DevOps

Experience with

* Infrastructure Automation Tools , Chef, Puppet, Ansible, SaltStack/Windows Powershell
* Web Languages , Ruby/Python/PHP/Java

1. What is the difference between Agile & DevOps
2. What is the need for DevOps?

# DevOps Tools

Git, Version Control

Jenkins, Continuous Integration

|  |  |
| --- | --- |
| **Usage** | **Tools** |
| Version Control | **GIT**, Subversion, AWS CodeCommit |
| Unit Testing | **JUnit** |
| Code Review | Codacy, Codebeat, Codeclimate, Scrutinizer |
| Static Analysis | **Checkstyle, FindBugs, PMD, FindSecurityBugs, SONAR** |
| Build | **Maven**, Gradle |
| Continuous Integration | **Jenkins**, Travis CI, Atlassian Bamboo |
| Continuous Testing | **Selenium**, TestComplete, TestingWhiz  QTP, Appium, Robot, Cucumber, Nose |
| Continuous Deployment | **Jenkins, GoCD, Codar, AWS Code Deploy** |
| Infrastructure Automation | **Puppet, Chef, Ansible, Salt, Terraform** |
| Continuous Monitoring | **Nagios, Splunk, ELK (Elastic Search, Logstash, Kibana)** |
| Containerization | **Docker, Docker Swarm, Kubernetes Docker Orchestration** |
| Virtualization | OpenStack, VMWare, SoftLayer, Vagrant |
| Performance Testing | **JMeter**, Grinder, Tsung, WebLoad, BlazeMeter, NeoLoad, LoadRunner, Loadster, |
| Security Testing |  |
| Resilience Testing | Hystrix, ChaosMonkey |
| UI Testing |  |

# Devops Course , Edureka

## Overview of DevOps

Why DevOps?

What is DevOps?

DevOps Market Trends

DevOps Engineer Skills

DevOps Delivery Pipeline

DevOps Ecosystem

DevOps Case Studies ,

## Version Control with GIT

What is Version Control?

What is GIT?

Why GIT for your organization?

Install GIT

Common commands in GIT

Working with Remote Repositories

Use Cases of Real time using GIT

Branching and Merging in GIT

GIT Workflows , Branching, Merging, Stashing, Rebasing, Reverting, Resetting

GIT Cheatsheet

## Continuous Integration (using Jenkins)

What is CI?

Why is CI required?

Introduction to Jenkins

Introduction to Maven

Jenkins Management

Adding a slave node to Jenkins

Building a Delivery Pipeline

Pipeline as a Code

Implementation of Jenkins in a Project

Build pipeline of Jobs using Jenkins

Create a pipeline script to deploy an application over tomcat server

## Continuous Testing with Selenium

What is Continuous Testing?

What are popular Continuous Testing Tools?

Characteristics of CT Tools and when to use which one

Introduction to Selenium

Why Selenium?

Selenium , Webdriver

Creating Test cases in Selenium Webdriver (Walts)

What and why X-Path

Handling different controls on WebPage

Framework in Selenium

Selenium Integration with Jenkins

Installing Selenium

Creating Test Cases using Selenium WebDriver

## Containerization with Docker, Ecosystem and Networking

Introduction to Docker Ecosystem

Docker Compose

Docker Swarm

Managing Containers

Running Containers

Introduction to Docker Networking

Network Types

Docker Container Networking

Use Docker Compose to create a Word press site

Start Containers on a Cluster with Docker Swarm

Deploy a Multi-tier application over a cluster

Scale an application

## Containerization using Kubernetes

Containers and Container Orchestration

Introduction to Kubernetes

Revisiting Kubernetes Cluster Architecture

Spinning up a Kubernetes Cluster on Ubuntu VMs

Exploring your Cluster

Understanding YAML

Creating a Deployment in Kubernetes using YAML

Creating a Service in Kubernetes

Installing Kubernetes Dashboard

Deploying an app using Dashboard

Using Rolling Updates in Kubernetes

## Continuous Deployment, Configuration Management with Puppet

Introduction to Puppet

Puppet Installation

Puppet Configuration

Puppet Master and Agent Setup

Node Classification

Puppet Environment

Puppet Classes

Automation and Reporting

Configure and Implement servers using Puppet

## Configuration Management with Ansible

What is Infrastructure as Code

What is Configuration management? What are its advantages?

What are the tools for Config Management?

Introduction to Ansible

Ansible Installation

Configuring Ansible Roles

Write Playbooks

Executing Ad-hoc Commands

## Continuous Monitoring with Nagios

Introduction to Continuous Monitoring

Introduction to Nagios

Installing Nagios

Nagios Plugins (NRPE) and Objects

Nagios Commands and Notification

Monitoring different Servers using Nagios

## Introduction to DevOps on Cloud

Why Cloud?

