WA2753 Custom TD DevOps bootcamp

Web Age Solutions Inc.
USA: 1-877-517-6540
Canada: 1-866-206-4644
Web: http://www.webagesolutions.com

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

IBM, WebSphere, DB2 and Tivoli are trademarks of the International Business Machines Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Table of Contents

Chapter 1 - What is DevOps

Objectives
Key objectives of this chapter

Key objectives of this chapter

- DevOps introduction
- Business value of DevOps
- Standing up DevOps capability

1.1 Dev and Ops Views

The Dev View	The Ops View
• We have aggressive deadlines -- Business is all over us • Ops are much too sluggish supporting us (provisioning integration environment, etc.) • They lost the application zip file we emailed them yesterday night -- it was eventually found in their "junk mail" folder • Overall, we don't have trust and confidence in Operations (Ops) -- they are more like *Oops*, then *Ops*	• Dev is all over us • The application set-up guide sent by Dev was not complete -- they missed some critical steps, which resulted in our wasted time • With so many new applications being released in the environment, we can no longer guarantee uninterrupted services • Overall, we don't trust Dev

1.2 Leading By Example ...

1.3 What is DevOps?

- DevOps is short for **Dev**elopment and **Op**erations

- It is an approach to delivering software solutions in a continuous manner based on lean (minimizing waste of resources, reducing number of defects, etc.) and agile practices

- DevOps help manage complexities of Enterprise applications by creating a collaborative environment with participants coming not only from Development and Operations, but also from Business, QA, and other stakeholder groups

  ◇ In other words, DevOps is not only about Development and Operations!

- The DevOps practice has been popularized by organizations adopting the Cloud-as-a-Service computing model

1.4 More DevOps Definitions

- You can view DevOps as

  - a culture, or

  - a cross-team software delivery discipline (paradigm)

that tries to reconcile competing perspectives (e.g. those of Dev vs Ops) and promote collaboration by stepping over the silos of isolated, group-centric interests

1.5 DevOps and Software Delivery Life Cycle

- To efficiently increase the velocity of application delivery, DevOps activities span the whole software delivery life cycle (not only its deployment!):

  ◇ Development

  ◇ Testing

  ◇ Deployment

  ◇ Operation

1.6 Main DevOps' Objectives

- Continuous software delivery planning and control

- Software delivery processes optimization

- Software delivery process consistency and predictability

- Minimization of the number of software defects and unnecessary re-work

- Software delivery cycle time reduction

1.7 The Term "DevOps" is Evolving!

- Originally, DevOps was used to refer to the practice adopted by Operations to borrow some of the tools and processes used in software development
  - ◇ For example:
    - Admin scripts were placed in a version control system
- Now *DevOps* is used in the context of the shared responsibility between Development and Operations for delivering high quality software products
  - ◇ For that, where appropriate, the reporting lines are merged and simplified
  - ◇ Development goals (introduce code, configuration, etc. changes to the system) are aligned with those of Operations (maintain the target system stability)
- DevOps nowadays is becoming more embracing being not only about *Tools*, but also about *People* and *Processes*

1.8 Infrastructure as Code

- "Infrastructure as Code" is a practice of provisioning infrastructure by executing system management and configuration scripts

- Under DevOps, Dev is granted system administration privileges to run the infrastructure set-up scripts to automatically provision the necessary development and testing environments

  - Provisioning of other environments (staging, production) may still be the exclusive prerogative of personnel in the DevOps' Operations role

- *Infrastructure as Code* is effectively supported by such tools as Chef, Puppet, and IBM UrbanCode Deploy

1.9 Agile IT in the Cloud

- Cloud facilitates agile IT practices that allow businesses to quickly respond to market forces

- IT responds to rapidly changing business requirements by creating a cloud-based execution environment that allows effective and optimized reuse of existing services through

  - re-configuration,

  - re-composition, and

  - introducing new services that also lend themselves to composition and reuse

1.10 DevOps on the Cloud

- Some of the more important activities performed by Operations are related to provisioning computing resources

- DevOps agility can been dramatically enhanced with adopting Cloud Computing

  - ◇ Developers can easily self-provision the needed resources (virtual servers, extra disk storage, back-end systems, etc.)

  - ◇ Many cloud platforms allow for application code snapshotting which can be used for environment replication / cloning (Dev → QA → UAT → Prod)

    - This feature also facilitates defect reproduction

  - ◇ In the cloud, it is much easier to quickly set up and tear down "Production-like" systems at a minimal cost

1.11 Prerequisites for DevOps Success

- A good relationship between Dev and Ops is necessary but not sufficient for the overall success of DevOps operations

- DevOps success depends on the proper execution of all the technical aspects of the Software Development Life Cycle (SDLC) phases and steps established in the organization

  - Sometimes, the existing SDLC documentation needs to be adjusted to make it aligned with the agile practices used by DevOps

- The following elements and capabilities need to be put in place for DevOps to meet its objectives:

  - Alignment with the business needs

  - Collaborative development

  - Continuous testing and integration

  - Continuous release and deployment

  - Continuous application monitoring

1.12 Alignment with the Business Needs

- In essences, DevOps is a business-driven software solutions delivery process

- The DevOps practice is instrumental in reliably materializing a business idea in a software product, ultimately delivering value to customers

- DevOps processes improve product time-to-market metric (faster time to value) and enable organizations to react to new market demands more quickly

- With DevOps, customer feedback on product is captured and quickly incorporated in the next iteration of the product delivered in a continues manner

  ◇ The ultimate result of a fast accommodation of the customer feedback is an enhanced customer experience, customer loyalty, and larger market share

1.13 Collaborative Development

- High quality software development is predicated on the collaborating development practices, including:

  ◇ Development teams work is done in accordance with established code standards, styles, and <u>living</u> centralized developer documentation (wiki pages, etc.)

    - Development teams may be geographically dispersed or formed at the last moment, making the above practice indispensable

  ◇ The stewards of the target system's architecture and design are cooperating with development to quickly accommodate any design flaws / gaps identified during development

1.14 Continuous Testing and Integration

- Development of discrete software components must go hand-in-hand with the development and application of appropriate unit tests as prescribed by the Test-Driven Development (TDD) process

    ◇ TDD is an agile software development practice

- Integration of software components must start as early as possible (even though the work on components may not yet been fully completed) and conducted frequently (sometimes, several times a day)

    ◇ This process is known as the Continuous Integration (CI) agile practice which helps with catching integration problems early

1.15 Continuous Release and Deployment

- Deployment has always been one of the primary activities of Operations

    ◇ With the advent of the Cloud Computing, Development can take over some workload of application deployment and perform self-provisioning of cloud computing resources they require

- To support reliable continuous releases, the deployment process must be automated; any failed deployments must be rolled back in its entirety in an atomic operation without affecting applications currently running in production

- Deployment parameters, such as average deployment time, size of the deployment bundle, etc., must be recorded and kept for reference

1.16 Continuous Application Monitoring

- Application run-time behavior monitoring should begin in production-like environments where the application would have setup, configuration, and other parameters close to those used in production

- Things to look for:

    - Run-time application behavior (CPU, RAM, I/O, average duration of garbage collection pauses, if applicable, and other metrics)

    - Response time per application interface

    - Excessive or insufficient logging

    - Logging of sensitive information

    - etc.

1.17 Benefits of DevOps

- Automation

- Faster resolution of problems

- Faster delivery of features

- More stable operating environments

- Versionable, testable, and repeatable code as infrastructure

- Aligning IT and Business

- Breaking down the silos

- Improved customer experience and satisfaction

- Reduced risk to change

1.18 What is Involved in DevOps

- Code Versioning

- Configuration Management

- Continuous Build

- Continuous Quality Assurance

- Continuous Deployment

- Continuous Monitoring

- Automation

1.19 Summary

- DevOps can be viewed as a cross-team software delivery discipline (paradigm)

- The DevOps practice has been popularized by organizations adopting the Cloud-as-a-Service computing model

- The following elements and capabilities need to be put in place for DevOps to meet its objectives:

  ◇ Alignment with the business needs

  ◇ Collaborative development

  ◇ Continuous testing and integration

  ◇ Continuous release and deployment

  ◇ Continuous application monitoring

Chapter 2 - Configuration Management

Objectives
Key objectives of this chapter

Key objectives of this chapter

- Understanding Chef
- Installing Chef
- Creating Cookbooks

2.1 What is Chef?

- Software developed by Opscode community in 2008

- Configuration management software / automation platform

- Automates application configuration, deployment and management across network

- Infrastructure as a code

- DevOps platform / Continuous delivery (Build → Test → Deploy)

- Written in Ruby and Erlang

- Alternative to Chef – Puppet, CFEngine

2.2 Benefits of Infrastructure-as-Code

- Repeatable

- Versionable

- Testable

2.3 Chef – Sample Usages

- Set up environments
  - ◇ development
  - ◇ test
  - ◇ production
- Set up machines
  - ◇ physical machine
  - ◇ cloud node
  - ◇ hypervisor / virtual machine / container
- Install Operating System
- Download Software
- Install Packages / Software
- Create Configuration Files (e.g. .conf, web.config, app.config, .ini, ...)
- Upload Files
- Deploy Software / Application
- Run Scripts

2.4 Deployment / License

- Open Source

- Hosted Chef

- Private Chef

2.5 Who uses Chef

- Adobe

- Alaska

- Facebook

- IBM

- Intuit

- Microsoft

- …

2.6 Chef Architecture
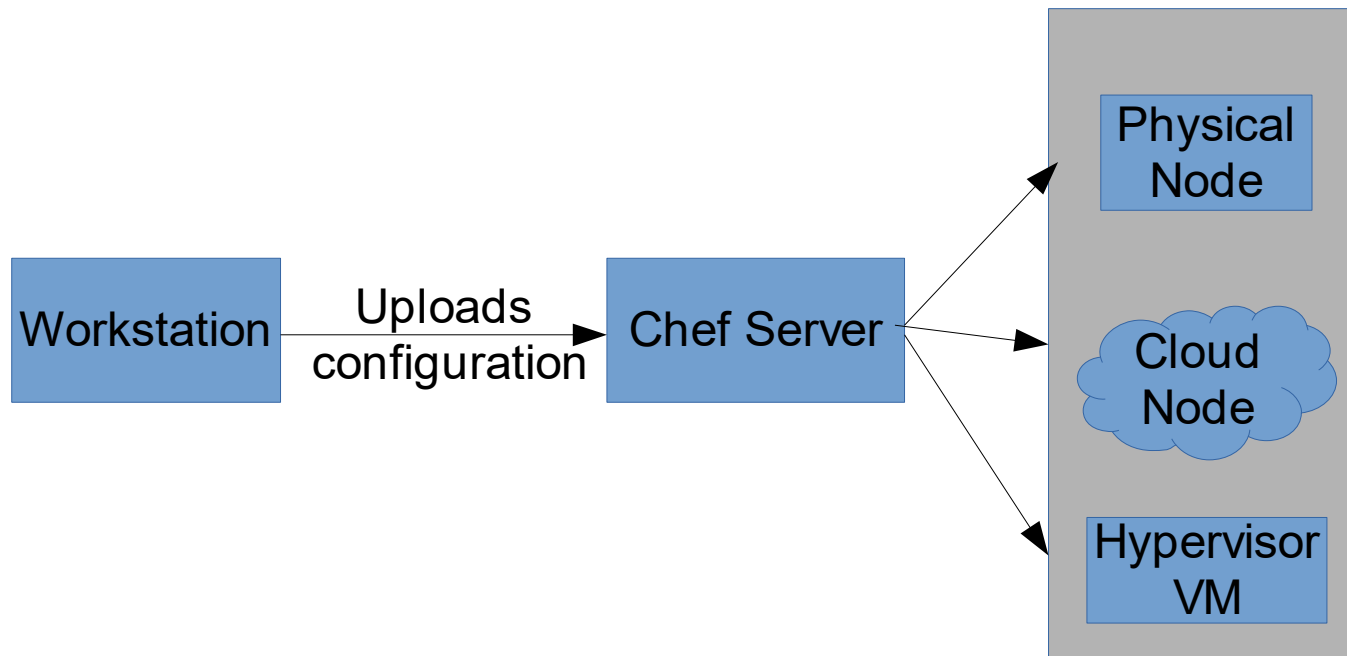
2.7 Chef Components

- Chef server
  - ◇ Cookbooks
    - Recipes
      - Actions
- Workstation
  - ◇ Knife
- Node(s)
- Ruby
- Chef Analytics
- Chef Supermarket

2.8 Workstation

- One (or more) workstations are configured

- Author, test and maintain cookbooks

- Can be outside of Chef service installation, machine where Chef server is configured, or one of the managed nodes.

- Keep chef-rep synchronized with version source control

- Command-line tools

- Interacting with nodes

2.9 Recipe

- A script that that describes a set of steps

- e.g.
  - ◇ Deployment of your custom software
  - ◇ Provisioning of a database
  - ◇ Adding a host to load balancer

2.10 Cookbook

- Collection of recipes

- Recipes are used for configuration items like packages, files, services, templates and users.

- Custom cookbooks

- Community cookbooks available form Chef Supermarket

2.11 Ruby

- Chef is a DSL that extends Ruby
- Authoring syntax for cookbooks

2.12 Knife

- Installed on a workstation

- Command-line tool

- Used to interact with nodes or with objects (cookbooks etc.) on the Chef server

- Plugins available to allow interaction with external cloud-based service, such as AWS, OpenStack, Google, Microsoft Azure, Google Compute Engine or Rackspace.

2.13 Node

- Any machine – physical, virtual, cloud, network device
- Chef-client is installed on every node

2.14 Chef-client

- Installed on every node

- Performs all of the configuration tasks specified in the run-list

- Pulled down from Chef server as it is needed during the chef-client run

2.15 Chef Server

- A hub of information

- Stores cookbooks

- Chef-client accesses the Chef server from the node on which it's installed

- Chef-client uploads updated run data to the Chef server, as the updated node object, when client run completes

2.16 Chef Analytics

- Used for reporting

- What changed, who made those changes and when the changes occurred

- Details are uploaded to the Chef server at the end of the chef-client run

2.17 Chef Supermarket

- Location in which community cookbooks are authored and maintained

- Can be used by any user

2.18 Salient Features of Chef

- Highly scalable

- Secure

- Fault-tolerant

- Integration with third-part tools

- Open Source

- Easy to Learn

- Cloud Friendly

- DevOps Friendly

- Strong Community Support

- Support for Major Platforms

- Allows scripts testing without actually running them on Chef server

2.19 Supported Platforms

- Ubuntu

- Debian

- RHEL/CentOS

- Fedora

- Mac OS X

- Windows / Windows Server

2.20 Chef Components

- Chef has 3 major components:
  - workstation
  - Chef server
  - node(s)

2.21 Chef Server prerequisites

- An x86_64 compatible system architecture

- A resolvable hostname that is specified using a FQDN or an IP address

- A connection to Network Time Protocol (NTP) to prevent clock drift

- A local mail transfer agent that allows the Chef server to send email notifications

- Using cron and the /etc/cron.d directory for periodic maintenance tasks

- Disabling the Apache Qpid daemon on CentOS and Red Hat systems

- Optional: A local user account under which services will run, a local user account for PostgreSQL, and a group account under which services will run.

2.22 Install Configuration Scenarios

- Standalone (everything on a single machine)

- High availability (machines configured for front-end and back-end, allowing for failover on the back-end and load-balancing on the front-end, as required)

- Tiered (machines configured for front-end and back-end, with a single back-end and load-balancing on the front-end, as required)

2.23 Standalone Installation

- Download the package from http://downloads.chef.io/chef-server/ on machine that will act as Chef server

- As a root user, install the Chef server package on the server, using the name of the package provided by Chef

  ```
  $ dpkg -i /tmp/chef-server-core-<version>.deb
  ```
- Run all services

  ```
  $ chef-server-ctl reconfigure
  ```
- Create administrator and save RSA private key

  ```
  $ chef-server-ctl user-create USER_NAME FIRST_NAME LAST_NAME
  EMAIL 'PASSWORD' --filename FILE_NAME
  ```
- Create an organization (used to authenticate the newly-created node to the Chef server during the initial chef-client run)

  ```
  $ chef-server-ctl org-create short_name 'full_organization_name'
  --association_user user_name --filename ORGANIZATION-validator.pem
  ```

2.24 Installing Optional Chef Server Components

- Chef Manage (Management console to manage run-lists, cookbooks etc from a web user interface)

```
$ chef-server-ctl install chef-manage
$ chef-server-ctl reconfigure
$ chef-manage-ctl reconfigure
```

- Reporting (view status of each Chef-lient run)

```
$ chef-server-ctl install opscode-reporting
$ chef-server-ctl reconfigure
$ opscode-reporting-ctl reconfigure
```

2.25 Workstation

- At least one workstation is configured

- Can be a separate machine, same machine where Chef server is installed, or one of the managed nodes.

- Chef Development Kit is installed on a workstation

2.26 Chef DK

- Contains:

  - Cookbook dependency manager – Berkshelf

  - Integration Testing Framework – Test Kitchen

  - Unit Testing cookbooks - ChefSpec

  - Linting tool for static code analysis on cookbooks – Foodcritic

  - Connecting to node(s) – Knife

2.27 Chef DK Prerequisites

- A computer running UNIX, Linux, Mac OS X or Microsoft Windows

- Apple XCode is installed on machines running Mac OS X; this application can be downloaded from Apple for free

- A GitHub account for downloading and/or cloning Chef repository from GitHub

- Access to a Chef server

- Access to a machine (physical or virtual) for bootstrapping a node

- Ruby – scripts / cookbook recipes are written in Ruby

2.28 Chef Repository

- Chef-repo is a directory structure on workstation

- Stores cookbooks, environments, configuration files etc

- Always version controlled

2.29 Installing Chef DK

- Download Chef DK from http://downloads.chef.io/chef-dk/

- Install the Chef DK package

- Verify Chef DK installation

```
$ chef verify
```
- Set Chef DK's Ruby as the default Ruby

```
echo 'eval "$(chef shell-init bash)"' >> ~/.bash_profile
```

2.30 Ohai

- Ohai is required by the chef-client and must be present on a node

- A tool that detects and provides attributes to the chef-client at the start of every chef-client run

- Ohai is installed on a node as part of the chef-client install process

- Collects data for many platforms, including AIX, Darwin, HP-UX, Linux, FreeBSD, OpenBSD, NetBSD, Solaris, and Windows.

