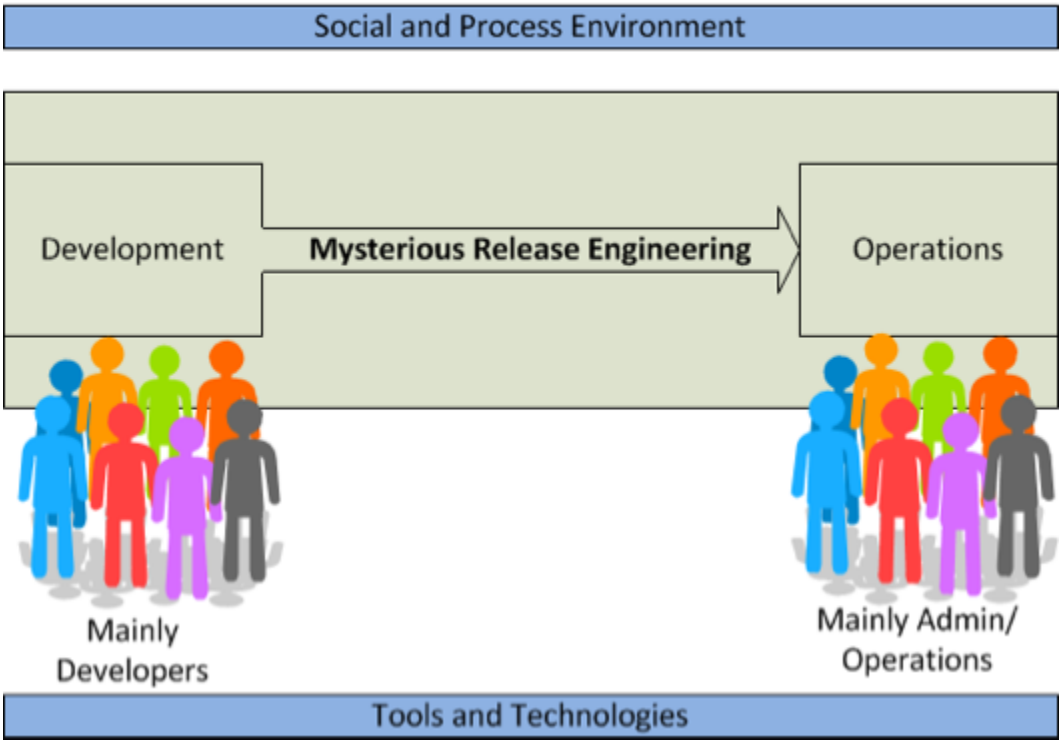
- CD extends CI into deployment and operations
- Agile/XP speeds up the development process
  - Include the Customer or Voice of the Customer
  - Ensure releasable artifact after every iteration
- Nonetheless, Agile/XP releases fit into the standard Software Development Life Cycle
  - A release engineering and deployment process follows the development process.
  - Reflects traditional split between development and operations

Continuous Delivery

Continuous Delivery, or CD extends the idea of continuous integration into the deployment and operations realm.

Although Agile Development and Extreme Programming seek to speed up the development process, they originally viewed development as one piece of the traditional Software Development Life Cycle (SDLC). XP tries to include the customer (or at least the "voice of the customer") in the development process, and tries to make sure that we have a deliverable artifact at the end of every iteration. Nonetheless, there's an expectation that a formal release process will follow that iteration. This model reflects the traditional split between development and operations.

1.2 Continuous Delivery (cont'd)



Continuous Delivery (cont'd)

Development creates a system. Operations installs and runs it. In between, we have some mysterious release engineering process that usually includes some form of quality assurance, user acceptance testing and management sign-offs.

1.3 DevOps and Continuous Delivery

- Managing deployments and development across horizontally-scaled environments is difficult
  - Requires tight control of the process
  - Requires automation of development, deployment and monitoring
  - Along with the technologies of virtualization, cloud, etc.
- This is the central theme of DevOps - Integrated provision of services using appropriate technology and processes
- Deploy software as fast as we create it, while maintaining quality, trace-ability and accountability

DevOps and Continuous Delivery

Managing application deployments and rapid application development across these sophisticated, horizontally-scaled environments requires tight control of the process and significant automation, as reflected in the "DevOps" movement. Much of this effort is aimed at being able to deploy software as fast as we develop it, while maintaining quality, trace-ability, and accountability.