Introduction to Cloud Computing

Why DevOps on the Cloud

Introduction to AWS

Various AWS Services

DevOps using AWS

### Linux Academy

### Beginner

|  |  |
| --- | --- |
| DevOps Essentials | 4 hrs |
| Git Quick Start | 2 Hrs |
| Jenkins Quick Start | 1.5 Hrs |
| Ansible Quick Start | 2 Hrs |
| Puppet Quick Start | 4 hrs |
| Kubernetes Quick Start | 3 Hrs |
| DevSecOps Essentials | 6 hrs |
| Beginners Guide to Containers and Orchestration | 1 Hr |
| YAML Essentials | 1.5 Hrs |
| Kubernetes Essentials | 4 Hrs |
| Big Data Essentials | 3.5 Hrs |
| Elastic Stack Essentials | 4 hrs |

### Intermediate

|  |  |
| --- | --- |
| Source Control with Git | 5 hrs |
| Elastic Search Deep Dive | 18 Hrs |
| Implementing a Full CI/CD Pipeline | 22 Hrs |
| Kubernetes the Hard Way | 17 Hrs |
| Managing Applications and infra with Terraform + AWS | 17 Hrs |
| Docker Certified Associate Prep Course | 18 hrs |
| Certified Kubernetes Administrator | 5 hrs |
| Docker Deep Dive | 7 hrs |
| Certified Jenkins Engineer | 14 Hrs |

Advanced

Cloud Foundry Certified Developer

Elastic Certification Preparation Course

LPI DevOps Tools Engineer Certification

Certified Kubernetes Administrator

Certified Jenkins Engineer – 14 Hrs

RedHat Certified Specialist in Ansible Automation – 21 Hrs

Advanced: Using Ansible for Configuration Management and Deployments – 22 Hrs

JAVA DevOps

1. IDE
   1. Use a good IDE , Eclipse or Jetbrains; Does JetBrains really Improve productivity? Does it warrant cost?
   2. IDE configuration
   3. IDE Keyboard Shortcuts
   4. IDE Plugins , GIT Integration, Auto-Formatting, Static Analysis (PMD, FindBugs, FindSecBugs, CheckStyle), Unit Testing (Junit), Code Review
2. Code Review
3. Unit Testing (Junit)
4. GIT
   1. Commit Code (Developer Cycle for all GIT Activities)
   2. GIT Triggers for CheckStyle, FindBugs, PMD,
5. Continuous Integration

Simplilearn , Projects

Project 1 Scenario,

A media company wishes to offer a website where users can upload photographs. Captions and titles can be added to the photographs. Thumbnails need to be created from photographs so that multiple images can be displayed on index pages. Customers can order prints of photographs on T-shirts, mugs, and other items. You will need to design, implement, and deploy part of the system.  
   
The objective of this project is to design and implement the business and integration tiers of this project. In the final system, image files and thumbnails will be stored in a file system accessible to the web server.

Goals of the project,

* Set up a MySQL database in a Docker container
* Design a database table structure for the data
* Write an SQL script to create the table and enter some sample data
* Create the table in the database
* Create a Java entity object and map it to the database table
* Create a DAO and its implementation
* Create a Junit test case and write integration tests

The deliverables are,

* SQL script
* Entity object code
* Hibernate configuration file
* DAO and implementation code
* Junit test case

Project 2 Scenario,

A library has a large CD collection which it needs to catalog. It needs to store information in a database and have a web front end. You will need to design, implement and deploy part of the system. The objective of this project is to produce a web application which implements the catalog. The information required for each CD is,

* Title
* Artist(s)
* Year of release
* Number of tracks
* Total playing time

Each CD will also have a list of tracks containing the following information,

* Title
* Author(s)
* Playing time

Goals of the project,  
  
The starting point for this project is the code and instructions for exercise 8.7. Take a copy of the DevOps project from lesson 8 and rename the files from Monitoring to Library. You are only required to produce the web front end. The data manager can generate a hard-coded list of CDs rather than extract them from the database.  
  
The deliverables are,

* The data manager Java file
* The library controller Java file
* The JSP which displays the CD list
* A screenshot of the web page showing the list of CDs

Project 3 Scenario,

There is a requirement to be able to convert an integer into words. For example, 57 needs to be converted into “fifty-seven”.The objective of this project is to develop a method which can convert the numbers 1 through 999 to words. If you have time to extend the range up to 2 billion. The method needs to be developed using Test Driven Development (TDD). A standalone application is also required which reads a number from the keyboard and prints out its value in words or “Invalid number”. The program should exit when the user enters 0.  
   
The application needs to be packaged as an executable jar file using Ant.  
  
 Goals of the project,

* Clone the repository and install Ant.
* Git clone https,//github.com/simplilearn-devops/devops-project-3
* Curl -O http,//apache.mindstudios.com//ant/binaries/apache-ant-1.10.0-bin.zip
* Unzip apache-ant-1.10.0-bin.zip
* Ins apache-ant-1.10.0 ant
* Start the VNC server
* On your local machine connect to the VNC server on port 5901
* Open the Numbers project in Eclipse
* Add the jar files in lib to the classpath
* Develop the numbers to words application using TDD
* Implement the main application to read numbers from the keyboard and print out the values
* On the server navigate to the Numbers project
* Run ant to build the project. The build will fail if the unit tests fail.
* ~/ant/bin/ant dist
* The results of the unit tests are in the report directory which got created
* Run the application and try it out
* java ,j Numbers.jar

The deliverables are,

* The unit test java file
* The number of words Java file
* The main application Java file
* The test results file from the report directory
* A screenshot of the application output after running it with a representative set of numbers

Project 4 Scenario,

You were asked to put some downloadable materials to Amazon S3 so that your company’s customers all around the world can access them.   
A month later, during a massive marketing campaign, the marketing team realized that using S3 is expensive and asked you to find and use a more cost-effective way to distribute the files.