- Returned data is in JSON format

2.31 Ohai Attributes

- Ohai collects various attributes, such as, platform, configured languages etc.

```
ohai
ohai current_user
ohai hostname
ohai ipaddress
ohai languages/ruby/version
```

2.32 Cookbooks

- One of the main components of Chef.

- A collection of recipes

- Each cookbook is a collection of directories and files that describe the cookbook and it's recipes and functionality

2.33 Components of a Cookbook

- Cookbook Metadata

- Attributes

- Recipes

- Templates

- Definitions

- Resources

- Providers

- Ruby Libraries

- Support Files

}

2.34 Metadata

- Each cookbook contains a metadata.rb file

- It contains information such as who maintains it, the license, version, contained recipes, supported platforms, and the cookbook's dependencies.

- JSON file is generated from the metadata

2.35 Recipes

- Script that defines what functionality should become available

- Written in Ruby

- Recipes are placed in recipes directory of a cookbook

- A recipe can be any functionality, such as provisioning accounts, installing and configuring a database server, create a file, and custom software deployments

- Combines configuration data with the current state of the host to execute commands that will cause system to enter a new state

2.36 Resources

- Anything that can be manipulated by Chef

- Built-in resources includes Cron jobs, deployments, file system components, source code repositories, logs, PowerShell scripts, Shell scripts, Templates, Packages, and Users and groups

- Syntax:

```
resource_name "name attribute" do
   attribute "value"
   attribute "value"
end
```

2.37 Directory Resource

■ Used for manipulating directories

```
directory '/tmp/folder' do
 owner 'root'
 mode '0775'
 action :create
end
```

2.38 Package Resource

- Used for package manipulation

- Default action is install

```
# specifying action explicitly
package 'httpd' do
 action :install
end

# using default action
package 'httpd'
```

2.39 Service Resource

- Used for service manipulation

- Default action is start

```
service 'httpd' do
 action [ :enable, :start]
end
```

2.40 File Resource

- Used for file manipulating

- Default action is create

- Often used for creating configuration files

- Code can be made cleaner by using a template resource

```
file '/etc/motd' do
 content 'Hello, world!'
end
```

2.41 Script Resource

- Used for executing scripts using a specified interpreter, such as Bash, csh, Perl, Python, or Ruby

```
script 'myscript' do
 interpreter "bash"
 cwd '/tmp/'
 code <<-EOH
  mkdir 'test'
 EOH
end
```

2.42 User Resource

```
user "bob" do
  action :create
  system true
  comment "Agent Bob"
  uid  1000
  gid  "agents"
  home "/home/bob"
  shell "/bin/bash"
end
```

2.43 Additional Chef Advanced Features

- Roles

- Environments

- Test Kitchen

- RuboCop

- Foodcitric

2.44 Summary

- Chef is used for implementing infrastructure as code which can be versioned, repeated, and tested.

Chapter 3 - Version Control

Objectives

Key objectives of this chapter
 	
   ■ Understanding Version Control
 	
   ■ Understanding Git

3.1 What is Version Control

- AKA Source Code Control

- AKA Software Configuration Management

- In the beginning, there was SneakerNet

  - ◊ Surprisingly common even now

3.2 What is Version Control (cont'd)

- Version Control can be a lot of things
  - ◇ "Undo" Capability
  - ◇ Collaboration
  - ◇ Communications and Sharing
  - ◇ Audit and Tracking
  - ◇ Release Engineering, Maintenance, SDLC
  - ◇ Diagnostics when things go wrong

3.3 What is Version Control (cont'd)

- Side-Trip: Locking vs Merging

    - One early criticism of systems like CVS (and this carries on into SVN and git) is that there is no "locking".

    - We don't get exclusivity when we check out a working copy.

    - Deal with conflicts at merge time

    - So, no warning if two developers work on the same file!

    - But in reality, this doesn't usually happen.

    - On the other hand, no stuck locks.

3.4 History of Version Control

- Tapes, Disk Packs, Floppies, etc

- Shared Folders

- RCS

- SCCS

- CVS

- SubVersion

- All these generally work on the basis of having a "master" copy

3.5 "Undo" Capability

- Consider a simple project with one developer...

- Project includes a number of files on disk under a given folder

- Any given file will be modified multiple times over the project lifetime

- Every time you edit, the old file contents are overwritten

- You might make a change that doesn't work out

  ◇ Would like to "Undo" the changes

- Changes usually don't just cover one file

  ◇ Modern software has a lot of components!

- Version control tells us what files have changed, can revert easily

3.6 Collaboration

- What if we have more than one developer?

- Developers need to share their work

- Can't just work out of a shared directory

    - You're working on multiple files at once

    - Software spends a lot of time in an "unbuildable" state

        - one or more files is "unfinished"

    - If we had multiple developers in a shared folder, it would almost never be in a "finished' state

3.7 Collaboration (Cont'd)

- The usual pattern:
  - ◇ Buildable --> Unbuildable --> Buildable
- Start with a build that works
- Move through a construction/debug phase
- Return to a buildable state
- Share the new state with other developers
  - ◇ Of course, your new "buildable" state might break their "buildable" state.
    - Might require an integration or "merge" step
  - ◇ Not just developers - CI systems, repositories, downstream users, etc.

3.8 Communication and Sharing

- Version control can be a quick and easy way to move working copies around

- Let's say an employee works from home occasionally

  ◊ VC gives an easy way to synchronize code between a desktop and a laptop machine

- Developers in different locations?

  ◊ VC provides a communications point

- Developers on different operating systems?

  ◊ VC can handle "trivial" issues like end-of-line character differences between operating systems

3.9 Auditing and Tracking

- Occasionally, you need to figure out "who did what?"

- or, "Where did that code come from?"

- Sometimes a "feature" turns out to be a bug, and we need to evaluate the impact

- Version Control provides a record of when code was brought in to a project, and who imported it

  ◇ This gets a little challenging with DVCS, but there are ways...

3.10 Release Engineering, Maintenance, SDLC

- Software projects are complicated

- Many steps take a long time

  - QA

  - Load Testing

  - Security Audits

- Often have a "released" version and an "in-development" version

  - Issues in the released version need to be fixed

    - Must be able to replicate the release!

  - Fixed issues need to be applied to the "in-development" version

    - but in a controlled way - need to wait til the "in-development" version is stable, then apply patches

3.11 Diagnostics

- Let's say you identify an issue

  - ◇ further, let's say you have a test that verifies the issue exists

- You need to know what code causes the issue, and fix it

- You could troubleshoot the behavior

- Sometimes, it's easier to ask "when did this problem start happening"

  - ◇ And then see what code was introduced around that time

- In those cases, a chronological record of how the code changed is invaluable

3.12 Distributed Version Control

- Latest development is Distributed Version Control Systems

  - Git

  - Mercurial

  - Darcs

- These reflect the distributed nature of projects

  - Many versions in flight

  - Different versions often in different locations

  - As we get more developers, a central system gets less practical

- Driven primarily by open source software

  - But turns out to be very applicable to enterprise use as well

- We can tune the distributed systems to match the engineering workflow

3.13 Integrating Version Control into Jenkins

- There are various plugins available for integrating version control software into Jenkins

- E.g. Git, SVN, Perforce, …

- Jenkins can be configured to run a job when code is checked-in

3.14 What is Git

- Brief History
  - The Linux kernel was initially stored in a proprietary VCS called BitKeeper
  - Up til then, BitKeeper had been provided free to the Linux kernel team
  - In 2005, there was a falling-out
  - Linus Torvalds decided to create his own VCS
    - Git

3.15 Git's Design Goals

- Essentially, Git was designed with a view towards running the Linux development process.

- Which is...

  - ⬦ Big

  - ⬦ Distributed responsibilities

  - ⬦ Ability to selectively apply patches

  - ⬦ Big

  - ⬦ Non-linear development (thousands of parallel branches)

  - ⬦ Did we mention big?

3.16 Git's Design Goals (cont'd)

- Linus had a few design goals

  ◇ Speed

  ◇ Simplicity

  ◇ Fully Distributed Development

3.17 Branching and Merging

- Centralized Systems have one central repository

  ◇ Developer checks out a working copy, does work, then commits changes back to central repository

  ◇ What if developers are working on different things? e.g.

    ▪ Different features being added for next release

    ▪ Bug fix to released code

    ▪ Bug fix to a legacy version (n-2)

3.18 Branching and Merging (cont'd)

- Solution is "Branching"

  - System has multiple streams of development, e.g.

    - Main branch

    - Release Branch

    - Development Branch

    - Feature Branches

  - Then, you also need to be able to copy changes from one branch to another

    - Merge

  - These have historically been "hard"

3.19 Centralized Version Control

- Even with branches, you have the problem of working offline

  - Developers now work from home, on airplanes, at coffee shops, etc

  - We'd like to be able to store changes as we work

    - Commit early, commit often!

    - So we can back out things that don't work

    - At the same time, we don't want to "break the build"

3.20 Distributed Version Control

- Proprietary
  - Sun WorkShop TeamWare – 1990s
  - BitKeeper -1998
- Started to appear as open-source in mid-2000's
  - Arch -2001
  - Monotone – 2003
  - Darcs - 2003
  - Git – 2005
  - Mercurial (Hg) – 2005
  - Bazaar (2005)

3.21 Git Basics

- Git doesn't track files, it takes snapshots
  - ◇ Traditional systems track files individually, store deltas
  - ◇ git takes snapshots of the whole directory, stores the whole snapshot
  - ◇ With some optimizations so as not to duplicate data!
- Git maintains a database of your file tree
  - ◇ Generally only adds data to the database
  - ◇ The data added for a commit naturally forms a chunk that can be moved around

3.22 Git Basics (Cont'd)

- Branching is easy and cheap
    - ◊ No longer a rare occurrence, it's part of daily workflow
- Many "workflows" are possible in git
    - ◊ You can keep on doing things as though you had a central repo
    - ◊ Feature branches?
    - ◊ "Gitflow"
    - ◊ "Forking" workflow

3.23 Git Basics (cont'd)

- Push

  - ◇ Takes changes committed to the local repository (branch) and applies them to a remote repository (branch)

- Pull

  - ◇ Takes changes committed to a remote repository (branch) and applies them to a local repository (branch)

- Push and Pull are the mechanisms that allow distributed collaboration

- Important - You will have a distributed workflow!

  - ◇ It isn't possible to "commit" to a central repository

  - ◇ All commits are done against a local repository

  - ◇ Then pushed to a remote repository (or pulled)

3.24 Getting Git

- http://git-scm.com

- But have a look first – you might already have it

  ◇ Shipped with many Linux distributions

  ◇ Included in Apple Xcode

  ◇ Included in Microsoft Visual Studio

  ◇ Embedded in Eclipse, Netbeans, etc

  ◇ But may require a local install as well

3.25 Git on the Server

- Easiest way is to just allow ssh access to the server
  - ◇ "Server" functionality is built into the cmd-line tools
  - ◇ Can also access repos in shared file systems
    - But often, ssh will be faster
- Git protocol
  - ◇ built-in but not authenticated
  - ◇ Won't route through proxies, etc
- Http/https
  - ◇ Easy to setup for read/pull
  - ◇ Push can be done, but rarer – usually use ssh

3.26 Git Repository Managers

■ Non-Exhaustive List

  ◇ GitLab

  ◇ Atlassian Stash

  ◇ GitHub Enterprise

  ◇ GitBlit

  ◇ scm-manager

  ◇ Perforce

  ◇ There are probably others...

3.27 Git on Somebody Else's Server

- The rise of Git has coincided with the emergence of hosted Git repositories
  - GitHub
  - BitBucket
- These repositories have brought workflow and collaboration along with them, e.g.
  - Pull requests
  - Code review
  - Documentation sites
- Also, there are in-house versions of git repos
  - Atlassian Stash
  - GitHub Enterprise

3.28 Using Git

- Git is fundamentally a Linux command line utility

  ◇ 'git'

- Same functionality is often put into IDEs

- Git for Windows ships with

  ◇ Bash shell that runs under Windows

    ■ derived from Cygwin

  ◇ GUI

  ◇ Explorer integration (context menu)

3.29 Definitions

- Developer's Work Area

  - Terminology is a little loose.

  - A folder on disk can contain a repository

  - As a hidden folder called ".git"

  - Typically thought of as three areas:

    - Working Copy

    - Repository

    - Staging Area

3.30 Definitions (cont'd)

- Repository
    - A repository is an area that stores a version-controlled image of a folder on disk
    - Recall that Git doesn't track files exactly, it stores snapshots of a repository
    - Repository contains a complete version history
    - Casually, we might say a folder contains a git repository
    - A repository can also exist on a remote server (or actually just anywhere besides 'here')
    - Repository can be 'bare', meaning it doesn't have a working copy

3.31 Repository (cont'd)

- 'git init' creates a repository in the current directory

  ◇ stored in a hidden folder called '.git'

- 'git init --bare' creates a bare repository in the current directory

  ◇ Bare repository is used as a remote repository

  ◇ Similar to the server-side repo in CVS or SVN.

  ◇ Doesn't have '.git' folder - repository files are in current directory.

3.32 Definitions (cont'd)

- Working Copy

  ◊ The files and folders contained in the developer's work area, apart from the Git repository

  ◊ We edit and manipulate files in the Working Copy

  ◊ We use git commands (or a gui) to move files between the working copy, staging area and repository

3.33 Definitions (cont'd)

- Staging Area
  - Sometimes called the "Index"
  - Basically the list of files that are part of the next commit
  - 'git add <file>' adds a file to the staging area
  - 'git add .' or 'git add --all' adds everything
  - 'git rm --cached <file>' removes from the index
  - 'git status' shows status of staging area and working copy

3.34 Commit

- A snapshot of the working area at a point in time

- Add files to the staging area, then commit --> Files go to repository

- 'git commit' sends files to the repository from the staging area
  - ◇ Requires a 'commit message' that is stored with the commit
    - Displays an editor to edit the message
      - 'git config core.editor <editor-exe>' configures which editor
      - Ships with 'vim'
    - 'Use git commit -m "message here"' to supply message on cmd line

3.35 Commit (continued)

- Each commit, when complete, is identified by an SHA-1 hash value

    ◇ e.g. d11b35f57881f7665a16a3195eeaf722fd156e67

    ◇ We can usually use a shortened version of the hash

- Each commit has a "parent" commit, so the history is viewed as a chain of commits.

    ◇ Actually, some commits have two parents - a "merge" commit

- The commit represents a snapshot of the working copy

- We can return to any snapshot at any time

    ◇ 'git checkout <commit-id>'

- The current parent commit is called the "HEAD".

3.36 How to Think About Commits

- The repository is a collection of snapshots, or "Commits"

- Each commit has a parent

- The current parent is the "HEAD"
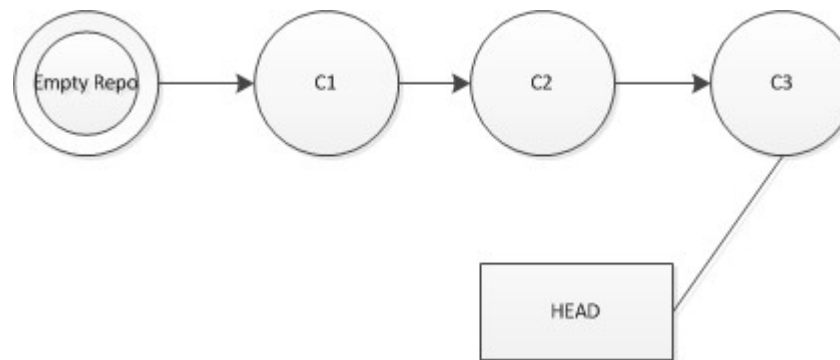
3.37 Viewing History

- 'git log'
  - ◇ Show history

- 'git log -p'
  - ◇ Show history with patches

3.38 Configuring Git

- There are a few things git needs to know...

  ◇ Developer's name and email

  ◇ Preferred editor

  ◇ PGP Signing key

    - What? - we'll get to that...

  ◇ Default behaviors

    - e.g. what to do about end-of-line characters

  ◇ Files of interest, files to ignore

  ◇ Activity "hooks"

    - Code that we can execute as part of git's operations

3.39 Configuration Scope

- We can configure Git at a few different scopes

  - System Global - Settings for all users

    - 'git config --system <attr-name> <attr-value>'

  - Global - Settings for the current user (all repositories)

    - 'git config --global <attr-name> <attr-value>'

  - Local

    - 'git config <attr-name> <attr-value>

    - Per-repository

3.40 User Identification

- Think about this for a minute - in a distributed version control system, we have a little problem
  - ◇ We can't count on operating system ids or logins, because we're going to be moving commits from one repository to another
  - ◇ Repositories could be on different systems (remotes)
  - ◇ These systems may not be directly connected
  - ◇ Systems may not share an authentication store
- Git needs to record the user's identity information with each commit
- At the very least, you need to be rigorous about email address standards
  - ◇ If you have multiple email addresses, be very careful!
- You might need cryptographic verification of contributors' identity

3.41 User Identification (cont'd)

- To configure user id across all the user's repositories

  ```
  git config --global user.name "Jane Doe"

  git config --global user.email jane@doe.com
  ```

- To configure for a local repository only (execute in your working copy)

  ```
  git config user.name "Jane Doe"

  git config user.email jane@doe.com
  ```

3.42 GPG Signing

- An obvious problem with the user identification!

  - Anybody could do 'git config --global user.name billg@microsoft.com'

- Even if it were practical to have git look at user authentication, what about moving commits between different repositories?

  - The only information is what's in the repository.

  - System context can't move with the commit data.

- The solution is GPG signing of commits and tags

3.43 Gnu Privacy Guard

- GPG is Gnu Privacy Guard

- Open-source implementation of RFC4880, PGP (Pretty Good Privacy)

- It's a little out-of-scope for this document

3.44 GPG Basics

- Dual-Key Encryption

- Generate a key pair

- Publish your public key to one or more key servers

- Keys are identified by their "Key Fingerprint" or "Key ID"

  - Key ID is the last 8 digits of the Key Fingerprint

- There is no central authority, unlike X.509 certificates

- "Web of Trust"

  - People "sign" other people's keys, certifying that they know that person and that person owns that key

3.45 GPG and Git

- Git can calculate and record a digital signature for a commit or a tag

- Advantage - committer is positively identified

- Identification is recorded in the repository, can't be repudiated

- To use:

  - Setup GPG

  - Generate a key pair

  - 'git config --global user.signingkey <key-id>' or

  - 'git config user.signingkey' (per-repository)

  - On commit, 'git commit -S'

    - Note Capital-S!