1.4 Continuous Delivery Challenges

- More than one department/group involved, not just Developers
  - QA, Compliance, Business Customers, etc
- CI job takes a few minutes, CD process could extend over days
  - Could also include human input, manual tests, acceptance tests, etc

- Extended cycle means we have multiple process instances "in flight"
  - "Version 6" part-way through User Acceptance Testing, while work continues on "Version 7"
- Multiple resources and test/deployment environments involved
  - "Version 6" goes on to Performance Validation
  - "Version 7" moves into User Acceptance Testing
  - Meanwhile, development gets going on "Version 8"

Continuous Delivery Challenges

Continuous Delivery extends the ideas of Continuous Integration to cover the release engineering process and deployment of software into the production operating environment.

From an automation perspective, things change a little when we look to Continuous Delivery.

- There is probably more than one group involved. Not just development, but perhaps product management, quality assurance, regulatory compliance and others.
- Whereas CI can be automated to be essentially a batch job, taking at most a few minutes, the release engineering process followed by deployment may take days.
- Because of the extended cycle, we are more likely to have multiple instances of the process "in flight" at any given time.
- The multiple instances of the release process are likely to include multiple resources and deployment environments.

1.5 Continuous Delivery with Jenkins

- Jenkins has always been useful for this
  - A job completed successfully can trigger another job
  - Jobs can be run on different machines
  - Parameterized jobs can gather information from users
  - Plugins can manage deployment
- Workflow processing has been difficult though
  - Builds in-process don't persist across restart
  - No logic in linking jobs
  - No unified user interface
  - Job definitions are not in Version Control
- Solution: The Pipeline Plugin

Continuous Delivery with Jenkins

Jenkins has always been capable of performing some of the work in a Continuous Delivery system.

We can have one job trigger another job. With distributed Jenkins, we can have different jobs done on different machines. Parameterized jobs can gather information from users. Jenkins plugins can be used to manage deployments and so on. But up until recently, there hasn't been a way to really represent that the release engineering activities form a business process, with the usual elements of a workflow processor. Also, it hasn't been possible to do logic in the selection and execution of jobs and processes.

In other words, Jenkins has handled Continuous Integration admirably, but Continuous Delivery has been a challenge, mainly because the process of Continuous Delivery has such a longer duration than a typical Continuous Integration job

The Jenkins Pipeline plugin (and associated plugins) is intended to address these challenges.

1.6 The Pipeline Plugin

- Represents a workflow
- More like a business process in SOA
  - whereas a "Job" is more like a "Service Operation"
- Define a process and then execute an "instance"
- Completion of the process can span more than one machine
- Process has "State"
  - "where are we in the process"
  - Process variables
- State is stored persistently
  - Also distributed when the build uses a different machine

1.7 The Pipeline Plugin (cont'd)

- Steps can be executed conditionally
- Can have explicit parallelism
  - And can span machines
- Definition of Pipeline can be stored in Version Control
  - Including all the steps that would make traditional "jobs"
- User Interface
  - lets us see multiple instances
  - See processes that span distributed installations
  - Interact with the process (input and output)

1.8 Defining a Pipeline

- The pipeline is defined by a script written in the Groovy programming language
  - http://groovy-lang.org
- The script defines "steps" that the Pipeline plugin executes
- Steps:
  - built-in steps like "checkout" , "sh" or "bat"
  - Calls to plugins
  - Calls to existing jobs

1.9 A Pipeline Example

```
node {
   stage 'Checkout'
   checkout scm

   stage 'Build'
   bat 'nuget restore SolutionName.sln'
   bat "\"${tool 'MSBuild'}\" SolutionName.sln"

   stage 'Archive'
   archive 'ProjectName/bin/Release/**'

}
```

1.10 Pipeline Example (cont'd)

- 'node' indicates a section that should be run on an Agent machine
- 'stage' defines a pipeline stage
  - for presentation in the UI
- 'checkout' invokes the SCM plugin to checkout a copy of the project on the node in question
  - Note: this file would be stored in version control and checked-out by the Jenkins master. 'checkout' is checking out the same project on the execution node
- 'bat' runs a Windows batch file
- 'archive' stores the build results