Goals of the Project,

* Create an S3 bucket
* Set its access permissions to allow all anonymous users to download the files from it
* Upload sample files to an S3 bucket
* Create a CloudFront distribution using an S3 bucket as its origin
* Verify that the files are accessible

Project 5 Scenario,

The Oscrop Corporation’s public-facing web app currently runs on an IIS web server at the company’s chosen ISP. Oscrop wants to migrate this web app to Azure. You must test the web app’s functionality by setting up a test Oscrop web app. An internal team will provide you with a test web app to deploy. You must ensure that they can continue to stage changes to the test web app before deploying those changes to the public-facing site.

The operations team at Oscrop currently uses a Microsoft SQL Server database to store back-end data for the web app on the company’s servers. You want to investigate the option of using Azure SQL Database to host this database.

The operations team is interested in monitoring the performance of this database in Azure.

Goals of the Project,

* Create a new web app
* Deploy a web app
* Manage web apps
* Create an Azure SQL Database
* Configure server firewall rules
* Use SQL Server Management Studio
* View database metrics

# DevOps Essentials – Linux Academy

### What is DevOps?

DevOps = Dev (Development) + Ops (Operations)

“DevOps is a Software Engineering **Culture** and **Practice** that aims at unifying Software Development (Dev) and Software Operations (Ops)…

DevOps aims at-

* Shorter Development Cycles
* Increased Deployment Frequency
* More dependable releases
* Close alignment with Business objectives

### DevOps Is…

* DevOps is first a **Culture** of collaboration between developers and operations people
* The culture has given rise to a set of **Practices**
* DevOps is a grassroots movement, by practitioners, for practitioners

### DevOps Is NOT…

* DevOps is NOT Tools, but Tools are essential to success of DevOps (Ex: Jenkins, Ansible, Kubernetes etc.)
* DevOps is NOT a standard. (There are Best Practices but there is no Standard)
* DevOps is NOT a Product (Many companies have products that help in DevOps, but DevOps itself is not a product)
* DevOps is NOT a Job Title (Job Titles arise from DevOps roles like DevOps Engineer, DevOps Architect

### Course Coverage

* DevOps Culture – The **Culture** of collaboration between Development and Operations
* DevOps Practices – The **Practices** that support the goals of DevOps Culture
* DevOps Tools – The **Tools** that help implement DevOps Practices
* DevOps & Cloud – The close relationship between DevOps and Cloud

### History of DevOps

DevOps grew out of a necessity to better perform Agile Software Development

### Agile Software Development

* DevOps grew out of the Agile Software development movement
* Agile seeks to develop software in small, frequent release cycles in order to deliver functionality to customers quickly and quickly respond to changing business goals
* DevOps and Agile go hand-in-hand

### The Timeline of DevOps

**2007:** Agile software development was gaining popularity, but it was suffering from a growing divide between development and operations.

Development was trying to push releases faster while Operations struggled to maintain stability of systems and releases

**2007:** Patrick Debois, an engineer with experience doing both Dev and Ops, was doing testing on a project and became frustrated by the huge divide between Dev and Ops

(As a tester, he was going back and forth between Dev & Ops teams. There were many issues in maintaining stability whilst releasing software fast)

**2008:** Patrick Debois and Andrew Shafer met at Agile2008 Conference in Toronto, Canada. They began to start conversations and seek others interested in bridging the divide between Dev and Ops

**June 23, 2009:** John Allspaw and Paul Hammond gave a talk at Velocity Conference: “10+ Deploys per Day: Dev and Ops Cooperation at Flickr.” Patrick was watching via livestream. People began discussing it via twitter.

**October 30-31 2009:** Patrick hosted the first **DevOpsDays** in Ghent, Belgium; a conference for both devs and ops engineers. The conversation continued on Twitter: #devops

DevOps grew into an organic, grassroots movement all over the world and spawned many tools to support the practices valued by DevOps

### Today

The DevOps movement has not stopped and growing since 2009. It is no longer small, niche movement. It has since,

* Become mainstream
* Spawned a large variety of tools
* Completely changed the IT industry forever

## DevOps Culture

The goals of DevOps talks about the shared goals of Development and Operations.

### The Goals of DevOps

DevOps culture is about **collaboration** between Dev and Ops

Under the traditional separation between Dev and Ops, Dev and Ops have **different** and **opposing** goals

**Development Goal -> Speed**

[Dev wants to multiple releases and frequent code changes and deployments to provide features to the Customer faster]

**Operations Goal -> Stability**

[Operations is concerned about Stability of the system. Frequent changes are the risk to stability of system]

**So, goals of Dev and Ops are in Conflict and opposition.**

Dev teams are measured by number of features delivered to customer. Ops teams are measured by Low Down time.