3.46 .gitignore

- Quite often, there are files that we don't want to put in version control

- e.g.
  - ◇ Build artifacts
  - ◇ Binaries
  - ◇ Editor's temporary files
  - ◇ Generated source code

- '.gitignore' lets us exclude files from 'git add' etc

- Contains a list of patterns to ignore

- Version-controlled and distributed when we clone a repo

- For non-shared, edit "$GIT_DIR/info/exclude"
  - ◇ $GIT_DIR in a working copy is ".git"
  - ◇ In a bare repository is just the repository folder

3.47 Other Useful Configurations

- 'man git-config' - tells you the options

- core.editor - Preferred text editor

- commit.template - points to a file for commit messages

3.48 Summary

- Version Control is very important in DevOps.

- There are various tools for utilizing version control, such as, Git and SVN

Chapter 4 - Continuous Integration

Objectives

Key objectives of this chapter

- Understanding Continuous Integration
- Understanding Jenkins
- Understanding Apache Maven

4.1 What is Continuous Integration

■ "Integrate and test changes after no more than a couple of hours."

  ■ Beck & Andres "Extreme Programming Explained"

  ◇ Integrate and build the complete product

  ◇ If a website, build the website

  ◇ If a GUI install, build the installer

  ◇ Etc.

4.2 What is Continuous Integration (cont'd)

- Purposes

    - You find out quickly about integration problems

    - Immediately evident if a code change "breaks the build"

    - Prevents a long drawn-out integration step at the end of code changes

    - Should be complete enough that eventual first deployment of the system is "no big deal".

4.3 What is Continuous Integration (cont'd)

- Usually, CI goes along with Test-First design and automated QA
  - ◇ Run the unit-test suite and smoke testing to ensure the build isn't broken
- Clearly, automatic build is a pre-requisite
  - ◇ So we need a build system – Maven for instance
- Can be synchronous or asynchronous
  - ◇ Asynchronous – Integration happens automatically on code committal
  - ◇ Synchronous – Trigger the build manually after a development session

4.4 What is Continuous Integration (cont'd)

- Side Effects
    - ◇ Generate development reports
    - ◇ Install to QA, User Test, etc
    - ◇ Always have an installable artifact
- It's a great time to generate development metrics
    - ◇ E.g. Code Coverage, Standards Compliance, Static Analysis

4.5 Integration Tools

- Jenkins

- Travis-CI

- TeamCity

- Chef Delivery

- CloudBees

- ...

4.6 Typical Setup for Continuous Integration
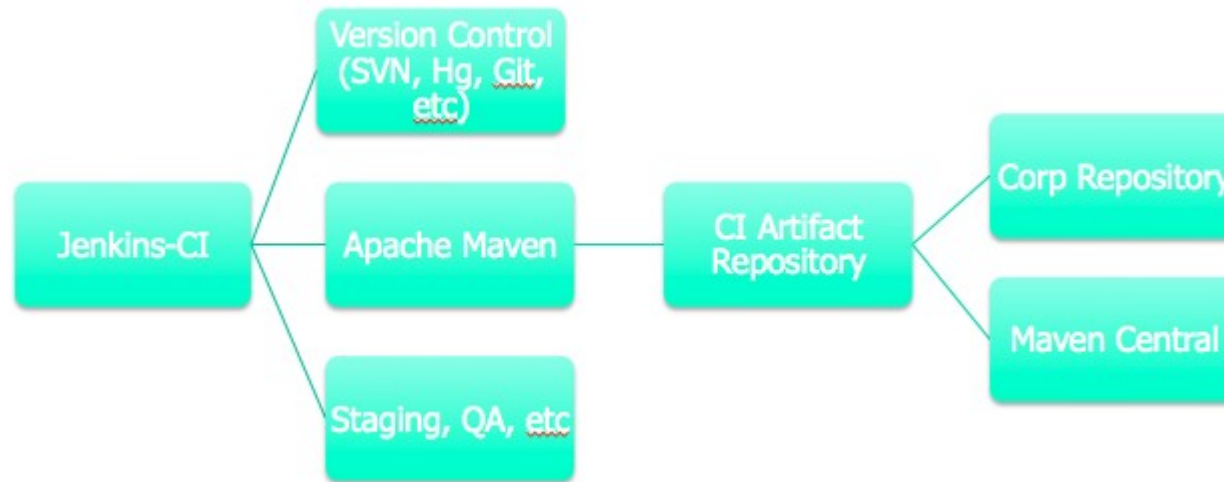
4.7 Typical Setup for Continuous Integration

- Notes:

  - CI system gets the code directly from version control

  - Build is independent of any local artifacts that are on the developer's machine

    - Controlled links to corporate repository and Maven Central

    - Goal is to ensure that the package can be built from the corporate repository

  - Jenkins can have connections to a deployment environment

    - Production Staging

    - User Acceptance Testing

    - QA

    - Load Test

    - Etc…

  - It turns out that if we use Maven, we ABSOLUTELY NEED a local repository manager

    - More info to follow…

4.8 Jenkins Continuous Integration

- Originally developed at Sun by Kohsuke Kawaguchi?
  - Originally "Hudson" on java.net circa 2005
  - Jenkins forked in November 2010
  - Hudson is still live, part of Eclipse Foundation
  - But Jenkins seems to be far more active

4.9 Jenkins Features

- Executes jobs based on a number of triggers

  - Change in a version control system

  - Time

  - Manual Trigger

- A Job consists of some instructions

  - Run a script

  - Execute a Maven project or Ant File

  - Run an operating system command

- User Interface can gather reports

  - Each job has a dashboard showing recent executions

4.10 Running Jenkins

- You can run Jenkins Standalone or inside a web container

- You can setup distributed instances that cooperate on building software

- Can setup jobs in place of what might have been script commands.

4.11 Jenkins Integration with various Version Control Solutions

■ Git

■ SVN

■ ...

■ ...

4.12 Jenkins Job

- A job is a series of steps

- E.g.

  - Check-in / Check-out source code

  - Build

  - Run tests

  - Check code quality

  - Execute scripts

  - Execute a Chef cookbook / recipe

4.13 Apache Maven

- ◇ Originally created by Jason van Zyl

- ◇ Part of the build system for Apache Jakarta Turbine

  - ▪ an early web application framework

- ◇ Turbine had many modules that needed to share 'jar' files

- ◇ Maven attempts to unify and simplify the build across projects

4.14 Goals of Maven

◇ Help Developers comprehend the complete state of a development effort in the shortest period of time

◇ Make the build process easy

◇ Provide a uniform build system

◇ Provide quality project information

◇ Provide guidelines for best practices in development

◇ Allow transparent migration to new features

4.15 What is Apache Maven?

◇ Maven is many things:

- A build system for Java projects

- Dependency Management System

- Packaging and Release Management

- An EcoSystem

  - Maven Central

  - Plugins – e.g. mojo.codehaus.org

  - Apache Ivy

  - Gradle

4.16 What is Apache Maven (cont'd)

◇ A Philosophy

■ Maven has opinions about how you should organize and manage software projects

◇ An alternative to Ant and other build systems

4.17 Why Use Apache Maven?

◇ Short Answer – simplify the process of building software in Java

▪ There is a cost – you need to adopt and embrace the "Maven Way"

◇ Dependency Management is probably the most visible feature

▪ We don't specify how to build the project, so much as "what the project is"

▪ Describe the software dependencies

▪ What jar files do we need to reference?

▪ Maven can then go download the jar files we've specified

▪ So developers don't have to find or manage local copies

4.18 The Maven EcoSystem

◇ Side effect of delegating dependency management is that the Maven EcoSystem includes a central repository for software

◇ Maven Central

■ One of the primary distribution mechanisms for open-source software

◇ Business organizations can use the central repository idea

■ Setup internal software repository

■ Repository contains any software modules used by a business

■ plus a local cache of modules downloaded from Maven Central

4.19 Consistent Easy-to-Understand Project Layout

◇ Maven encourages all projects to use the same layout for sources, targets, etc

◇ Multi-module projects are always a parent module and one or more child modules

◇ Dependencies automatically downloaded from Maven Central or a local repository

◇ Large ecosystem of plugins that give consistent handling of different types of software projects

4.20 Convention Over Configuration

◇ With most build systems, you describe the sequence of steps required to build the package

- e.g. with Ant

  - 'package' target builds jar file, depends on 'compile' target

  - 'compile' target compiles source files, depends on 'prepare' target

  - 'prepare' target creates the target directories

  - and so on...

- essentially, you tell Ant how to build the package

4.21 Maven is Different

- ◇ With Maven, we describe the project itself
- ◇ XML-formatted text file 'pom.xml'
  - ■ 'pom' stands for Project Object Model
- ◇ Describe the name and version of the package that's being produced
- ◇ Describe which external 'jar' files the project depends on
- ◇ Describe this module's place in a multi-module build.

4.22 Maven Projects have a Standardized Build

⬦ Sequence of steps in a build is called the "lifecycle"

- Defined by the packaging of the project - e.g. 'jar' or 'war'

⬦ No need to read the build script to know what the build looks like

- All Maven builds for 'ear' packaging are the same

- The only question is what tools or "plugins" are applied during the build

  - Specified in the 'pom.xml'

4.23 Effect of Convention Over Configuration

- ◇ Once you understand the build for a given packaging, you understand that pattern for any Maven project that uses the same packaging

- ◇ Layout of source files is consistent across projects

- ◇ e.g. an EJB project

  - ▪ consists of a module for client API, separate module for the implementation, and a module for the 'ear' file.

  - ▪ There will be a 'src' directory with 'main' and 'test' children, with 'java' directories under each.

4.24 Importance of Plugins

◇ Plugins encapsulate the best practices on how to use a given set of functionality

◇ e.g. Code Coverage should be applied at unit test

- Procedure is to instrument the test classes, run the test while gathering data, then generate a report

- Can be applied in Maven simply by referencing the plugin

- As a bonus, the plugin is automatically downloaded just like a software dependency – no installation!

◇ e.g. 'war' packaging plugin knows all the files that need to go into a standard 'war' file.

4.25 A Key Point on Maven!

◇ In prior build systems like Ant, we tell the build system what to do with a set of files

◇ In Maven, we are describing a project, not what to do with it

   ▪ pom.xml

◇ If you try to treat Maven as just a build system, it isn't going to work out well!

◇ Maven is declarative, not procedural

4.26 Summary

- Continuous Integration is very important in DevOps.

- It can involve interaction with version control software, build code using a build manager, execute scripts for automation, …

- There are various tools for implementing continuous integration

Chapter 5 - 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

5.1

5.2 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

5.3 Continuous Delivery (cont'd)
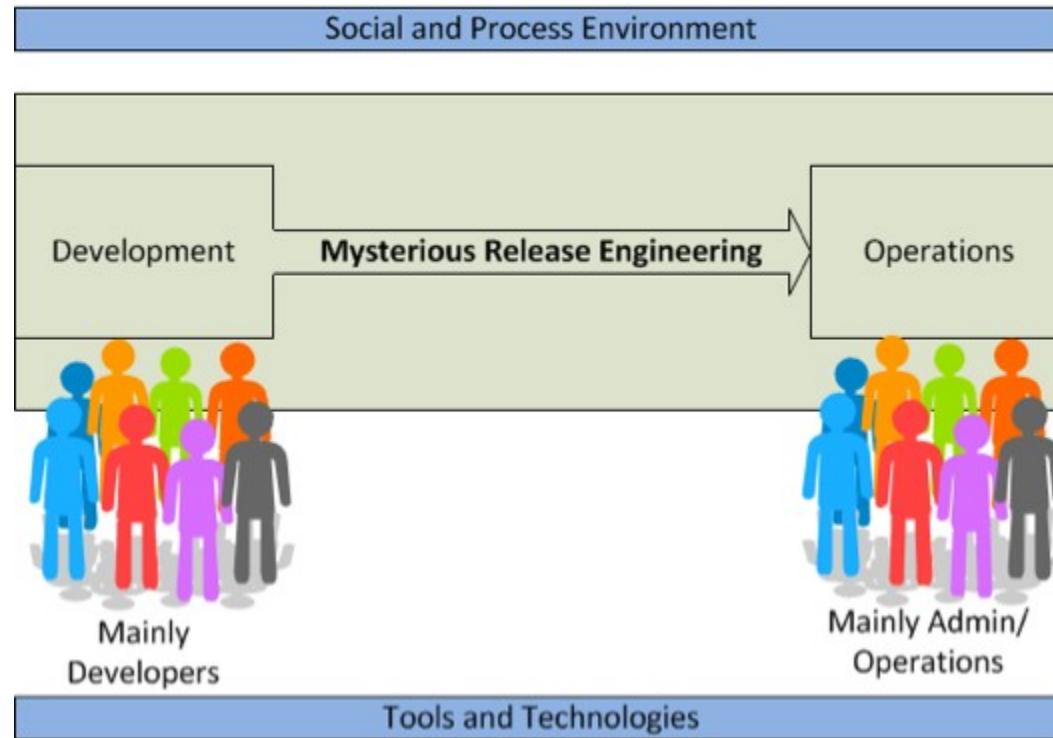
5.4 DevOps and Continuous Delivery

◇ 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

5.5 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"

5.6 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

5.7 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

5.8 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)

5.9 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

5.10 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/**'

}
```

5.11 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

5.12 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
```

5.13 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.

5.14 Invoking the Pipeline

◇ The pipeline shows up just like a regular Jenkins job

All	**+**			
S	**W**	**Name** ↓	**Last Success**	**Last Failure**
🔵	☁	ReleasePipeline	8 hr 18 min - #3	8 hr 22 min - #2
🔵	☁	SimpleGreeting	2 days 9 hr - #4	N/A

Icon: S M L

Legend RSS for all RSS for fai

■ It can be invoked manually, at a predefined interval, or based on SCM polling

◇ Just like a regular job

5.15 Interacting with the Pipeline

◊ Stages in the pipeline appear on the user interface

Pipeline ReleasePipeline

📝add description

📝 Recent Changes

Stage View

	CI Build on head	Release to Staging Repo	CI Build on release
Average stage times: (Average full run time: ~16s)	16s	373ms	3ms
#3 Aug 17 02:17 No Changes	16s	373ms	3ms (paused for 28s)
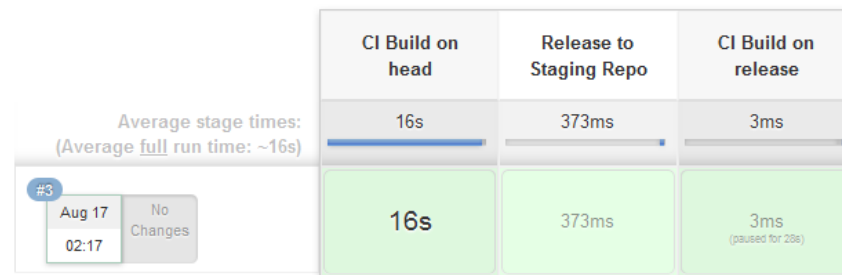
◊ In case of input required or output, these appear on the UI

5.16 Pipeline vs Traditional Jobs

⬧ In the long term, we expect that use of traditional (config.xml) jobs will be replaced by Pipeline jobs

⬧ 'config.xml' is not stored in version control

  ▪ It is possible to manually store it, but Jenkins doesn't know anything about versioning the config.xml

⬧ Libraries allow the same process to be applied to multiple jobs

⬧ Plugins like the "BitBucket Branch Source Plugin" and the "GitHub Branch Source Plugin" let administrators basically apply the same job setup to all the repositories in a given organization

  ▪ Under this practice, you get a CI job/jobs automatically when the repository is created

5.17 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

Chapter 6 - Continuous Code Quality

Objectives

Key objectives of this chapter

- Understanding Continuous Code Quality
- Understanding SonarQube
- SonarQube Installation
- Using sonar-scanner

6.1 Continuous Code Quality

- DevOps involves development, operations, and quality assurance

- Plays a very important role in DevOps

- Application / software should be tested

- Test types: unit tests, integration tests, functional tests, smoke tests, …

- Continuous quality assurance also involves checking the code quality

- Code quality can be checked by using software, such as, SonarQube, Clover, and FxCop

6.2 What is SonarQube

- SonarQube is a free and open source "code quality platform"

- It gives you a moment-in-time snapshot of your code quality, as well as trending lagging and leading quality indicators

- It also offers quality-management tools e.g. IDE plugins, integration with Jenkins, and code-review tools

- It covers not just bugs but also coding rules, test coverage and duplication

6.3 SonarQube - Benefits

- Tester: It can help in pinpointing the spots where automated testing is thin or non-existent

- Developer: It can identify potential pitfalls e.g. language-specific subtleties, thread safety and resource management

- Software Architect: It can help in keeping an eye on whether your cleanly delineated initial design is being degraded over time with creeping dependency cycles. It can show whether the internal coding rules are being followed and whether code should be refactored

- Project Manager: Testing can show level of external quality where as SonarQube can show level of internal quality. Code with higher internal quality is easier to maintain

- Business: It can help in offering a strong ROI because its acquisition and setup costs are low and intuitive code can help in faster training

6.4 SonarQube (Multilingual)

- SonarQube is written in Java, and it started as a way to measure the quality of Java projects

- It supports: ABAP, C, C++, C#, COBOL, Delphi, Drools, Flex/ActionScript, Groovy, JavaScript, Natural, PHP, PL/I, PL/SQL, Python, Visual Basic 6, Web (JSP, JSF, XHTML), XML

- More languages can analyzed by adding plugins, many of which are provided by the SonarQube community and offered for free

6.5 Seven Axes of Quality

- SonarQube not only checks for bugs but for other metrics as well. It's called the Seven Axes of Quality by the developers.

- The Seven Axes of Quality are:

  ◇ Potential bugs

  ◇ Coding rules

  ◇ Tests

  ◇ Duplication

  ◇ Comments

  ◇ Architecture and design

  ◇ Complexity

6.6 Potential Bugs

- The creators of SonarQube, list potential bugs and coding rules as separate axes, but for reporting they group them together under issues

- Issue counts are considered as lagging quality indicators. They show what's already gone wrong

- Issues are ranked at different severities blocker, critical, major, minor, and info

6.7 Tests

- In test-driven development, the test side of the code is entered first

- The test coverage widget shows what percent of the code is tested and whether your tests are passing, failing, or erroring-out

6.8 Comments and Duplication

- There are two types of code comments:

  ◇ The ones inline in any method

  ◇ The ones outside a public method

- SonarQube measures the second kind of comment (the API documentation)

- Duplication doesn't encourage code reuse. SonarQube can detect code duplication

6.9 Architecture and Design

- The tidiness of code architecture helps in better code maintenance

- It checks whether something is wrong in the implementation (lagging), or an indication of how hard the code base will be to understand and maintain in future (leading)

6.10 Complexity

- It's about how many pairs of curly braces your method has

- The me pairs of curly braces there are, the more complex the logic is

- Complex code should be refactored

6.11 SonarQube Installation

- Requires database engine, such as, MySQL

- Nemo is SonarQube's public instance: http://nemo.sonarsource.org

- SonarQube can be downloaded and installed on a local machine

- Ideally, SonarQube and its database should be on two separate hosts to split the CPU load

- Network I/O is one of the biggest determinants in how long an analysis takes

- SonarQube can be accessed via web browser. It's default port is 9000

- For analyzing the code, the SonarQube Runner is also required.