1.11 Parallel Execution

- Sections of the pipeline can be executed in parallel

```
def labels = ['precise', 'trusty'] // labels for
Jenkins node types we will build on
def builders = [:]
for (x in labels) {
    def label = x // Need to bind the label variable
before the closure - can't do 'for (label in
labels)'

    // Create a map to pass in to the 'parallel'
step so we can fire all the builds at once
    builders[label] = {
      node(label) {
```

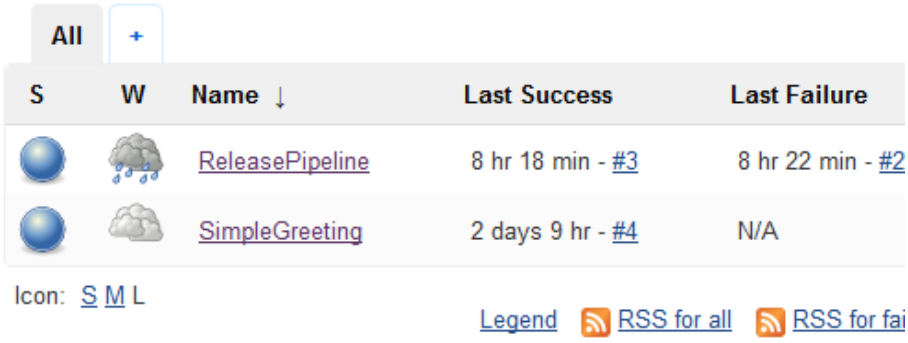
```
        // build steps that should happen on all nodes
go here
          }
        }
}

parallel builders
```

1.12 Creating a Pipeline

- Various ways
  - From the UI, like a Job
    - Enter pipeline script into the web interface
    - This is nice because of syntax help available
  - In a "Jenkinsfile"
    - Stored in version control
    - Could be in the same repository as a project, or in its own repository
  - Commonly-used scripts and functions can be stored in a "Global Library"
    - Jenkins runs an instance of 'Git' to let you push global libraries that are then available to all Pipeline scripts.
  - Libraries can be stored in the project and loaded as needed.

1.13 Invoking the Pipeline

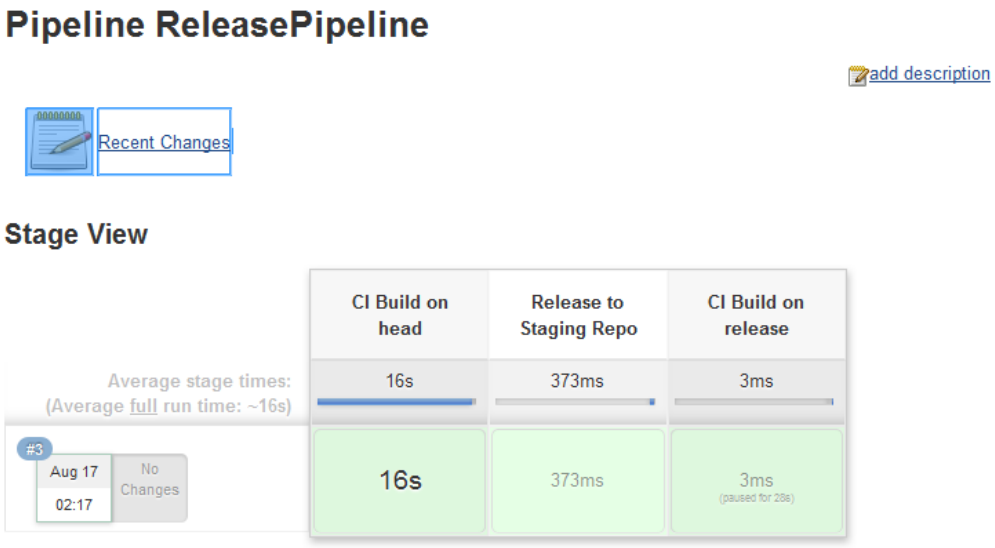
- The pipeline shows up just like a regular Jenkins job



- It can be invoked manually, at a predefined interval, or based on SCM polling
  - Just like a regular job

1.14 Interacting with the Pipeline

- Stages in the pipeline appear on the user interface



- In case of input required or output, these appear on the UI

1.15 Conclusion

- The Pipeline plugin turns Jenkins into an Orchestration system
- Continuous Delivery processes may span extended time periods
- Pipeline definition is in the form of a Groovy script
- The Jenkins UI lets you interact with the Pipeline