Dev Teams PRIORITIZE speed of deployment over Stability. Operations teams PRIORITIZE stability over speed and changes. So, neither of them will be able to achieve their goals and will be in fight.

### DevOps Culture

* DevOps culture is about **collaboration** between Dev and Ops
* With DevOps, Dev and Ops work together and share the **same goals**

**Dev and Ops teams are fused and now have a common goal -> Speed AND Stability**

* Both Dev and Ops are in the same group
* They both share the same goal
* They both are measured by the same goal
* They invent tools and practices that PRIORITIZE and Optimize both Speed and Stability
* This allows Development to release quickly, make frequent changes in a way such that System is Robust

### The Goals of DevOps Culture

With DevOps

* Dev and Ops are playing on the same team
* Dev and Ops share the same goals

**Goals**

* Fast time-to-market (TTM)
  + TTM is the time that it takes for what the development has done to reach the hands of the customer.
* Few Production failures
* Immediate recovery from failures (Recover even before Customer notices)

### Summary

* DevOps is about Dev and Ops working together
* In DevOps culture, Dev’s care about stability as well as speed, and Ops care about speed as well as Stability
* The traditional roles of developers and operations engineers can get blurred – Engineers can take more Operations roles and responsibility and vice-versa.
* As in traditional SW development Instead of “Throwing code over the wall”, now there is no wall. Dev and Ops work together to create and use tools and processes that support both Speed and Stability
* DevOps recognizes that dev and ops are more powerful when they are together!

## DevOps Vs Traditional Silos

### Traditional Silos

Developers || QA Team || Operations

* Dev writes Code
* Dev “throw it over the wall” to QA
* Code **BOUNCES BACK AND FORTH** between Dev and QA as QA discovers problems and Dev’s Fix them
* Finally, QA team declares Code ready for production

When QA is testing, Dev is not bothered, and they work on other features. Similarly, when Dev is working, QA are either Idle or are assigned to other projects.

* QA/Dev “throws the code over the wall” to Operations
* Oh No! There’s a problem! (New code could not be deployed or caused a problem in production setup). Ops Throws it back over the wall to Dev
* Each group’s domain is a “Black Box” to the other groups
* “Our Systems are fine; It’s your code”
* “But the code works on my machine!”

### What went wrong?

* Dev and Ops are black boxes to each other, which leads to finger pointing
  + Because Ops is a Black-box, Dev’s don’t really trust them
  + And Ops doesn’t trust Dev
* Dev and Ops have different priorities, which pits them against each other:
  + Ops views Dev as breaking stability
  + Dev see Ops as an obstacle to delivering their code
* Even if they WANT to work together:
  + Dev is measured by delivering features, which means deploying changes
  + Ops is measured by uptime, but changes are bad for stability

### Downsides of Traditional Silos

* “Black boxes” lead to finger pointing
* Lengthy process means slow time-to-market
* Lack of automation means things like builds and deployments are inconsistent
* It take a long time to identify and fix problems

### DevOps Model

**Developers + QA + Operations**

**Build >> Integrate >> Test >> Deploy >> Monitor**

* Dev Writes Code
* When Dev Commits code, it Triggers a series of Automated processes – Automated Build, Automated Integration, Automated Tests
* QA can get their hands on it almost immediately
* Once it is ready, kick off an Automated Deployment to Production

### Advantages

* Since everything is automated, it is much easier to deploy while keeping things stable
* Deployments can occur much more frequently, getting features into the hands of the customers faster

### What happens on failure in Production?

* Oh no! The latest deployment broke something in Production!
* Fortunately, Automated Monitoring notified the team immediately
* The team does a Automated Rollback by deploying the previous working version, fixing the problem quickly
* Quickly, the dev team was able to deploy a fixed version of the new code

### What went right?

* Dev and Ops worked together to build a robust way of changing code quickly and reliably:
  + Both Dev and Ops worked together to prioritize both speed of delivery and stability
* Automation led to consistency
  + Building, testing and deploying happened the same way every time
  + Building, Testing and deploying happened much more quickly and more often
* Good monitoring, plus the swift deployment process, ensured problems could be fixed even before users noticed them
  + Dev and Ops worked together up-front to build good processes
  + Even though a code change caused a problem, users experienced little or no downtime

### Why do DevOps?

* Happier Teams:
  + Tech employees tend to be happier doing DevOps than under traditional silos
  + More time innovating and less time putting out the fires
  + Devs don’t feel like they have to fight to get their work out to customers
  + Operations people don’t have to fight Dev to keep the system stable
* Happier Customers:
  + DevOps lets you give customers the features they want quickly
  + And you don’t have to sacrifice stability to do it!