- SonarQube Runner runs the analysis. Alternatively, you can run analysis using Maven or Ant

6.12 SonarQube Components

- SonarQube Server

- Command line tool: sonar-scanner

- Web user interface
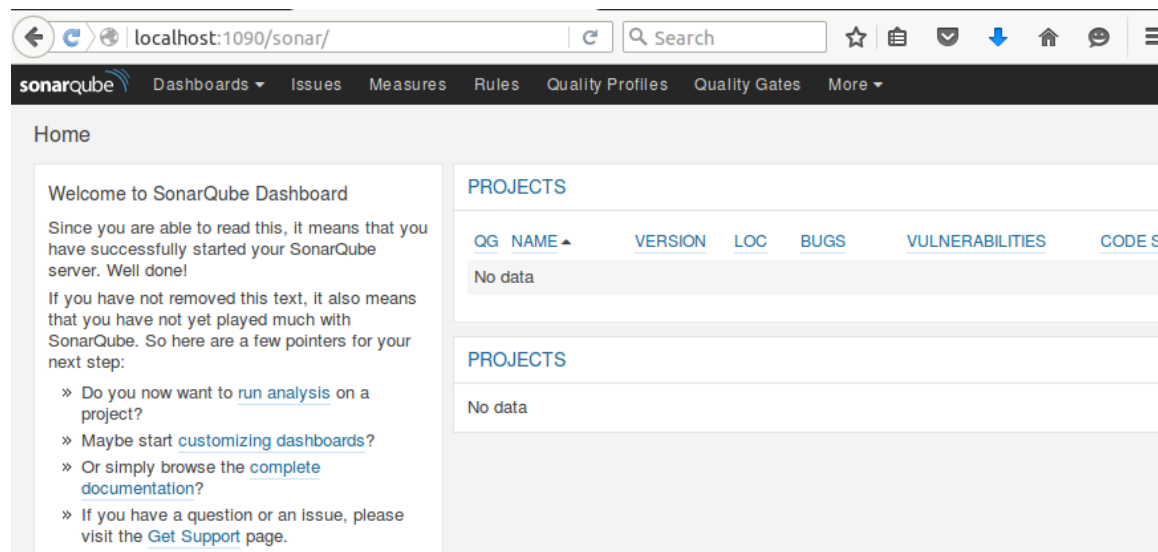
6.13 Code Quality (LOC, Code Smells)

PROJECTS

QG	NAME ▲	VERSION	LOC	BUGS	VULNERABILITIES	CODE SMELLS	LAST ANALYSIS
✓	My SonarQube project	1.0	7	0	0	4	02:54

1 results

6.14 Code Quality (Project Files)

My SonarQube project   src/java/hello.java

Move this file to a named package.   ...
Code Smell   ⊘ Minor   ○ Open   Not assigned   10min effort

Add a private constructor to hide the implicit public one.   ...
Code Smell   ⊗ Major   ○ Open   Not assigned   30min effort

Replace this usage of System.out or System.err by a logger.   ...
Code Smell   ⊗ Major   ○ Open   Not assigned   10min effort

My SonarQube project   src/js/hello.js

Add the "let", "const" or "var" keyword to this declaration of "b" to make it explicit.   ...
Code Smell   ⊗ Major   ○ Open   Not assigned   2min effort

6.15 Code Quality (Code)

My SonarQube project
src/js/hello.js

```
1        var a = 0;
2    ⌃   b = 10;

         Add the "let", "const" or "var" keyword to this declaration of "b" to make it explicit. ⋯
         Code Smell   ⬥ Major   ◯ Open   Not assigned   2min effort

3
```
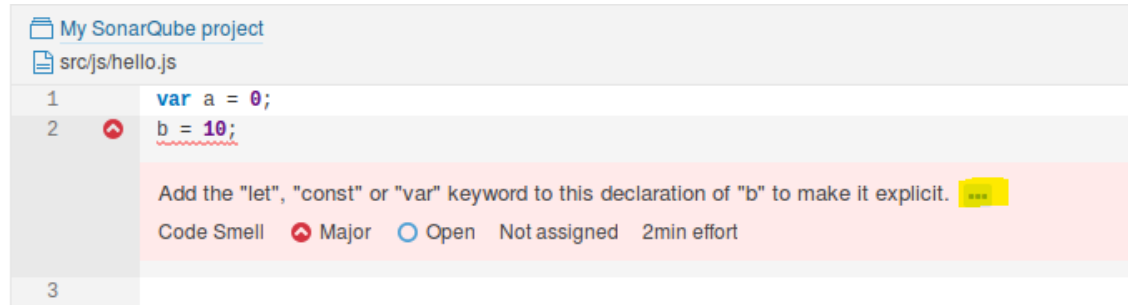
6.16 Summary

- Continuous Quality Assurance / Continuous Code Quality are very important in DevOps.

- Checking for code quality can reduce the effort required to maintain code and fix bugs

- There are various options for checking code quality.

- SonarQube is one of the most popular options for ensuring code quality

Objectives

Key objectives of this chapter

- Understanding Continuous Monitoring
- Understanding Dynatrace
- Understanding Splunk
- Understanding Nagios
- Installing Nagios
- Monitoring Hosts and Services using Nagios

7.1 What is Continuous Monitoring

- Application run-time behavior monitoring should begin in production-like environments where the application would have setup, configuration, and other parameters close to those used in production

- Things to look for:

  ◇ Run-time application behavior (CPU, RAM, I/O, average duration of garbage collection pauses, if applicable, and other metrics)

  ◇ Response time per application interface

  ◇ Excessive or insufficient logging

  ◇ Logging of sensitive information

  ◇ etc.

7.2 Monitoring Tools

- Common monitoring tools

  - Nagios

  - Dynatrace

  - Splunk

  - New Relic

  - Graphite

  - Icinga

  - Cacti

  - PagerDuty

  - Sensu

  - ...

7.3 Dynatrace Application Monitoring

- Dynatrace is an American Application Performance Management (APM) Software company

- APM is the monitoring and management of performance of software applications

- APM detects complex application performance problems to maintain an expected level of service

- APM helps in translating IT metrics into business value

- Two sets of performance metrics are monitored:

  - The first set of performance metrics defines the performance experienced by end users of the application

  - The second set of performance metrics measures the computational resources used by the application for the load

7.4 Dynatrace Application Monitoring (contd.)

- Dynatrace appmon detects and diagnoses problems in real time, drilling down to the offending code

- It automatically sees and analyzes every user transaction in real-time

- It integrates with CI/CD solutions to automatically check performance in your build pipeline, before you get to production

- It can capture timing and code-level context for every transaction and across every tier

- Services, tiers, inter-dependencies and inter-tier timings are automatically generated in real-time for any environment – distributed, cloud, mainframe, and hybrid

- It correlates guest and host infrastructure health with individual application transactions in real-time, including guest OS, and virtual and host server details

- With base-lining, it learns what normal looks like for your application and eliminates "false positives" alerts

- It can be combined with Dynatrace Load to ensure application readiness for deployment or a major event. Load is a SaaS that simulates up to 1 million virtual users to stress test applications from the cloud

7.5 Dynatrace Application Monitoring

- It helps in releasing software more frequently and faster by focusing on quality early on throughout your pipeline

- It closes the feedback loop by delivering end-to-end user experience metrics from production back to engineering

- It integrates into your IDEs, CI, Test Frameworks and Production and provides automated metrics-driven feedback and regression detection for every line of code

- It captures the root cause and alerts on quality issues as they get introduced in your code base

- It supports technology stack including: Java, .NET, .PHP, Node.js, NGINX, Docker, Jenkins, iOS, Android, Apache, Chef, Puppet, ...

7.6 Splunk

- Splunk is an American corporation

- Its name was inspired by the process of exploring caves or splunking

- Splunk's software uses a standard API to connect directly to applications and devices

- It comes in enterprise and free version

- It helps analysts, operators, programmers, and others explore data from their organizations by obtaining, analyzing, and reporting on it

- It manages, searches, inserts, deletes, filters, and analyzes big data that is generated by machines, as well as other types of data

7.7 Splunk Functionalities

- Data Collection: The process of collection data from various sources. Such data is referred to as machine data. Most of this data is streaming data

- Data Indexing: Before data can be searched, it needs to be indexed. Index creation requires two steps: parsing and indexing. Parsing is separation of data into events

- Data Searching: Splunk helps in quickly searching for the data.Splunk makes it easy to search on different dimensions of the data.

- Data Analysis: Splunk can be used for quick data analysis. Indexing creates a centralized data repository that can house data of many types from a variety of sources. Splunk has a variety of default data visualizations for reports and dashboards that helps users in improving decision-making

7.8 Splunk Searching

- Searches are not case sensitive

- To get exact case enclose it in CASE(<search_term>)

- There is an implied **AND** in between words. E.g. "log error" is equivalent to "log AND error"

- **OR** can be used explicitly e.g. "log OR error"

- **NOT** operator is also available e.g. "log NOT error"

- To get search results for an exact phrase, enclose it in quotes. E.g. "log error" with quotes

- Field can be specified as part of search query e.g. text=*log* OR text=*error*

7.9 Splunk Functions

- avg(X) – average

- dc(X) – distinct count

- earliest(X) – earliest value of field X, chronologically

- last(X) – last seen value

- latest(X) – latest value of field

- list(X) – list of all values of field X as a multi-value entry

- max(X) – maximum value of X

- median(X) – middle value

- ...

7.10 Nagios

- Network, Server and Log Monitoring

- There are various flavors of Nagios:

  - Nagios Core: Event scheduler, event processor, and alert manager

  - Nagios XI: Monitor entire IT infrastructure

  - Nagios Log Server: View, sort and configure logs from any source on any given network

  - Nagios Fusion: Gain a centralized visual operational status and enable faster problem resolution over entire network

  - Nagios Network Analyzer: Identify where your bandwidth is dipping or spiking

7.11 Nagios (contd.)

- It is implemented as a daemon written in C for performance

- It ships with a default CGI interface

- The web interface is used to view and manage elements that are monitored by Nagios

7.12 Nagios – Installation

- Nagios Core source code is downloadable for free.

- Source code is compiled using Java.

- Web server is required and CGI must be configured

- Plugins are optional but highly recommended

  - Plugin examples: check_http, check_ftp, check_uptime, check_disk

7.13 Nagios – Hosts

- Host is a server / machine to monitor

- By default, the local machine is monitored where Nagios is installed

- Additional hosts can be defined in a configuration file

```
define host {
        use                     linux-server
        host_name               yourhost
        alias                   My Apache server
        address                 10.132.234.52
        max_check_attempts      5
        check_period            24x7
        notification_interval   30
        notification_period     24x7
}
```

7.14 Nagios – Web User Interface (Hosts)

7.15 Nagios – Monitoring Services

- Services on a host can be monitored by using plugins

- Plugins make a call to command line utilities

- E.g. check_http, check_ftp, check_uptime, check_disk, …

7.16 Nagios – Monitoring Services (contd.)

Host ⬆⬇	Service ⬆⬇	Status ⬆⬇	Last Check ⬆⬇	Duration ⬆⬇	Attempt ⬆⬇	Status Information
Server2	Current Load	OK	07-21-2016 07:55:03	0d 0h 7m 50s	1/4	OK - load average: 0.13, 0.21, 0.28
	Current Users	OK	07-21-2016 07:56:36	0d 0h 6m 17s	1/4	USERS OK - 4 users currently logged in
	HTTP	OK	07-21-2016 07:53:10	0d 0h 4m 43s	1/4	HTTP OK: HTTP/1.1 301 Moved Permanently - 374 bytes in 0.000 second response time
	PING	OK	07-21-2016 07:53:08	0d 0h 4m 45s	1/4	PING OK - Packet loss = 0%, RTA = 0.05 ms
	Root Partition	OK	07-21-2016 07:55:22	0d 0h 7m 31s	1/4	DISK OK - free space: / 19378 MB (62% inode=80%):
	SSH	CRITICAL	07-21-2016 07:54:55	0d 0h 5m 58s	4/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	07-21-2016 07:53:28	0d 0h 4m 25s	1/4	SWAP OK - 100% free (8676 MB out of 8676 MB)
	Total Processes	OK	07-21-2016 07:54:13	0d 0h 8m 40s	1/4	PROCS OK: 114 processes with STATE = RSZDT
Server3	Current Load	OK	07-21-2016 07:56:13	0d 0h 1m 40s	1/4	OK - load average: 0.15, 0.22, 0.28
	Current Users	OK	07-21-2016 07:57:15	0d 0h 0m 38s	1/4	USERS OK - 4 users currently logged in
	HTTP	PENDING	N/A	0d 0h 2m 42s+	1/4	Service check scheduled for Thu Jul 21 07:58:18 MDT 2016
	PING	PENDING	N/A	0d 0h 2m 42s+	1/4	Service check scheduled for Thu Jul 21 07:59:20 MDT 2016
	Root Partition	OK	07-21-2016 07:56:37	0d 0h 1m 16s	1/4	DISK OK - free space: / 19378 MB (62% inode=80%):
	SSH	CRITICAL	07-21-2016 07:57:25	0d 0h 1m 28s	2/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap Usage	OK	07-21-2016 07:57:28	0d 0h 0m 25s	1/4	SWAP OK - 100% free (8676 MB out of 8676 MB)
	Total Processes	PENDING	N/A	0d 0h 2m 42s+	1/4	Service check scheduled for Thu Jul 21 07:58:30 MDT 2016
localhost	Current Load	OK	07-21-2016 07:53:36	1d 5h 54m 37s	1/4	OK - load average: 0.10, 0.24, 0.30
	Current Users	OK	07-21-2016 07:56:42	1d 5h 53m 59s	1/4	USERS OK - 4 users currently logged in
	HTTP	OK	07-21-2016 07:56:06	1d 5h 53m 22s	1/4	HTTP OK: HTTP/1.1 301 Moved Permanently - 374 bytes in 0.000 second response time
	PING	OK	07-21-2016 07:57:21	1d 5h 52m 44s	1/4	PING OK - Packet loss = 0%, RTA = 0.04 ms
	Root Partition	OK	07-21-2016 07:54:14	1d 5h 52m 7s	1/4	DISK OK - free space: / 19378 MB (62% inode=80%):
	SSH	CRITICAL	07-21-2016 07:55:28	1d 5h 51m 29s	4/4	connect to address 127.0.0.1 and port 22: Connection refused
	Swap					SWAP OK - 100% free (8676 MB

7.17 Monitoring HTTP

```
define service{
        use                 generic-service        ; Inherit default values from a template
        host_name                   remotehost
        service_description     HTTP
        #check_command    check_http
         check_command    check_http!-u /download/index.php -t 5 -s "latest-version.tar.gz"


        }
```

7.18 Monitoring FTP

```
define service{
        use             generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     FTP
        check_command   check_ftp
        }
```

7.19 Monitoring SSH

```
define service{
        use                     generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     SSH
        check_command   check_ssh
        }
```

7.20 Monitoring SMTP

```
define service{
        use                 generic-service         ; Inherit default values from a template
        host_name               remotehost
        service_description     SMTP Response Check
        check_command   check_smtp!-t 5 -e "mygreatmailserver.com"
        }
```

7.21 Monitoring POP3

```
define service{
        use                 generic-service         ; Inherit default values from a template
        host_name                   remotehost
        service_description     POP3 Response Check
        check_command   check_pop!-t 5 -e "mygreatmailserver.com"
        }
```

7.22 Monitoring IMAP

```
define service{
        use             generic-service        ; Inherit default values from a template
        host_name               remotehost
        service_description     IMAP4 Response Check
        check_command   check_imap!-t 5 -e "mygreatmailserver.com"
        }
```

7.23 Summary

- Continuous monitoring plays a very important role in DevOps.

- It's useful for delivering performance as a feature

- Nagios is one of the most popular tools for implementing continuous monitoring

Chapter 8 - Introduction to Kubernetes

Objectives
Key objectives of this chapter
■ What is Kubernetes?
■ What Is a Container?
■ Microservices and Orchestration
■ Microservices and Infrastructure-as-Code
■ Kubernetes Container Networking

8.1 What is Kubernetes

- Kubernetes is Greek for "helmsman" or "pilot"

- Originally founded by Joe Beda, Bendan Burns and Craig McLuckie

- Afterward, other Google engineers also joined the project

- The original codename of Kubernetes within Google was Project Seven, a reference to a Star Trek character. The seven spokes on the wheel of the Kubernetes logo is a nod to that codename

- Kubernetes is commonly referred to as **K8s**

- An open-source system for automating deployment, scaling, and management of containerized applications

- Originally designed by Google and donated to the Cloud Native Computing Foundation

- Provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts

- Supports a range of container tools, including Docker

8.2 What is a Container

- Over the past few years, containers have grown in popularity

- Containers provide operating-system-level virtualization

- It is a computer virtualization method in which the kernel of an operating system allows the existence of multiple isolated user-space instances, instead of just one

- Such instances are called containers

- A container is a software bucket comprising everything necessary to run the software independently.

- There can be multiple containers in a single machine and containers are completely isolated from one another as well as from the host machine.

- Containers are also called virtualization engines (VEs) or Jails (e.g. FreeBSD jail)

- Containers look like real computers from the point of view of programs running in them

- Items usually bundled into a container include:

  ◇ Application, dependencies, libraries, binaries, and configuration files

8.3 Container – Uses

- OS-level virtualization is commonly used in virtual hosting environments

- A container is useful for packaging, shipping, and deployment of any software applications that are presented as lightweight, portable, and self-sufficient containers, that will run virtually anywhere.

- It is useful for securely allocating finite hardware amongst a large number of mutually-distributing users

- System administrators may also use it for consolidating server hardware by moving services on separate hosts into containers on a single server

- Container is useful for packaging everything the app needs into a container and migrating that from one VM to another, to server or cloud without having to refactor the app.