## Build Automation

### What is Build Automation?

* Build automation is the automation of the process of preparing code for deployment to a live environment
* Depending on what languages are used, code needs to be compiled, linted, minified, transformed, unit-tested etc.
* Build automation means taking these steps and doing them in a consistent, automated way using a script or tool
* The tools of build automation often differ depending on what programming languages and frameworks are used, but they have one thing in common: automation

### What does build automation look like?

* Usually, build automation looks like running a command-line tool that builds code using configuration files and/or scripts that are treated as part of the source code
* Build automation is independent of IDE
* Even if you can build within the IDE, it should be able to work the same way outside the IDE
* As much as possible, build automation should be agnostic of the configuration of the machine it is built on
* Your code should be able to build on someone else’s machine the same way it builds on yours

### Why do build automation?

* Build automation is **fast** – Automation handles tasks that would otherwise need to be done manually
* Build automation is **consistent** – The build happens the same way every time, removing problems and confusion that can happen with manual builds
* Build automation is **repeatable** – The build can be done multiple times with same result. Any version of the source code can always be transformed into deployable code in a consistent way
* Build automation is **portable** – The build can be done the same way on any machine. Anyone on the team can build on their machine, as well as on a shared build server. Building code doesn’t depend on specific people or machines
* Build automation is **reliable** – There will be fewer problems caused by bad builds

## Continuous Integration (CI)

### What is Continuous Integration?

* Continuous Integration is the practice of frequently merging code changes done by developers
* Traditionally, developers would work separately, perhaps for weeks at a time, and then merge all their work together at the end in one large effort
* Continuous Integration means merging constantly throughout the day, usually with the execution of automated tests to detect any problems caused by the merge
* Merging all the time could be a lot of work, so to avoid that, it should be automated!

### What does Continuous Integration look like?

* Continuous Integration is usually done with the help of a CI server
* When a developer commits a code change, the CI server sees the change and automatically performs a build, also executing automated tests
* This occurs multiple times a day
* If there is any problem with the build, the CI server immediately and automatically notifies the developers
* If anyone commits code that “breaks the build”, they are responsible for fixing the problem or rolling back their changes immediately so that other developers can continue working

### Why do Continuous Integration?

* Early detection of certain types of bugs – if the code doesn’t compile or an automated test fails, the developers are notified and can fix it immediately. The sooner these bugs are detected, the easier they are to fix
* Eliminate the scramble to integrate just before a big release – The code is constantly merged, so there is no need to do a big merge at the end
* Makes frequent releases possible – Code is always in a state that can be deployed to production
* Makes continuous testing possible – Since the code can always be run, QA testers can get their hands on it all throughout the development process, not just at the end
* Encourages good coding practices – Frequent commits encourages simple, modular code

## Continuous Delivery and Continuous Deployment

### What is Continuous Delivery?

* Continuous Delivery (CD) is the practice of continuously maintaining code in a deployable state
* Regardless of whether or not the decision is made to deploy, the code is always in a state that is able to be deployed
* Some use the terms Continuous Delivery and Continuous Deployment interchangeably, or simple use the abbreviation CD

### What is Continuous Deployment

* Continuous Deployment is the practice of frequently deploying small code changes to production
* Continuous delivery is keeping the code in a deployable state. Continuous deployment is actually doing the deployment frequently
* Some companies that do continuous deployment, deploy to production multiple times a day
* There is no standard for how often you should deploy, but in general the more often you deploy, the better!
* With Continuous Deployment, deployments to production are routine and commonplace. They are not a big scary event

### What does Continuous Delivery and Continuous Deployment look like?

* Each version of code goes thro’ a series of stages such as automated build, automated testing and manual acceptance testing. The result of this process is an artifact or package that is able to be deployed.
* When the decision is made to deploy, the deployment is automated. What the automated deployment looks like depends on the architecture, but no matter what the architecture is, the deployment is automated
* If a deployment causes a problem, it is quickly and reliably rolled back using an automated process (hopefully even before a customer notices the problem)
* Rollbacks are not a big deal because the developers can redeploy a fixed version as soon as they have one available
* No one grips their desk in fear during a deployment, even if the deployment does cause a problem, since automated rollback is available

### Why do Continuous Delivery and Continuous Deployment?

* **Faster time-to-market**: Get features into the hands of customers more quickly rather than waiting for a lengthy deployment process that doesn’t happen often
* **Fewer problems caused by deployment process**: Since the deployment process is frequently used, any problem with the process are more easily discovered
* **Lower risk**: The more changes are deployed at once, the higher the risk. Frequent deployments of only a few changes are less risky
* **Reliable rollbacks**: Robust automation means rollbacks are a reliable way to ensure stability for customers, and rollbacks don’t hurt developers because they can roll forward with a fix as soon as they have one
* **Fearless deployments**: Robust automation plus the ability to rollback quickly means deployments are commonplace, everyday events rather than big, scary events

## Infrastructure as Code

### What is Infrastructure as Code?

* Infrastructure as Code (IaC) is a way to manage and provision infrastructure through code and automation
* With Infrastructure as code, instead of doing things manually, you use automation and code to create and change:
  + Servers
  + Instances
  + Environments
  + Containers
  + Other Infrastructure

### What does infrastructure as code look like?

* Without IaC you might:
  + SSH in to a host
  + Issue a series of commands to perform a change
* With IaC:
  + Change some code or configuration files that can be used with an automation tool to perform changes
  + Commit them to source code
  + Use an automation tool to enact the changes defined in the code and/or configuration files
* With IaC, provisioning new resources and changing existing ones are both done through automation

### Why do infrastructure as code?

* **Consistency** in creation and management of resources – The same automation will run the same way every time; different humans tend to do things differently
* **Reusability** – Code can be used to make the same change consistently across multiple hosts and can be used again in the future
* **Scalability** – Need a new instance? You can have one configured exactly the same way as the existing instances in minutes (or seconds)
* **Self-documenting** – With IaC, changes to infrastructure document themselves to a degree. The way a server is configured can be viewed in source control, rather than being a matter of who logged in to the server and did something
* **Simply the complexity** – Complex infrastructures can be stood up quickly once they are defined as code. A group of several interdependent servers can be provisioned on demand

## Configuration Management

### What is Configuration Management?

* Configuration Management is maintaining and changing the state of pieces of infrastructure in a consistent, maintainable and stable way
* Changes always need to happen – configuration management is about doing them in a maintainable way
* Configuration management allows you to minimize “**configuration drift**” – the small changes that accumulate over time and make systems different from one another and harder to manage
* Infrastructure as Code is very beneficial for Configuration management

### What does Configuration management look like?

* You need to upgrade a software package on a bunch of servers
  + Without good configuration management, you log in to each server and perform the upgrade. However, this can lead to lot of problems. Perhaps one server was missed due to poor documentation, or perhaps something doesn’t work while the versions are temporarily mismatched between servers, causing a lot of downtime while you do the upgrade
  + With good configuration management, you define the new version of the software package in a configuration file or tool and automatically roll out the change to all of the servers
* Configuration management is about managing your configuration somewhere outside of the servers themselves

### Why do Configuration management?

* **Save Time** – It takes less time to change the configuration
* **Insight** – With good configuration management, you can know about the state of all the pieces of a large and complex infrastructure
* **Maintainability** – A more maintainable infrastructure is easier to change in a stable way
* **Less configuration drift** – It is easier to keep a standard configuration across a multitude of hosts

## Orchestration

### What is Orchestration?

* **Orchestration** is automation that supports processes and workflows, such as provisioning resources
* With orchestration, managing a complex infrastructure is **less like being a builder** and more like **conducting an orchestra**
* Instead of going out and creating a piece of infrastructure, the conductor simply signals what needs to be done and the orchestra performs it
  + The conductor does not need to control every detail
  + The musicians (automation) are able to perform their piece with only a little bit of guidance

### What does Orchestration look like?

Here’s an example:

* A customer requests more resources for a web service that is about to see a heavy increase in usage due to a planned marketing effort
* Instead of manually standing up new nodes, operations engineers use an orchestration tool to request five more nodes to support the service
* A few minutes later, the tool has five new nodes up and running

A much Cooler Example:

* A monitoring tool detects an increased load on the service
* An orchestration tool responds to this by spinning up additional resources to handle the load
* When the load decreases again, the tool spins the additional resources back down, freeing them up to be used by something else
* All of this happens while the engineer is getting coffee

### Why do orchestration?

* **Scalability**: Resources can be quickly increased or decreased to meet changing needs
* **Stability**: Automation tools can automatically respond to fix problems before users see them
* **Save Time**: Certain tasks and workflows can be automated, freeing up engineer’s time
* **Self-service**: Orchestration can be used to offer resources to customers in a self-service fashion
* **Granular insight in to resource usage**: Orchestration tools give greater insight in to how many resources are being used by what software, services or customers.

## Monitoring

### What is Monitoring?

* Monitoring is the Collection and Presentation of data about the Performance and Stability of services and infrastructure
* Monitoring tools collect data over things such as:
  + Usage of Memory
  + CPU
  + Disk I/O
  + Other resources over time
  + Application Logs
  + Network Traffic
* The collected data is presented in various forms, such as charts and graphs, or in the form of real-time notifications about problems

### What does Monitoring look like?

Real-time Notifications:

* Performance of website is beginning to slow down
* A Monitoring tool detects that response times are growing
* An administrator is immediately notified and is able to intervene before downtime occurs

Postmortem Analysis:

* Something went wrong in production last night
* It’s working now, but we don’t know what caused it
* Luckily, monitoring tools collected a lot of data during the outage
* With that data, developers and operations engineers are able to determine the root cause (a poorly performing SQL query) and fix it

### Why do Monitoring?

* **Fast Recovery** – The sooner a problem is detected, the sooner it can be fixed. You want to know about a problem before your customer does!
* **Better root cause analysis** – The more data you have, the easier it is to determine the root cause of the problem
* **Visibility across teams** – Good monitoring tools give useful data to both developers and operations people about the performance of code in production
* **Automated Response** – Monitoring data can be used alongside orchestration to provide automated responses to events such as automated recovery from failures

## Microservices

### What are Microservices?

* Microservice architecture breaks up an application in to a collection of small, loosely-coupled services
* Traditionally, apps used a monolithic architecture. In a monolithic architecture, all features and services are part of one large application
* Microservices are small; each microservice implements only a small piece of an application’s overall functionality
* Microservices are loosely coupled; Different microservices interact with each other using stable and well-defined APIs. This means they are independent of one another

### Microservices Vs Monolith

Example of Monolith:

Single app handling – Authentication, Customer Info, Inventory, Payments

Example of Microservices:

UI interacting with following Services – Authentication Service, Customer Info service, Inventory Service and Payments Service

The services-

* Need not be hosted in the same place
* Need not be implemented using same technology

### What do microservices look like?

There are many different ways to structure and organize a microservice architecture.