- Container usually imposes little to no overhead, because programs in virtual partitions use the OS' normal system call interface and do not need to be subjected to emulation or be run in an intermediate virtual machines

- Container doesn't require support in hardware to perform efficiently

8.4 Container – Pros

- Containers are fast compared to hardware-level virtualization, since there is no need to boot up a full virtual machine. A Container allows you to start apps in a virtual, software-defined environment much more quickly

- The average container size is within the range of tens of MB while VMs can take up several gigabytes. Therefore a server can host significantly more containers than virtual machines

- Running containers is less resource intensive then running VMs so you can add more computing workload onto the same servers

- Provisioning containers only take a few seconds or less, therefore, the data center can react quickly to a spike in user activity.

- Containers can enable you to easily allocate resources to processes and to run your application in various environments.

- Using containers can decrease the time needed for development, testing, and deployment of applications and services.

- Testing and bug tracking also become less complicated since you there is no difference between running your application locally, on a test server, or in production.

- Containers are a very cost effective solution. They can potentially help you to decrease your operating cost (less servers, less staff) and your development cost (develop for one

consistent runtime environment).

- Using containers, developers are able to have truly portable deployments. This helps in making Continuous Integration / Continuous Deployment easier.

- Container-based virtualization are a great option for microservices, DevOps, and continuous deployment.

8.5 Container – Cons

- Compared to traditional virtual machines, containers are less secure.

  ◇ Containers share the kernel, other components of the host operating system, and they have root access.

  ◇ This means that containers are less isolated from each other than virtual machines, and if there is a vulnerability in the kernel it can jeopardize the security of the other containers as well.

- A container offers less flexibility in operating systems. You need to start a new server to be able to run containers with different operating systems.

- Networking can be challenging with containers. Deploying containers in a sufficiently isolated way while maintaining an adequate network connection can be tricky.

- Developing and testing for containers requires training. Whereas writing applications for VMs, which are in effect the same as physical machines, is a straightforward transition for development teams.

- Single VMs often run multiple applications. Whereas containers promotes a one-container one-application infrastructure. This means containerization tends to lead to a higher volume of discreet units to be monitored and managed.

8.6 Composition of a Container

- At the core of container technology are

  ◊ Control Groups (cgroups)

  ◊ Namespaces

  ◊ Union filesystems

8.7 Control Groups

- Control groups (cgroups) work by allowing the host to share and also limit the resources each process or container can consume

- This is important for both, resource utilization and security

- It prevents denial-of-service attacks on host's hardware resources

8.8 Namespaces

- Namespaces offer another form of isolation for process interaction within operating systems.

- It limits the visibility a process has on other processes, networking, filesystems, and user ID components

- Container processes are limited to see only what is in the same namespace

- Processes from containers or the host processes are not directly accessible from within the container process.
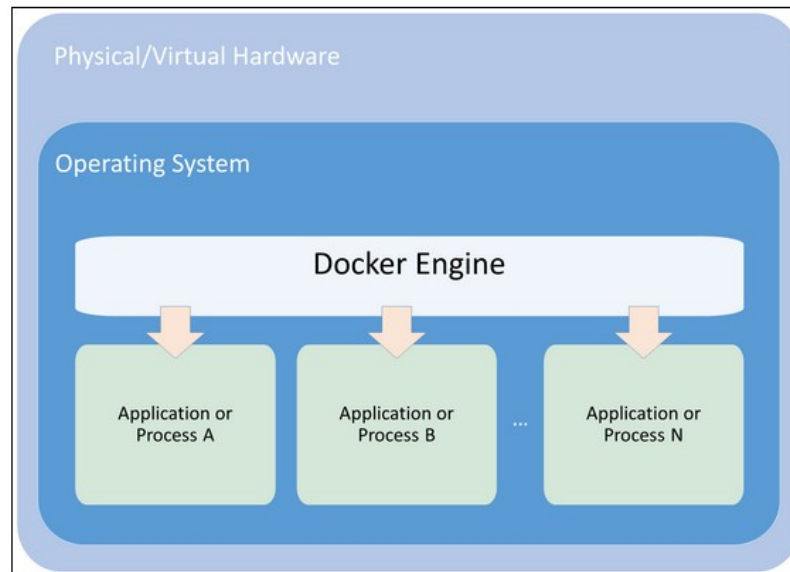
8.9 Union Filesystems

- Containers run from an image, much like an image in the VM or Cloud world, it represents state at a particular point in time.

- Container images snapshot the filesystems

- The snapshot tend to be much smaller than a VM

- The container shares the host kernel and generally runs a much smaller set of processes

- The filesystem is often layered or multi-leveled.

  ◇ e.g. Base layer can be Ubuntu with an application such as Apache or MySQL stacked on top of it

Base Ubuntu Image

Apache Ubuntu Image
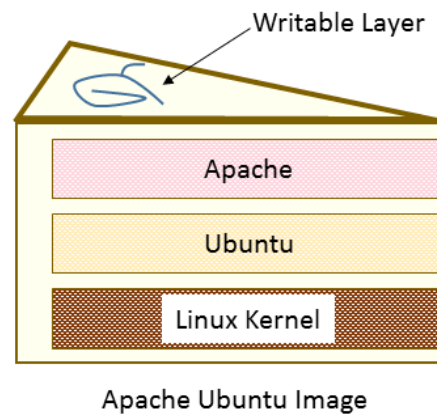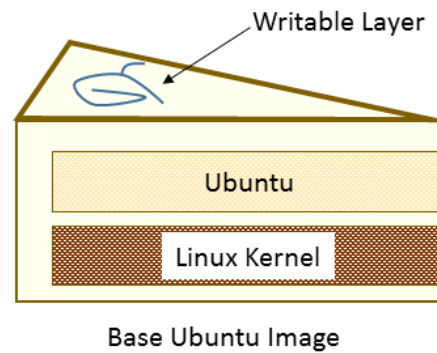
8.10 Popular Containerization Software

- Docker

    ◇ Docker Swarm

- Packer

- Kubernetes

- Rocket (rkt)

- Apache Mesos

- Linux Containers (LXC)

- CloudSang

- Marathon

- Nomad

- Fleet

- Rancher

- Containership

- OpenVZ

- Oracle Solaris Containers

Tectonic

8.11 Microservices

- The microservice architectural style is an approach to developing a single application as a suite of small services

- Each service runs in its own process and communicates with lightweight mechanisms, often an HTTP resource API

- These services are built around business capabilities and independently deployable by fully automated deployment machinery

8.12 Microservices and Containers / Clusters

- Containers are excellent for microservices, as it isolates the services.

- Containerization of single services makes it easier to manage and update these services

- Docker has led to the emergence of frameworks for managing complex scenarios, such as:

  - how to manage single services in a cluster

  - how to manage multiple instances in a service across hosts

  - how to coordinate between multiple services on a deployment and management level

- Kubernetes allows easy deployment and management of multiple Docker containers of the same type through an intelligent tagging system

- With Kubernetes, you describe the characteristics of the image, e.g. number of instances, CPU, RAM, you would like to deploy

8.13 Microservices and Orchestration

- Microservices can benefit from deployment to containers

- Issue with containers is, they are isolated. Microservices might require communication with each other

- Container orchestration can be used to handle this issue

- Container orchestration refers to the automated arrangement, coordination, and management of software containers

- Container orchestration also helps in tackling challenges, such as

  - service discovery

  - load balancing

  - secrets/configuration/storage management

  - health checks, auto-[scaling/restart/healing] of containers and nodes

  - zero-downtime deploys

8.14 Microservices and Infrastructure-as-Code

- In the old days, you would write a service and allow the Operations (Ops) team to deploy it to various servers for testing, and eventually production.

- Infrastructure-as-Code solutions helps in shortening the development cycles by automating the set up of infrastructure

- Popular infrastructure-as-code solutions include Puppet, Chef, Ansible, Terraform, and Serverless.

- In the old days, servers were treated as part of the family.

  ◇ Servers were named, constantly monitored, and carefully updated

- Due to containers and infrastructure-as-code solutions, these days the servers (containers) are often not updated. Instead, they are destroyed, then recreated.

- Containers and infrastructure-as-code solutions treat infrastructure as disposable.

8.15 Kubernetes Container Networking

- Microservices require a reliable way to find and communicate with each other.

- Microservices in containers and clusters can make things more complex as we now have multiple networking namespaces to bear in mind.

- Communication and discovery requires traversing of container IP space and host networking.

- Kubernetes benefits from getting its ancestary from the clustering tools used by Google for the past decade. Many of the lessons learned from running and networking two billion containers per week have been distilled into Kubernetes

8.16 Kubernetes Networking Options

- Docker creates three types of networks by default:

  - bridged – this is the default choice. In this mode, the container has its own networking namespace and is then bridged via virtual interfaces to the host network. In this mode, two containers can use the same IP range because they are completely isolated

  - host – in this mode, performance is greatly benefited since it removes a level of network virtualization; however, you lose the security of having an isolated network namespace

  - none – creates a container with no external interface. Only a loopback device is shown if you inspect the network interfaces

  - host, and none

  - In all these scenarios, we are still on a single machine, and outside of a host mode, the container IP space is not available, outside the machine. Connecting containers across two machines then requires NAT and port mapping for communication

- Docker user-defined networks

  - Docker also supports user-defined networks via network plugins

    - bridge driver – allows creation of networks somewhat similar to default bridge driver

    - overlay driver – uses a distribution key-value store to synchronize the network creation across multiple hosts

- Macvian driver – uses the interface and sub-interfaces on the host. It offers a more efficient network virtualization and isolation as it bypasses the Linux bridge

- Weave

  ◇ Provides an overlay network for Docker containers

- Flannel

  ◇ Gives a full subnet to each host/node enabling a similar pattern to the Kubernetes practice of a routable IP per pod or group of containers

- Project Calico

  ◇ Uses built-in routing functions of the Linux kernel.

  ◇ It can be used for anything from small-scale deploys to large Internet-scale installations.

  ◇ There is no need for additional NAT, tunneling, or overlays.

- Canal

  ◇ It merges both Calico for network policy and Flannel for overlay into one solution

8.17 Kubernetes Networking – Balanced Design

- Using unique IP address at the host level is problematic as the number of containers grow.

  ◇ Assigning an IP address to each container can also be overkill.

- In cases of sizable scale, overlay networks and NATs are needed in order to address each container.

  ◇ Overlay networks add latency

- You have to pick between

  ◇ fewer containers with multiple applications per container (unique IP address for each container)

  ◇ multiple containers with fewer applications per container (Overlay networks / NAT)

8.18 Summary

- Kubernetes provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts

- Kubernetes supports a range of container tools, including Docker

- Containers are useful for packaging, shipping, and deployment of any software applications that are presented as lightweight, portable, and self-sufficient containers, that will run virtually anywhere.

- Microservices can benefit from containers and clustering.

- Kubernetes offers container orchestration

Objectives

Key objectives of this chapter

- Masters
- Nodes
- Pods
- Namespaces
- Resource Quota
- Authentication and Authorization
- Routing
- Registry
- Storage Volumes

9.1 What is Kubernetes?

- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications

- It groups containers that make up an application into logical units for easy management and discovery.

- Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.

- Whether testing locally or running a global enterprise, Kubernetes flexibility grows with you to deliver your applications consistently and easily no matter how complex your need is

- Kubernetes is open source giving you the freedom to take advantage of on-premises, hybrid, or public cloud infrastructure, letting you effortlessly move workloads to where it matters to you.

- Kubernetes can be deployed on a bare-metal cluster (real machines) or on a cluster of virtual machines.

9.2 Container Orchestration

- The primary responsibility of Kubernetes is container orchestration.

- Kubernetes ensures that all the containers that execute various workloads are scheduled to run physical or virtual machines

- The containers must be packed efficiently following the constraints of the deployment environment and the cluster configuration

- Kubernetes keeps an eye on all running containers and replaces dead, unresponsive, or otherwise healthy containers.

- Kubernetes can orchestrate the containers it manages directly on bare-metal or on virtual machines

- A Kubernetes cluster can also be composed of a mix of bare-metal and virtual machines, but this is not very common.

- Containers are ideal to package microservices because, while providing isolation to the microservice, they are very lightweight compared virtual machines. This makes containers ideal for cloud deployment

9.3 Kubernetes Basic Architecture

- At a very high level, there are three key concepts

  - Pods

  - Master

  - Minions (old term) / Nodes (new term)
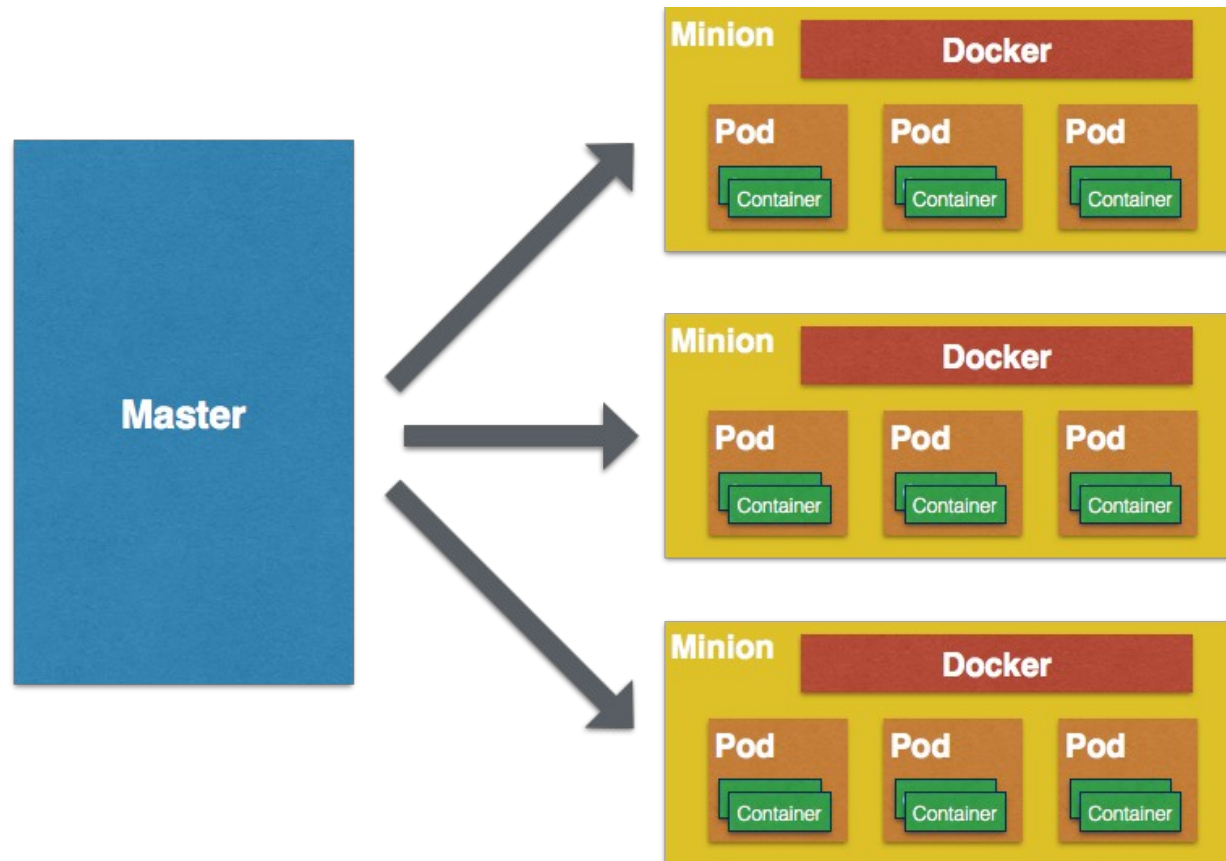
9.4 Kubernetes Detailed Architecture
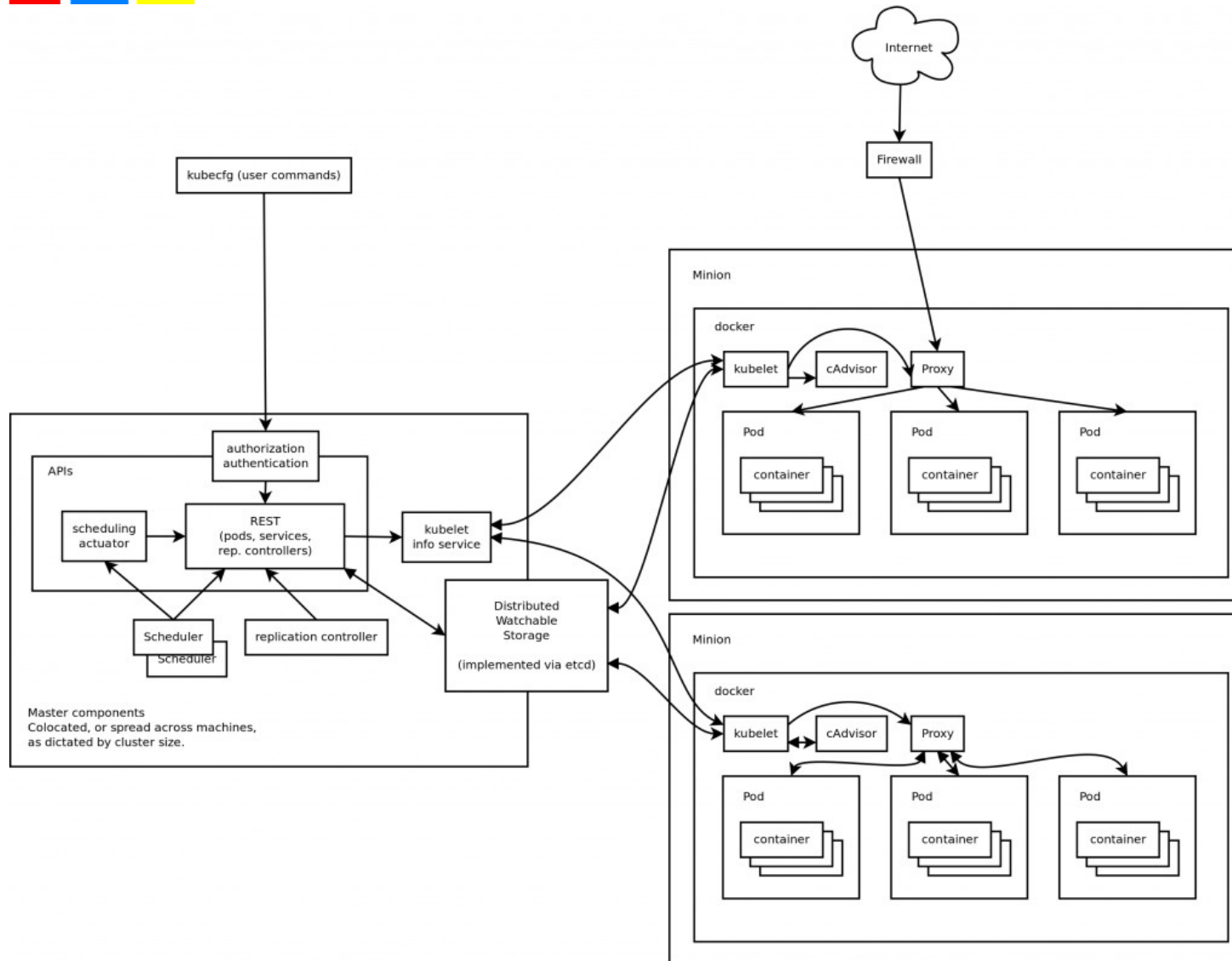
9.5 Kubernetes Concepts

- Cluster and Namespace

- Node

- Master

- Pod

- Label

- Annotation

- Label Selector

- Replication Controller and replica set

- Services

- Volume

- Secret

9.6 Cluster and Namespace

- Cluster

  ◇ A collection of physical resources, such as hosts storage and networking resources

  ◇ The entire system may consist of multiple clusters

- Namespace

  ◇ It is a virtual cluster

  ◇ A single physical cluster can contain multiple virtual clusters segregated by namespaces

  ◇ Virtual clusters can communicate through public interfaces

  ◇ Pods can live in a namespace, but nodes can not.