Example: A Pet Shop application might have

* A Pet Inventory service
* A Customer Details service
* An Authentication service - Login
* A pet adoption request service – Form filling
* A payment processing service – Payments for goods/service

Each of these is in its own codebase and separate running process (or processes). They can all be built, tested, deployed and scaled independently

### Why use Microservices?

* **Modularity** – Microservices encourage modularity; In monolithic apps, individual pieces become tightly coupled and complexity grows. Eventually it is very hard to change anything without breaking something
* **Technological flexibility** – You don’t need to use the same languages and technologies for every part of the app. You can use the best tool for each job
* **Optimized scalability** – You can scale individual parts of the application based upon resource usage and load. With a monolith, you have to scale up the entire application, even if only one aspect of the service actually needs to be scaled

Note: Microservices are not always the best choice. For smaller, simpler apps, a monolith might be easier to manage.

## Introduction to DevOps Tools

### The Role of Tools in DevOps

* DevOps is NOT a set of Tools
* But how can we achieve high speed of delivery while maintaining stability? **Using TOOLS!**
* The DevOps community has created a wide range of powerful tools
* Part of doing DevOps is identifying the tools you need and learning how to use them

### The Periodic Table of DevOps Tools

<https://xebialabs.com/periodic-table-of-devops-tools/>

## Build Automation Tools

* Build Automation is Automated Processing of code in preparation for deployment
* What tools you use for build automation usually depends on programming languages and frameworks used

A few examples

* Java – ant, **maven**, **gradle**
* Javascript – **npm**, **grunt**, **gulp**
* Make – widely used in Unix-based systems
* Packer – builds machine images and containers

## Continuous Integration Tools

* Continuous Integration is the process of continuously merging code in to a single branch or mainline
* Continuous Integration tools usually consist of a server that integrates with source control
* When source code is changed, the server responds by executing an automated build

### Jenkins

* Open source fork of Hudson (controlled by Oracle)
* Widely used
* Java Servlet-based
* Provides a Web-UI where one can configure all details

### Travis CI

* Open Source
* Built around GitHub Integration (If you use GitHub, the integration is super-smooth)
* Executes builds in clean VMs (Useful since there is no corruption in build system/server)

### Bamboo

* Enterprise product by Atlassian
* Great Out-of-the-box integration with other Atlassian products like JIRA and Confluence

## Tools for Configuration Management

* Configuration Management – Managing and changing the state of pieces of infrastructure in a consistent and maintainable way
* Configuration Management tools are a great way to implement infrastructure as code

### Ansible

* Open Source Tool
* Declarative Configuration – Instead of specifying the steps of the configuration, we specify the target or final state we want the infrastructure in and the tool determines how to reach there
* Uses YAML
* No control server needed – but Ansible tower is available.  
  Most Configuration Mgmt tools need a control server to push changes to the host environment. One can do this from a laptop or any machine
* Ansible does not need Agents, just python and ssh  
  Most configuration mgmt. tools need agents to be installed on the hosts to execute the actions on the host. In case of ansible, no agents are required. It manages to push changes via ssh

### Puppet

* Declarative Configuration
* Manage state through a UI
* Can create Custom Modules using Puppet DSL (Domain Specific Language)
* Pushes changes to clients using a control server and agents installed on client

### Chef

* Procedural configurations (Write steps/scripts)
* Agent/server model
* Uses Chef DSL

### Salt

* Made by Salt Stack
* Declarative configuration
* Agent(minions)/server(master) – but can support agentless
* Uses YAML
* Support for event-driven automation – Other tools depend on human or schedule interventions. Here events on host can trigger automation

## Tools for Virtualization and Containerization

### Virtualization Tools

* Virtualization – Managing resources by creating virtual rather than physical machines
* Hypervisor – Runs on bare metal and manages virtual machines (VMs)
* Examples:
  + VMWare ESX and ESXi
  + Microsoft Hyper-V
  + Citrix XenServer

### Containerization

* Containers – Lightweight, isolated packages containing everything needed to run a piece of software
* Require fewer resources than VMs – VMs contain and entire OS plus virtual extensions of all the hardware
* Containers have the bare minimum needed to run the software
* Docker – Docker is currently the leading container technology
* Containers are still relatively new but very useful for DevOps! Orchestration is a important DevOps practice and containers are very helpful for this.

Orchestration & Containerization go hand in hand

Another important use of Containerization is Portability – Developers can run a container similar to Production system on their Laptops to get more insight in to its behavior

## Monitoring Tools

Monitoring – Collecting and presenting data about the state and performance of applications

There are different types of monitoring:

* Infrastructure monitoring – focuses on things related to infrastructure  
  Ex: CPU, RAM etc.
* Application Performance Monitoring (APM) – focuses on performance and stability of individual parts of an application  
  Ex: Response Time, Logs etc.