  ◇ Kubernetes can schedule pods from different namespaces to run on the same node

9.7 Node

- A single host

- It may be a physical or virtual machine

- It's job is to run pods

- Each node runs several Kubernetes components, such as a kubelet and a kube proxy

- kubelet is a service which reads container manifests as YAML files that describes a pod.

- Nodes are managed by a Kubernetes master

- The nodes are worker bees of Kubernetes and shoulder all the heavy lifting

- In the past they were called minions.

9.8 Master

- The master is the control plane of Kubernetes

- It consists of components, such as

  - API server

  - a scheduler

  - a controller manager

- The master is responsible for the global, cluster-level scheduling of pods and handling events.

- Often, all the master components are set up on a single host

- For implementing high-availability scenarios or very large clusters, you will want to have master redundancy.

9.9 Pod

- A pod is the unit of work on Kubernetes

- Each pod contains one or more containers

- Pods provide a solution for managing groups of closely related containers that depend on each other and need to cooperate on the same host

- Pods are considered throwaway entities that can be discarded and replaced at will (i.e. they are cattle, not pets)

- Each pod gets a unique ID (UID)

- Pods are always scheduled together and always run on the same machine

- All the containers in a pod have the same IP address and port space

- The containers within a pod can communicate using localhost or standard inter-process communication

- The containers within a pod have access to shared local storage on the node hosting the pod and is mounted on each container

- The benefit of grouping related containers within a pod, as opposed to having one container with multiple applications, are:

  ◇ Transparency – making the containers within the pod visible to the infrastructure enables the infrastructure to provide services to those containers, such as process

management and resource monitoring

⋄ Decoupling software dependencies – the individual containers maybe be versioned, rebuilt, and redeployed independently

⋄ Ease of use – users don't need to run their own process managers

⋄ Efficiency – because the infrastructure takes on more responsibility, containers can be more lightweight

9.10 Label

- Labels are key-value pairs that are used to group together sets of objects, often pods.

- Labels are important for several other concepts, such as replication controller, replica sets, and services that need to identify the members of the group

- Each pod can have multiple labels, and each label may be assigned to different pods.

- Each label on a pod must have a unique key

- The label key must adhere to a strict syntax

  - Label has two parts: prefix and name

  - Prefix is optional. If it exists then it is separated from the name by a forward slash (/) and it must be a valid DNS sub-domain. The prefix must be 253 characters long at most

  - Name is mandatory and must be 63 characters long at most. Name must begin with an alphanumeric character and contain only alphanumeric characters, dots, dashes, and underscores. You can create another object with the same name as the deleted object, but the UIDs must be unique across the lifetime of the cluster. UIDs are generated by Kubernetes

  - Values follow the same restrictions as names

9.11 Annotation

- Unlike labels, annotation can be used to associate arbitrary metadata with Kubernetes objects.

- Kubernetes stores the annotations and makes their metadata available

- Unlike labels, annotations don't have strict restrictions about allowed characters and size limits

9.12 Label Selector

- They are used to select objects based on their labels
- A value can be assigned to a key name using equality-based selectors, (=, ==, !=).
  - ◇ e.g.
    - role = webserver
    - role = dbserver, application != sales
- **in** operator can be used as a set-based selector
  - ◇ .e.g
    - role in (dbserver, backend, webserver)

9.13 Replication Controller and Replica Set

- They both manage a group of pods identified by a label selector

- They ensure that a certain number of pods are always up and running

- Whenever the number drops due to a problem with the hosting node or the pod itself, Kubernetes fires up new instances

- If you manually start pods and exceed the specified number, the replication replication controller kills some extra pods

- Replication controllers test for membership by name equality, whereas replica sets can use set-based selection

- Replica sets are newer and considered as next-gen replication controllers

9.14 Service

- Services are used to expose some functionality to users or other services

- They usually involve a group of pods, usually identified by a label

- Kubernetes services are exposed through endpoints (TCP/UDP)

- Services are published or discovered via DNS, or environment variables

- Services can be load-balanced by Kubernetes

9.15 Storage Volume

- When a pod is destroyed, the data used by the pod is also destroyed.

- If you want the data to outlive the pod or share data between pods, volume concept can be utilized.

9.16 Secret

- Secrets are small objects that contain sensitive info, such as credentials

- They are stored as plain-text in etcd

- They can be mounted as files into pods

- The same secret can be mounted into multiple pods

- Internally, Kubernetes creates secrets for its components, and you can create your own secrets

9.17 Resource Quota

- Kubernetes allows management of different types of quota

- Compute resource quota

  - Compute resources are CPU and memory

  - You can specify a limit or request a certain amount

  - Uses fields, such as requests.cpu, requests. memory

- Storage resource quota

  - You can specify the amount of storage and the number of persistent volumes

  - Uses fields, such as requests.storage, persistentvolumeclaims

- Object count quota

  - You can control API objects, such as replication controllers, pods, services, and secrets

  - You can not limit API objects, such as replica sets and namespaces.

9.18 Authentication and Authorization

- Permission rules can be added to the Kubernetes system for more advanced management

- Applying authentication and authorization is a secure solution to prevent your data being accessed by others.

- Authentication is currently supported in the form of tokens, passwords, and certificates.

- Authorization supports three modes:

  - RBAC (Role-Based Access Control)

  - ABAC (Attribute-Based Access Control) – lets a user define privileges via attributes in a file

  - Webhook – allows for integration with third-party authorization via REST web service calls.

9.19 Routing

- Routing connects separate networks

- Routing is based on routing tables

- Routing table instructs network devices how to forward packets to their destination

- Routing is done through various network devices, such as routers, bridges, gateways, switches, and firewalls
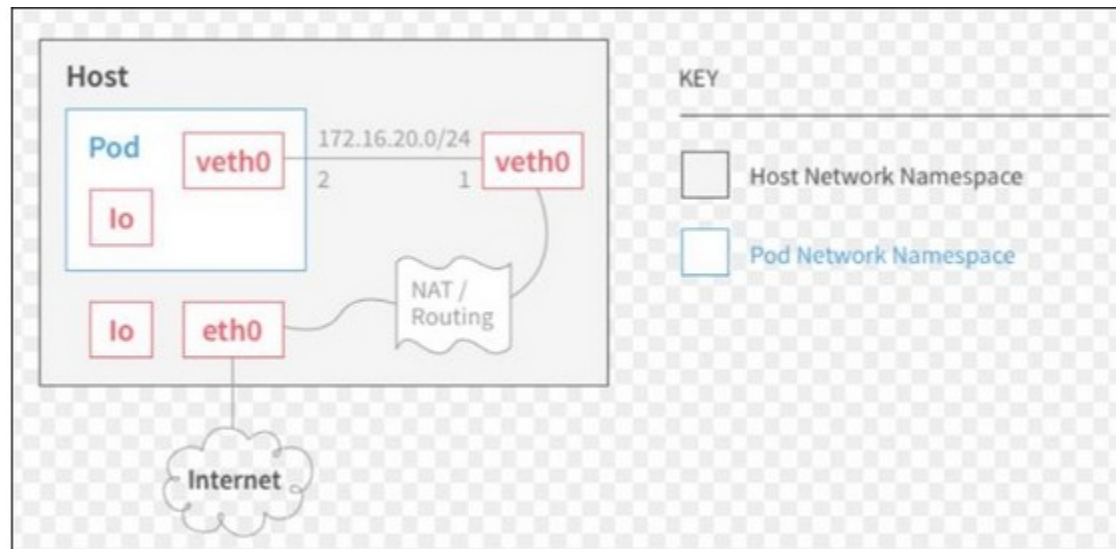
9.20 Registry

- Container images aren't very useful if it's only available on a single machine

- Kubernetes relies on the fact that images described in a Pod manifest are available across every machine in the cluster

- Container images can be stored in a remote registry so every machine in the cluster can utilize the images.

- Registry can be public or private.

- Public registries allow anyone to download images (e.g. Docker Hub), while private registries require authentication to download images (e.g Docker Registry)

- Docker Registry is a stateless, highly scalable-server side application that stores and lets you distribute Docker images.

- Docker Registry is open-source

- Docker Registry gives you following benefits

  ◇ tight control where your images are being stored

  ◇ fully own your images distribution pipeline

  ◇ integrate image storage and distribution tightly into your in-house development workflow

9.21 Using Docker Registry

- Default storage location is

```
/var/lib/registry
```

- Change storage location by creating and environment variable like this

```
REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/somewhere
```

- Start your registry (:2 is the version. Check Docker Hub for latest version)

```
docker run -d -p 5000:5000 –name registry registry:2.6
```

- Pull some image from the hub

```
docker pull ubuntu
```

- Tag the image so that it points to your registry

```
docker tag ubuntu localhost:5000/myfirstimage
```

- Push it

```
docker push localhost:5000/myfirstimage
```

- Pull it back

```
docker pull localhost:5000/myfirstimage
```

- Stop your registry and remove all data

```
docker stop registry && docker rm -v registry
```

9.22 Summary

- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications

- The primary responsibility of Kubernetes is container orchestration.

- At a high level, Kubernetes involves Pods, Master, and Nodes

- Kubernetes also involves labels, annotations, replication controllers, replica sets, and secrets

- Container images can be deployed to a public or private registry

Chapter 10 - Containerization

Objectives

Key objectives of this chapter
 	
   ■ Understanding Containerization
 	
   ■ Understanding Hypervisors
 	
   ■ Understanding Virtualization
 	
   ■ Understanding Docker

10.1 Containerization (Virtualization)

■ Virtualization is a technique of abstracting computer resources (hardware and networking) and allowing software (OSes and applications) to run in such environments as if it were a real computer

■ The collection of the virtualized resources are packaged in a Virtual Machine

■ Virtual Machines are managed by hypervisors (Virtual Machine Managers)

■ Virtualization helps drive server consolidation initiatives that leads to better resource utilization, management and driving operational costs down in public cloud computing, on-premise private clouds, and traditional enterprise data centers

■ Virtualization was fist introduced in mainframes back in 70s

10.2 Hypervisors

- A hypervisor (or Virtual Machine Monitor (VMM)) is a specialized software program that creates and runs virtual machines

- Hypervisors allow several operating systems (called "Guest OSes") to share a single hardware host at the same time in a way that

  ◇ Each guest OS receives its own fair share of virtualized resources (CPU, RAM, network, file-system, etc.)

  ◇ Guest OSes running on the same host do not affect others (allowing for a safe multi-tenant hosting arrangement)

10.3 Hypervisor Types

- There are two types of hypervisors:
  - ◇ Type 1 (native) hypervisors
  - ◇ Type 2 (hosted) hypervisors

10.4 Type 1 hypervisors

- Type 1 (native) hypervisors run directly on the host's hardware and manage guest OSes that are executed just a lever above the hypervisor
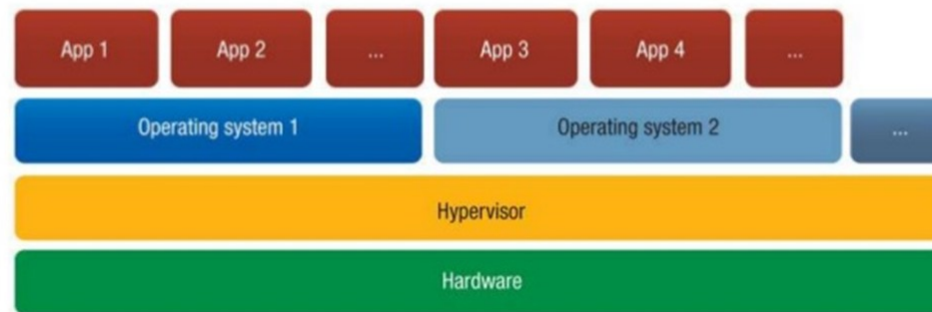
- ◇ Examples of Type 1 hypervisors: the Citrix XenServer, VMware ESX/ESXi, KVM, Microsoft Hyper-V, Oracle VM Server for SPARC, Oracle VM Server for x86 hypervisor

10.5 Type 2 hypervisors

- Type 2 (hosted) hypervisors run as a system program on a regular operating system environment (FreeBSD, Linux, or Windows.). The guest OSes run on top of the hypervisor

Type 2 – Hypervisor (Hosted)
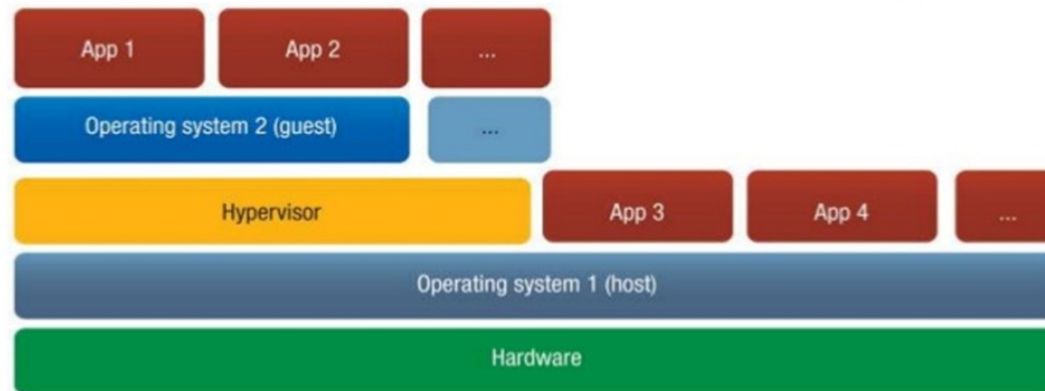
  - Examples of Type 2 hypervisors: VMware Workstation and VirtualBox are examples of Type 2 hypervisors

10.6 Type 1 vs Type 2 Processing

- Processing occurs differently between these two basic types of VM. Type 1 relies upon native hardware that supports virtualization whereas the Type 2 introduces a software abstraction layer to facilitate the virtualization

10.7 Paravirtualization

- Paravirtualization (PV) is a special case of system virtualization

- With PV, a hardware environment is not entirely simulated and the guest OSes get a coordinated access to the underlying physical hardware bypassing the VMM in certain cases
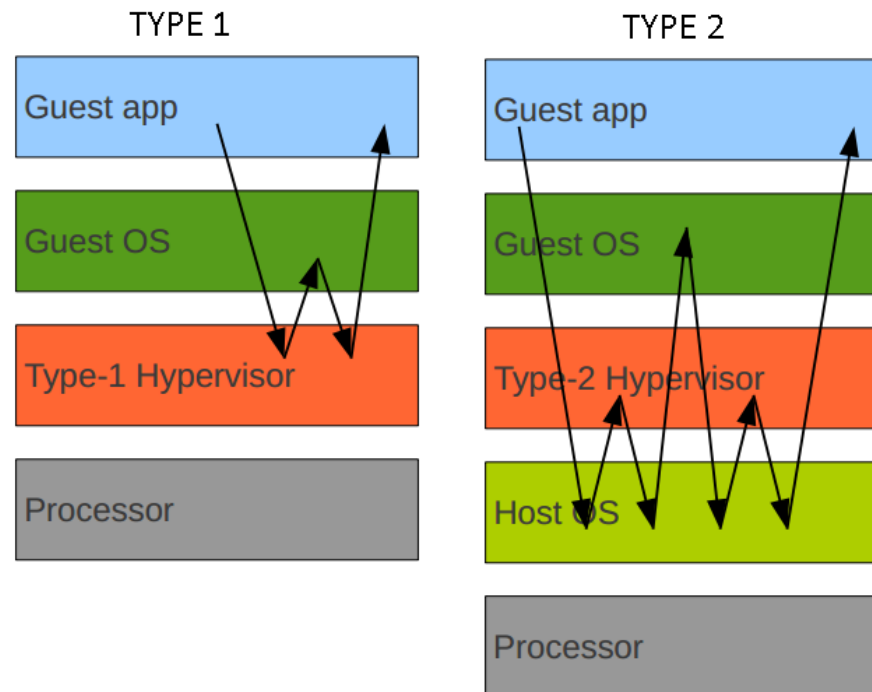
- PV requires the guest OSes to be modified (ported for the PV-API)

- A popular PV hypervisor *Xen* is developed under the Xen project using the GNU GPL v2 license (http://www.xenproject.org/)

- PV hypervisors have a simple design and offer excellent execution performance

  ◇ For example, with Xen you get performance degradation between 2% for a standard workload and 10 – 20% for a worst-case scenario

10.8 Virtualization Qualities (1/2)

- Transparent sharing of resources (memory, network, disk, CPU, etc.)
  - ⋄ decouples hardware from software
  - ⋄ hardware aggregates as a pool of sharable resources
- Live migration
  - ⋄ shift virtual servers between physical hardware instances while running
  - ⋄ facilitate zero downtime while still maintaining hardware
- Isolation
  - ⋄ limits security exposure
  - ⋄ reduces spread of risk

10.9 Virtualization Qualities (2/2)

- Management

    ◇ single point of control across all VMs

    ◇ ease deployment burden through repetitive scripts and templates

    ◇ block-level rollbacks to prior snapshots in the event of failure

- High availability

    ◇ boot virtual servers on alternate hardware in the event of primary hardware failure

    ◇ execute multiple instances of a virtual server across multiple hosts

10.10 Disadvantages of Virtualization

- System virtualization tools or emulators need to load and boot up a full image of the guest OS and, basically, need to emulate a complete machine, which results in a high operational overhead

- The virtualization overhead sharply increases in proportion to the number of guest OSes running on the same physical host:

  - CPU (CPU caches efficiencies fall)

  - Memory (swapping increases)

  - IO / Networking (contention increases)

  - The underlying scheduler contention

10.11 Containerization

- Containerization is a lightweight alternative to full machine virtualization

    ◇ It is often called virtualization environment, or OS-level virtualization

- A container encapsulates an application running inside its own operating environment which is derived from the underlying host OS

- Containerization has been popularized by cloud-like processing environments that require fast server boot-up time

- At the moment, the most widely used technology behind containerization is **L**inu**X** **C**ontainers (LXC), which is a userspace interface for the Linux kernel containment features (cgroups and namespaces)

- The main containerization action is happening in the Linux space (and x84 (32-bit) & x64 (64-bit) machine architecture)

10.12 Virtualization vs Containerization

- Both offer multi-tenancy for guests (OSes and applications)

- Virtualization is about translating communication between the hosted OSes and hypervisor

- Containerization is "native" in that containers share the host OS's kernel

  ◇ Containers' OS is the same as the hosts' OS

- Virtualization allows to run multiple guest OSes on the same host, while containerization is limited to the OS type the host uses

- Traditional virtualization offers better protection from "rogue" tenants

10.13 Where to Use Virtualization and Containerization

- Virtualization belongs to the Enterprise space (and big Ops budgets)

- If you an infrastructure provider and a Linux shop competing on price, use containerization

  - ◇ Platform-as-a-Service (PaaS) vendors such as Heroku, OpenShift, and Cloud Foundry use Linux containerization

- Containerization is the way to go if you are cost-conscious organization trying to save every nickel (most Linux containerization systems are free)

- DevOps can hugely benefit from containerization as well (fast system provisioning: build, test, integration machines, etc.)

10.14 Popular Containerization Systems

- Linux containerization systems:
    - ◇ **LXC** (more on LXC a bit later ...)
    - ◇ **Docker** (more on Docker a bit later ...)
    - ◇ **OpenVZ** (more on OpenVZ a bit later ...)
    - ◇ **Linux-VServer**
- Non-Linux systems containerization systems:
    - ◇ **Oracle Solaris Zones (Containers)** (more on them a bit later ...)
    - ◇ **IBM AIX Workload Partitions**
    - ◇ **FreeBSD Jails**

10.15 What are Linux Containers

- LinuX Containers (LXC) is an OS-level virtualization environment that allows multiple Linux systems to run on a single physical host

- LXC provides a user API and a toolset for interfacing with the Linux kernel containment features (cgroups and namespaces)

- The main differentiating factor between LXC and some other systems is that it runs on the unmodified Linux kernel of various Linux distros

- The LXC system is written in a number of programming languages, including C, Python 3, Lua, and others

- Initial release was in August 2008

10.16 Docker

- Docker is an open-source system for creating virtual environments

- Runs on any modern-kernel Linux distributions

- To communicate with the underlying host kernel, Docker uses virtualization interfaces based on a variety of systems:

  - *libvirt*

  - *LXC* (**L**inu**X** **C**ontainers)

  - *systemd-nspawn*

- You can install Docker inside a VirtualBox and run it on OS X or Windows

- Docker can be booted from the small footprint Linux distribution *boot2docker*

- Written in the Go programming language

10.17 OpenVZ

- OpenVZ is a Linux OS-level virtualization technology that leverages Linux modern features kernel

- OpenVZ is similar to FreeBSD jails and Oracle Solaris Zones

- Requires Linux kernel patching

  ◇ As of version 4.0, OpenVZ can work with <u>unpatched</u> Linux 3.x kernels offering a reduced functionality

- Written in C

10.18 Solaris Zones (Containers)

- Oracle Solaris Zones (previously known as Solaris Containers) is an OS-level virtualization technology for Interl-based and SPARC systems

- Oracle Solaris Zones are an integral part of the Oracle Solaris 10 and up

- Oracle promotes this technology as a way to maintain the one-application-per-server deployment model that share same hardware resources

- The first public release was in February 2004

10.19 What is Docker

- Docker is an open-source (and 100% free) project for IT automation

- You can view Docker as a system for creating virtual environments which are extremely lightweight virtual machines

- Docker allows the deployment of applications and their dependencies inside Linux containers supporting the multi-tenancy deployment model on a single host

  ◇ A container is a group of controlled processes associated with a separate tenant executed in isolation from other tenants

- Written in the Go programming language

10.20 Where Can I Ran Docker?

- Docker runs on any modern-kernel Linux distributions

    ◇ You can install Docker inside a VirtualBox and run it on OS X or Windows

    ◇ Docker can be booted from the small footprint Linux distribution *boot2docker*

10.21 Docker and Containerization on Linux

- Docker leverages resource isolation features of the modern Linux kernel offered by cgroups and kernel namespaces

  ◇ The cgroups and kernel namespaces features allow creation of strongly isolated containers acting as very lightweight virtual machines running on a single Linux host

- Docker helps abstract operating-system-level virtualization on Linux using abstracted virtualization interfaces based on *libvirt*, *LXC* (**L**inu**X C**ontainers) and systemd-nspawn

  ◇ As of version 0.9, Docker has the capability to directly use virtualization facilities provided by the Linux kernel via its own *libcontainer* library

10.22 Linux Kernel Features: cgroups and namespaces

- The control group kernel feature (cgroup) is used by the Linux kernel to allocate system resources such as CPU, I/O, memory, and network subject to limits, quotas, prioritization, and other control arrangements

- The kernel provides access to multiple subsystems through the *cgroup* interface

    ◇ Examples of subsystems (controllers):

    - The memory controller for limiting memory use

    - The *cpuacct* controller for keeping track of CPU usage

- The cgroups facility was merged into the Linux kernel version 2.6.24

- Systems that use cgroups: Docker, Linux Containers (LXC), Hadoop, etc.

- The namespaces feature is a related to cgroups facility that enables different applications to act as separate tenants with completely isolated views of the operating environment, including users, process trees, network, and mounted file systems

10.23 The Docker-Linux Kernel Interfaces
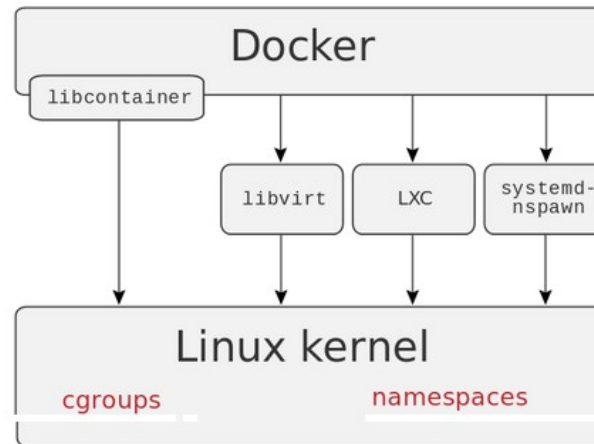
Source: Adapted from http://en.wikipedia.org/wiki/Docker_(software)

10.24 Docker Containers vs Traditional Virtualization

- System virtualization tools or emulators like KVM, Xen, HyperV, VMware, etc. boot virtual machines from a complete guest OS image (of your choice) and basically emulate a complete machine, which results in a high operational overhead

- Virtual environments created by Docker run on the existing kernel's image of the host's OS without a need for a hypervisor

  ◇ This leads to very low overhead and significantly faster container start-up time

- Docker-provisioned containers do not include or require a separate operating system (it runs in the host's OS)

  ◇ This circumnstance puts a significant limitation on your OS choices

10.25 Docker Containers vs Traditional Virtualization

- Overall, traditional virtualization has advantages over Docker in that you have a choice of guest OSes (as long as the machine architecture is supported)
  - You can get only some (limited) choice of Linux distros
    - You still have some choice: e.g. you can deploy a Fedora container on a Debian host
  - You can, however, run a Windows VM inside a Linux machine using virtual machine emulators like VirtualBox (with less engineering efficiency)
- With Linux containers, you can achieve a higher level of deployed application density compared with traditional VMs (10x more units!)
- Docker runs everything through a central daemon which is not a particularly reliable and secure processing model

10.26 Docker as Platform-as-a-Service

- Docker defines an API for creating, deploying and managing containers that make up highly distributed systems spanning multiple physical machines

  - ◊ Docker-based systems can also efficiently run multiple isolated applications on a single physical machine

- On-demand provisioning of applications by Docket supports the Platform-as-a-Service (PaaS)–style deployment and scaling

10.27 Docker Integration

- Docker can be integrated with a number of IT automation tools that extend its capabilities, including
  - Ansible
  - Chef
  - Jenkins
  - Puppet
  - Salt
- Docker is also deployed on a number of Cloud platforms
  - Amazon Web Services
  - Google Cloud Platform
  - Microsoft Azure
  - OpenStack
  - Rackspace

10.28 Docker Services

- Docker deployment model is application-centric and in this context provides the following services and tools:

    ◇ A uniform format for bundling an application along with its dependencies which is portable across different machines

    ◇ Tools for automatic assembling a container from source code: make, maven, Debian packages, RPMs, etc.

    ◇ Container versioning with deltas between versions

10.29 Docker Application Container Public Repository

■ Docker community maintains the repository for official and public domain Docker application images: https://hub.docker.com/account/signup

10.30 Competing Systems

- Rocket container runtime from CoreOS (an open source lightweight Linux kernel-based operating system)

- LXD for Ubuntu from Canonical (the company behind Ubuntu)

- The LXC (Linux Containers), used by Docker internally

- Many more are on the way …

- Other systems exist for non-Linux OSes

10.31 Docker Command-line

- The following commands are shown as executed by the root (privileged) user:

`docker run ubuntu echo 'Yo Docker!'`
  - ◇ This command (not very useful) will create a docker container on the fly and execute the echo command on it and then shuts down

`docker ps -a`
  - ◇ This command will list all the containers along with their IDs created by Docker

10.32 Starting, Inspecting, and Stopping Docker Containers

`docker start -i <container_id>`

⬧ This command will start an existing stopped container in interactive (**-i**) mode (you will get the target system shell on start-up)

`docker inspect <container_id>`

⬧ This command will provide JSON-encoded information about the running container identified by *container_id*

`docker stop <container_id>`

⬧ This command will stop the running container identified by *container_id*

10.33 Docker Benefits

- Rapid application deployment

- Portability across machines

- Version control and component reuse

- Sharing

- Lightweight footprint and minimal overhead

- Simplified maintenance

10.34 Summary

- Containerization can play a very important role in DevOps.

- Containerization promotes rapid application, portability across machines, and version control

- It can be used in development and

Chapter 11 - Collaboration

Objectives
Key objectives of this chapter

Key objectives of this chapter

- Understanding what JIRA is
- Understanding JIRA Products differences
- Understanding issue types

11.1 What is JIRA?

- DevOps heavily utilizes agile principles

- JIRA is an agile Issue tracking and agile project management solution

- Web-based product developed by Atlassian

- Not an acronym

- Truncation of *Gojira* (Japanese name for Godzilla)

- Reference to main competitor, Bugzilla

11.2 License

- Commercial software

- On-premises / cloud

- Free for open source projects and to organizations that are non-academic, non-commercial, non-governmental, non-political, non-profit use

11.3 JIRA Technical Specifications

- Written in Java

  ◇ Can create custom plugins

- Back-end

  ◇ MySQL, PostgreSQL, Oracle, SQL Server

11.4 Issues

- Bug

- Epic

- Story

- Task

- Sub-task

11.5 Who uses JIRA

- 50,000 customers
  - ◇ Adobe
  - ◇ AUTODESK
  - ◇ Audi
  - ◇ BBC
  - ◇ Deutsche Bank
  - ◇ eBay
  - ◇ Facebook
  - ◇ HSBC
  - ◇ NASA
  - ◇ VISA
  - ◇ Yellow Pages
  - ◇ …

11.6 JIRA Products

- JIRA Core

- JIRA Software

- JIRA Service Desk

11.7 JIRA Core

- Meant for non-development / business teams

- Project and task management solution

- Tasks, sub-tasks

- Workflows

- Tracking

- No agile

11.8 JIRA Software

- Meant for development teams

- JIRA Core + JIRA Agile

- Stories and epics in addition to tasks, sub-tasks

- Plan, track, release, report

- JIRA workflow engine

- Agile : Scrum, Kanban

- Integrates into development tools

11.9 JIRA Service Desk

- Meant for IT or business service desks

- Ticketing management system

- Incident, change and problem management

11.10 What a typical project involves?

- Project
- Component
- Version
- Issue
- Sprint
- Agile Board
- Report

11.11 JIRA Integration

◇ Add-ons

- Jenkins

- git

- Mercurial

- ...

11.12 Integrating JIRA into Jenkins

- Install Jenkins add-on

- On the menu bar click Administration > Add-ons.

- In Search text box type in 'Jenkins', without quotes, and press enter on the keyboard.

- Click Install button on right side of **Jenkins/Hudson build status gadget**.

- On **Licensed and ready to go!** dialog box click **Close** link.

- Click **Manage** button on right side of **Jenkins/Hudson build status gadget**.

11.13 Summary

- JIRA is a widely used agile issue tracking solution.

- JIRA Software supports Scrum and Kanban OOB.

- Issue is a very generic term and can represent bug, epic, story, task or a sub-task.

Chapter 12 - The Journey

Objectives
Key objectives of this chapter

- Understanding Continuous Integration & Delivery with DevOps

- Understanding DevOps in the enterprise

- Scaling DevOps

- DevOps patterns and anti-patterns

12.1 Agile Development

- Agile Development begins with Extreme Programming (XP)

  ◇ Invented/Promoted by Kent Beck and associates

- Beck describes XP as:

  ◇ A philosophy of software development based on the values of communication, feedback, simplicity, courage, and respect

  ◇ A body of practices proven useful in improving software development.

  ◇ A set of complementary principles, intellectual techniques for translating the values into practice, useful when there isn't a practice handy for your particular problem.

  ◇ A community that shares these values and many of the same practices.

12.2 Agile Development (cont'd)

- As time went on, people took inspiration from XP and developed other approaches with much the same goals

  - ◇ Speed

  - ◇ Include the Customer Early

  - ◇ Don't wait til you understand the entire problem - you'll understand it while you fix it

  - ◇ Deliver usable value at every iteration

  - ◇ Make reasonable use of modeling and development tools

12.3 Agile Development (cont'd)

- Two themes emerge from XP and Agile Development

  - ◇ Trust

    - ■ Vendor, customer, managers, developers all need to trust each other

  - ◇ Automation

    - ■ If we're going to go fast, we need tools that make it easy to produce software

- Trust is a social problem – solution is all about different approaches to process, requirements gathering, project management, etc.

- Automation is about tools – and Continuous Integration is a primary tool of Agile Development

12.4 What is Continuous Integration

- "Integrate and test changes after no more than a couple of hours."

    - Beck & Andres "Extreme Programming Explained"

    - Integrate and build the complete product

    - If a website, build the website

    - If a GUI install, build the installer

    - Etc.

12.5 What is Continuous Integration (cont'd)

- Purposes

  ◇ You find out quickly about integration problems

  ◇ Immediately evident if a code change "breaks the build"

  ◇ Prevents a long drawn-out integration step at the end of code changes

  ◇ Should be complete enough that eventual first deployment of the system is "no big deal".

12.6 What is Continuous Integration (cont'd)

- Usually, CI goes along with Test-First design and automated QA
  - ◇ Run the unit-test suite and smoke testing to ensure the build isn't broken
- Clearly, automatic build is a pre-requisite
  - ◇ So we need a build system – Maven for instance
- Can be synchronous or asynchronous
  - ◇ Asynchronous – Integration happens automatically on code committal
  - ◇ Synchronous – Trigger the build manually after a development session

12.7 What is Continuous Integration (cont'd)

- Side Effects
    - ◇ Generate development reports
    - ◇ Install to QA, User Test, etc
    - ◇ Always have an installable artifact
- It's a great time to generate development metrics
    - ◇ E.g. Code Coverage, Standards Compliance, Static Analysis

12.8 Typical Setup for Continuous Integration
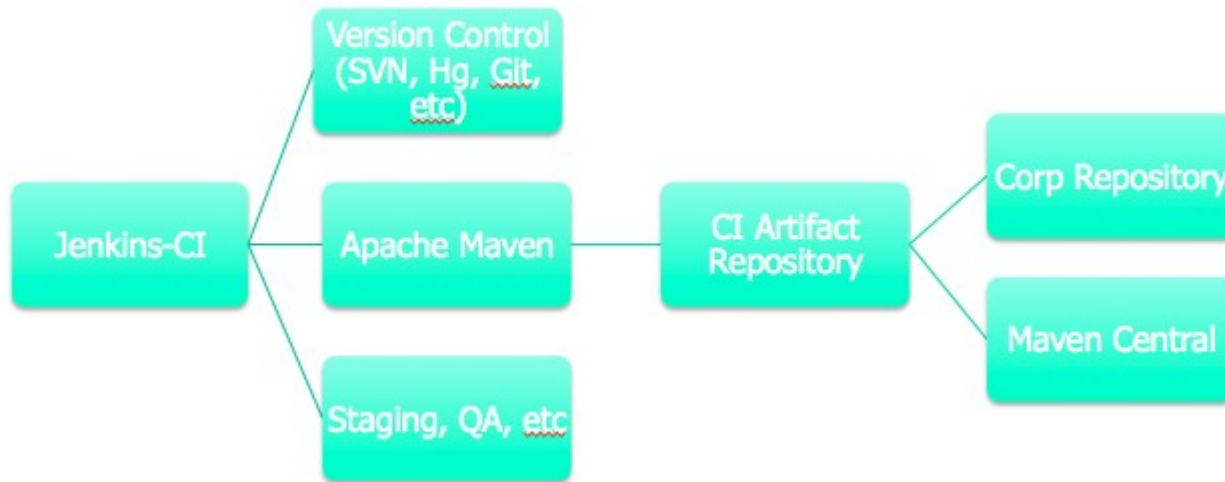
12.9 Typical Setup for Continuous Integration

- Notes:
  - CI system gets the code directly from version control
  - Build is independent of any local artifacts that are on the developer's machine
    - Controlled links to corporate repository and Maven Central
    - Goal is to ensure that the package can be built from the corporate repository
  - Jenkins can have connections to a deployment environment
    - Production Staging
    - User Acceptance Testing
    - QA
    - Load Test
    - Etc…
  - It turns out that if we use Maven, we ABSOLUTELY NEED a local repository manager

12.10 DevOps in the Enterprise

- No separate 'enterprise DevOps' with different tools and practices that applies only to companies with a large number of employees

- Application or implementation of principles can differ in larger organizations, not principles

- Concern that DevOps principles can be applied only to small startups, making it unsuitable for enterprise organizations or legacy systems is incorrect

  ◇ High performance is achievable if you architect with testability & deployability in mind (Puppet Labs 2015 State of DevOps Report)

- DevOps can be applied to organizations of any size and to any well-designed and architected software project, even legacy code running on mainframes.

12.11 Scaling DevOps

- Successful organizations must know how to scale – that is, to grow or shrink as needed

- Scaling can mean several things:

  ◇ Expanding the customer base

  ◇ Growing revenue

  ◇ Expanding a project or team to meed demand

  ◇ Maintaining or improving a ratio of people to systems or money spent

  ◇ Growing faster than competitors

  ◇ …

- Trying to identify a single mechanism that solves for all things, especially in eliminating humans from the system, is fundamentally wrong

- E.g. there is no one design for a building that can address all possible requirements in all cases, and the same applies for software projects and organizations as well

- Recognizing how a system should behave allows you to construct a set of prioritized systems for your current environment

12.12 Scaling DevOps (Organization Structure)

- Restructuring the way teams are organized may facilitate scaling

- Small cross-functional teams, for a single project or product, have everything needed to get their product off the ground

- Cross-team communication is very critical to have an effective organization overall

- If single-function teams are still communicating and working well with the rest of the organization, don't reorganize just for the sake of reorganization

12.13 Scaling DevOps (Locality)

- Communication is critical whether teams are collocated or distributed

- Distributed teams bring a new set of IT- and infrastructure-related considerations

- Understanding cultural nuances, and expectations of customers at a global level, also informs how to staff local support offices

12.14 Scaling DevOps (Team Flexiblity)

- A lot of research has been done on effective team size

- Teams that are too small generally lack the resources, person-hours and knowledge, to get everything done

- Larger teams, especially over 9-10 people, can find it much more difficult to make timely and effective decisions

- Large teams can also fall prone to groupthink, suppressing individual opinions in the interest of overall group harmony, which can diminish their creativity and problem-solving abilities

- One common complaint about larger companies is that there is too much bureaucracy to get anything done

- Organizational structures that lock people into role lead to a focus on optimizing work for the 'me', not the 'we'.

- Smaller cross-functional and self-organizing teams are generally more effective

- More smaller teams should be created rather than increasing the same of some existing team. E.g. Scrum of Scrums

12.15 Scaling DevOps (Teams: Hiring as Scaling)

- New hires should be trained to understand and use DevOps principles

- There's value in automation, but it will never be a complete replacement for humans

- Adding manpower to a late software project makes it later

- New team members take time before they are productive

- Even the most experienced members will take some time to get used to a new project an a new code base

- At times existing team members spend time helping the new people get up to speed

- Subcontracted work can result in saving in terms of dollars on a budget sheet but increased costs in terms of decreased collaboration between individuals and teams

- Outsourcing can create functional silos where people tend to hoard knowledge

- Be wary of conflicts of responsibility between in-house and outsources teams

12.16 Scaling DevOps (Teams: Employee Retention)

- Keeping experienced and trained employees is important since employee retention effects team productivity and morale

- Whenever possible experienced employees should be retained by providing them monetary and / or non-monetary benefits

- Non-monetary benefits can be:

  - Remote work opportunities

  - Education opportunities

  - Flexible work hours

  - Work-life balance

  - Paid leave

  - Retirement plans

  - Health insurance

  - Casual dress code

  - Transportation benefits

  - Gender-neutral facilities

  - On-site daycare

◇ …

12.17 DevOps Myths

- DevOps only involves developers and system administrators

  ◇ There's no definitive list of which teams or individuals should be involved, just as there is no one-size-fits-all to implement DevOps

- DevOps is a job title

- DevOps is relevant only to small startups

- You need DevOps certification

- DevOps means doing all the work with half the people

- There's one "right way" to do DevOps

- It will take X weeks/months to implement DevOps

- DevOps is all about the tools

- DevOps is about automation that results in fixed delivery time

- DevOps will become obsolete or replaced with something new

12.18 DevOps Anti-Patterns (Blame Culture)

- A culture that tends toward blaming and punishing people when mistakes are made, either at an individual or an organization level

- In this culture, a root cause analysis as part of retrospective is generally misapplied

- If this analysis happens to point toward a person's actions as being the "root cause" that person will be blamed, reprimanded, or even fired

- This culture is more prevalent when there are external auditors, or where there is some top-down mandate to improve performance according to a given set of metrics

- Individual contributors will be motivated to try to shift blame away from themselves and their own teams to somebody else

- Such a culture encourages self preservation and doesn't lend to a culture of openness and collaboration

12.19 DevOps Anti-Patterns (Silos)

- It's the mentality of teams that do not share their knowledge with other teams in the same company

- Instead of having common goals or responsibilities, siloed teams have very distinct and segregated roles

- Usually each team in a siloed environment uses different tools or processes to complete similar tasks

- Silos can also come from a lack of communication and collaboration between teams

- Cross-functional teams are often touted as being the anti-silos

12.20 DevOps Anti-Patterns (Root Cause Analysis)

- RCA is a method to identify contributing and "root" causes of events or near-misses/close calls and the appropriate actions to prevent recurrence

- It's an iterative process that is continued until all organizational factors have been identified or until data is exhausted

- One method of identifying root causes is the '*5 Whys'.* This method entails asking "why" until the root causes are identified

- Often RCA is associated with determining a single root cause

- This limits RCA's usefulness

12.21 DevOps Anti-Patterns (Human Error)

- The idea that a human being made a mistake that directly caused a failure is often cited as the root cause in a root cause analysis

- It tends to assume that human mistakes are made due to simple negligence, fatigue, or incompetence, and neglects to investigate the myriad factors that contributed to the person making the decision or taking the action they did

12.22 DevOps Patterns For Success

- Infrastructure and Applications must be rapidly deployed in a verifiable, repeatable and safe manner

- Key Patterns

  - Cloud

  - Automation

  - Culture

12.23 DevOps Patterns For Success (Cloud)

- Migrate applications to the cloud

- Support hybrid environments

- Support multi-cloud environments

- Support heterogeneous environments

12.24 DevOps Patterns For Success (Automation)

- Infrastructure as code

- It should be:

  - ◇ Versionable

  - ◇ Testable

  - ◇ Repeatable

- Avoid fragmentation of tools and processes for doing the same thing

12.25 DevOps Patterns For Success (Culture)

- Eliminate silos

- Smaller cross-functional and self-organizing teams

- Communication and collaboration

12.26 Summary

- There's no separate "Enterprise DevOps"

- Scaling DevOps involves various factors

- Avoid DevOps anti-patterns and follow the patterns for success

Appendix - Non-Java Jenkins Jobs

Objectives

Key objectives of this chapter

- Understanding Jenkins Jobs

- Building and Executing .NET based Jenkins Jobs

- Building and Executing C++ based Jenkins Jobs

- Building and Executing Node.js based Jenkins Jobs

- Executing PowerShell based Jenkins Jobs

Appendix.1 Jenkins Jobs

- In Jenkins the definition of a build process is called a **Job**

- Jobs are created using the main menu on the Jenkins home page.

- A job can be composed of multiple steps.

- Examples of steps include connecting to source code repository, executing shell / batch scripts, and sending emails.

- More types of steps can be added by installing plugins in Jenkins.

Appendix.2 Non-Java Jobs

- Source Code repository

  ◇ Git

  ◇ SVN

  ◇ TFVC

  ◇ ...

- .NET Project

  ◇ MSBuild

  ◇ MSTest

  ◇ …

- Node.js

  ◇ node.exe

  ◇ npm.exe

- PowerShell

- C++

Appendix.3 Building .NET Projects with Jenkins

- Install MSTest plugin in Jenkins

- Configure the MSBuild plugin

- Install MSTestRunner plugin in Jenkins

- Configure the MSTestRunner plugin

- Create a Jenkins Job to build and test the .NET project

- Deploy the project

- Run the Job

Appendix.4 Installing MSTest Plugin in Jenkins

■ Install MSTest plugin in Jenkins
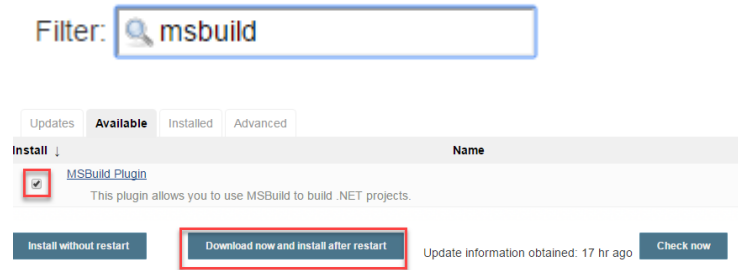
Appendix.5 Configuring the MSBuild Plugin

■ Manage Jenkins > Global Tool Configuration

Manage Jenkins

⚠ New version of Jenkins (2.32.2) is available for download (changelog). [Or Upgrade Automatically]

Configure System
Configure global settings and paths.

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Configure Credentials
Configure the credential providers and types

Global Tool Configuration
Configure tools, their locations and automatic installers.

■ Configure MSBuild

Gradle

Gradle installations   [Add Gradle]

List of Gradle installations on this system

MSBuild

MSBuild installations   [Add MSBuild]

List of MSBuild installations on this system

Ant

Ant installations   [Add Ant]

List of Ant installations on this system

■ Add path to msbuild.exe

Appendix - Non-Java Jenkins Jobs

MSBuild

MSBuild installations

MSBuild

Name

MSBuild 2015

Path to MSBuild

C:\Program Files (x86)\MSBuild\14.0\Bin

Default parameters

☐ Install automatically

Delete MSBuild

Add MSBuild

List of MSBuild installations on this system

Ant

Ant installations

Add Ant

List of Ant installations on this system
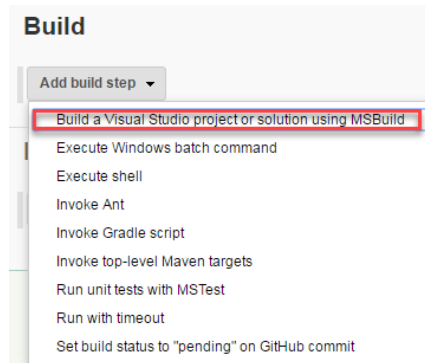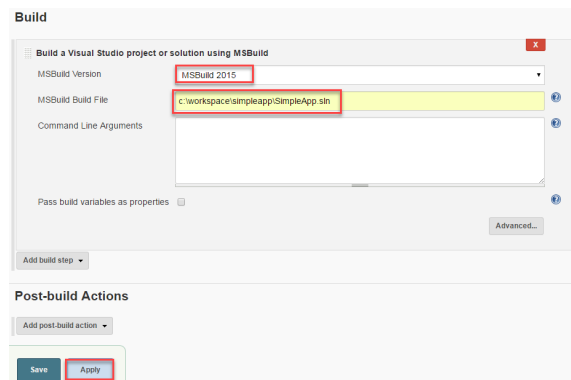
Maven

Save   **Apply**

Appendix.6 Creating a Jenkins Job and Specify a Build Step
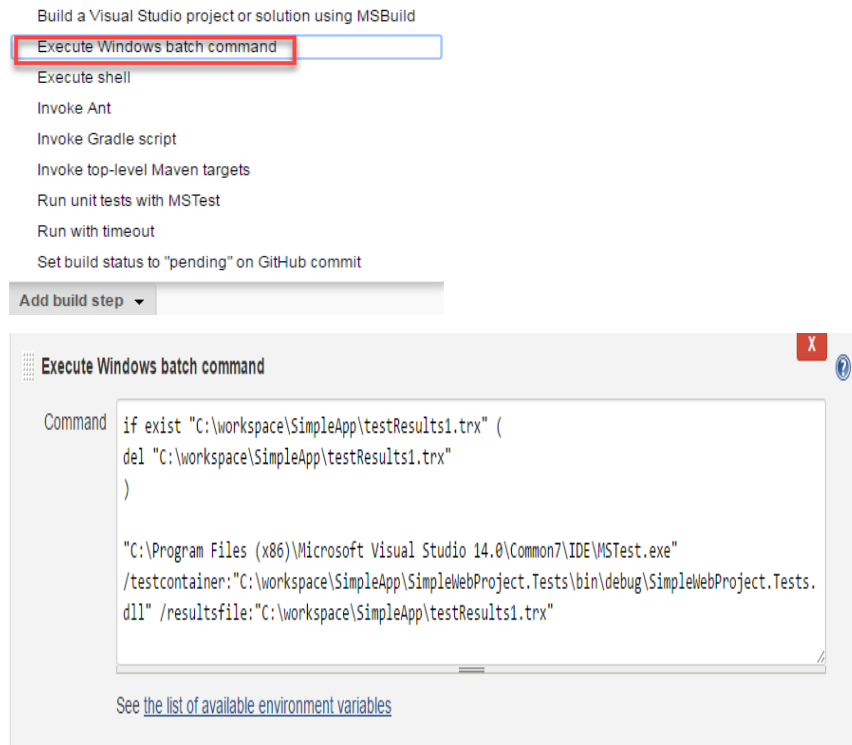
- ■ Add a build step

- ■ Specify solution (.sln) file

Appendix.7 Specifying a Step for Running Unit Tests

■ You can define a step for running unit tests in various ways

◇ MSTestRunner plugin

◇ Execute Windows batch command

■ Define a Windows batch command

Appendix.8 Adding a Step for Deploying the .NET Project

■ Configure **MSBuild Build File** and **Command Line Arguments**.

Appendix.9 Building a Node.js Application with Jenkins

- Execute Windows batch command

`npm install`

- Running unit tests with **Mocha**

```
cd c:\workspace\NodeApp\tests

c:\workspace\NodeApp\node_modules\.bin\mocha --reporter mocha-junit-
reporter --reporter-options mochaFile="%WORKSPACE%\test-results.xml"
```

- Publish test results with post-build action

Appendix.10 Node.js Plugin

- Running Node projects from a shell script will be fine for many projects

- There is also a Node.js plugin

    - It only runs on Linux

    - Provides auto-installers

    - Lets you create multiple Node environments

        - Different versions

        - Different sets of global modules

Appendix.11 Provides direct Pipeline supportBuilding a C++ Project with Jenkins

- Install CMake plugin in Jenkins

  ◇ CMake is used for generating a MakeFile

- The CMake plugin is useful if you want Jenkins to use a specific version of CMake

- Add a CMake build step

- Run "make all" from the build directory

Build

CMake Build

CMake installation	InSearchPath
Script Generator	Unix Makefiles
Source Directory	cmake-gtest/src/
Build Type	Debug
Build Directory	cmake-gtest/build
Clean Build	☐

Advanced...

Build tool invocations

☐ Run build tool

Use cmake ☐

Arguments

Env. Variables

Delete

Add build tool invocation

Delete

Execute shell

Command
```
cd cmake-gtest/build
make all
```

s

Appendix.12 Executing PowerShell Scripts with Jenkins

- Install PowerShell plugin in Jenkins
  - ◇ Manage Jenkins > Manage Plugins
- Add a PowerShell build step

- Run the Jenkins job

Appendix.13 Summary

- Jenkins has various plugins for building / executing non-Java projects, such as .NET, Node.js, C++, PowerShell etc.

Appendix - Introduction to Gradle

Objectives

Key objectives of this chapter
 	
   ■ DevOps introduction
 	
   ■ Business value of DevOps
 	
   ■ Standing up DevOps capability

Appendix.1 What is Gradle

- Gradle is a flexible general purpose build tool like ANT.

- Powerful dependency management

- Full support for your existing Maven or Ivy repository infrastructure

- Ant tasks are also supported

- Groovy language is supported for writing scripts

- It is free and an open source project, and licensed under the Apache Software License (ASL)

- Gradle can increase productivity, from single project builds to huge enterprise multi-project builds.

Appendix.2 Why Groovy

- ANT uses declarative XML based style

- Uses DSL based on Groovy which makes it more powerful.

- Gradle's main focus is on Java projects, but it's still a general purpose build tool

- Groovy offers transparency for Java people.

- Groovy's base syntax is the same as Java's

Appendix.3 Build Script

- buid.gradle file

- Written in Groovy

- Composed of tasks.

  ◇ Similar to ANT target.

- Tasks is composed of actions

  ◇ doFirst, doLast

Appendix.4 Sample Build Script

```
task hello1 << {
    println 'Hello World!'
}

task hello2 {
    doFirst {
        print 'This is '
    }
    doLast {
        println 'a test!'
    }
}
```

Appendix.5 Task Dependency

- A task can depend on one or more tasks

- If a dependent task already exists then following syntax can be used

```
task <task_name>(dependsOn: dependentTask)
```

- If dependent task doesn't exist then it must be enclosed in quotes like this:

```
task <task_name>(dependsOn: dependentTask)
```

Appendix.6 Plugins

- Plugins can be defined in build script to make more tasks available to Gradle

- e.g. apply plugin: 'java'

Appendix.7 Dependency Management

- A project can make use of additional libraries that are not already part of current project.

- Java projects can connect to various repositories, such as Maven Central and JCenter.

- Repositories can be defined like this:

```
repositories {
    mavenCentral()
}
```

- Dependencies can be defined like this:

```
dependencies {
    testCompile "junit:junit:4.12"
}
```

Appendix.8 Gradle Command-Line Arguments

- gradle tasks : displays tasks defined in build script, including plugin provided tasks

- gradle -q : suppresses log messages and runs default tasks

- gradle build: compiles code, generates jar file, and runs unit tests.

- gradle clean: cleans the build directory

- gradle test: runs unit tests

Appendix.9 Summary

- Gradle is an open source general purpose build tool

- Gradle offers more flexibility since it uses DSL language, which is closer to Java, instead of using XML based syntax.

- Gradle allows dependency management for packages / libraries