### Infrastructure Monitoring Tools

### SenSu

* Designed as a modern replacement for Nagios
* Server/agent architecture
* Agents push data to an AMQP broker
* Much more scalable and advanced than Nagios

### NewRelic

* SaaS + Agent (their central server is on the cloud)
* Wide variety of metrics (also does APM well)

### Application Performance Monitoring Tools

### AppDynamics

* Collects data Points about applications and presents it in a centralized dashboard
* Code-level diagnostics: also able to identify performance issues at code level
* Server/agent model

### NewRelic

* Does APM well

### Aggregation and Analytics Tools

* Aggregation and Analytics are about collecting, monitoring data and doing something with it
* Most Monitoring tools have some aggregation and analytics features
* Elastic Stack – pump data in and quickly create views to aggregate data and easily detect and diagnose problems

## Tools for Orchestration

* Orchestration is automation that supports processes and workflows, such as provisioning resources
* Lets you do things like-
  + Scale-Up and Scale-Down applications on request
  + Auto-Scale applications based on usage
  + Create Self-Healing systems by spinning down unhealthy nodes and replacing them with new ones

### Docker Swarm:

* Docker-native
* Orchestration of Docker containers

### Kubernetes

* Open Source
* Orchestration Server
* Manage containerized apps across multiple hosts

### Zookeeper

* Open source – Apache foundation
* Can work alongside Kubernetes
* Offers a centralized service registry that integrates with orchestration features

### Terraform

* Combines orchestration and infrastructure-as-code
* Works well with tools like Ansible
* Works well with AWS
* Integrates with Kubernetes

## DevOps and the Cloud

* DevOps and the cloud are not the same thing:
  + DevOps is a culture of collaboration between Dev and Ops
  + The Cloud – remote servers on the internet that offer services in place of locally-hosted solutions
* DevOps culture and practices are very useful in the world of the cloud
* DevOps and the cloud developed alongside one another, and many cloud services are built on DevOps practices
* They can also be a tool for DevOps. Many cloud services offer features that support DevOps practices

### Traditional Non-Cloud Stack

* A traditional stack is a regular, self-hosted datacenter
* In a traditional stack, you are responsible for every layer of the architecture
* You provide all of the infra necessary to run your apps

### Infrastructure-as-a-Service

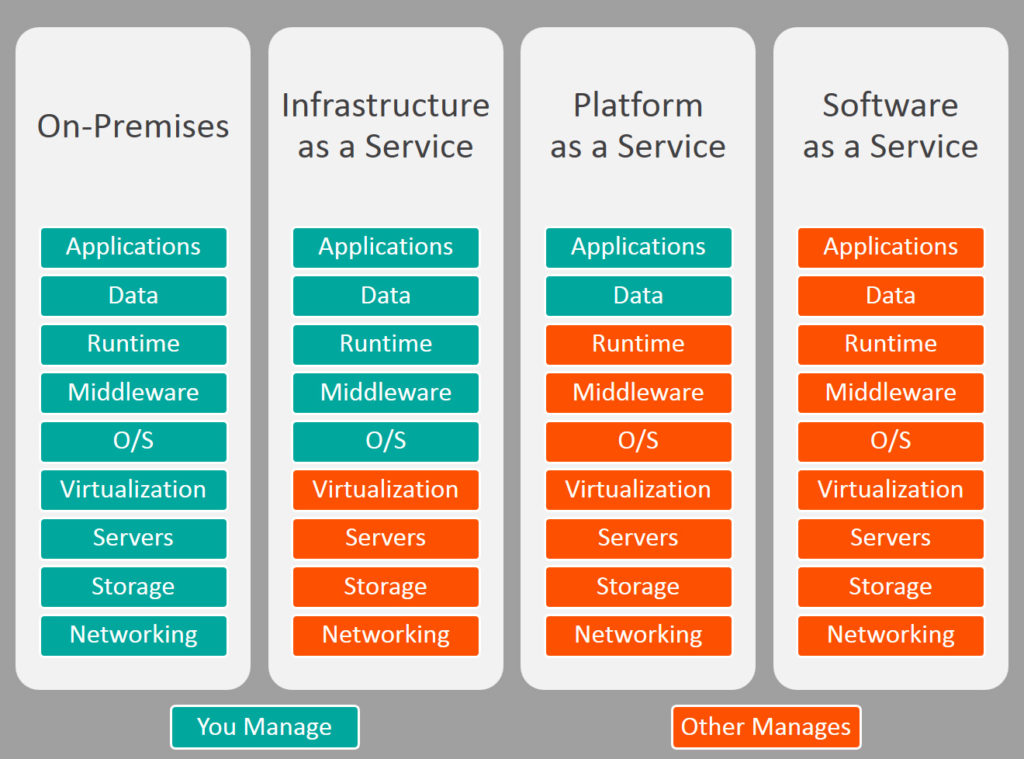
* With IaaS, someone else provides the low-level Infrastructure (Network, Compute, Storage, Virtualization)
* The cloud provider gives you a bare OS
* You are responsible for installation and configuration above OS level
* Examples – Amazon EC2 Instances, Microsoft Azure VMs and containers, Google Compute Engine

### Platform-as-a-Service

* With PaaS, everything below the Application and Data Layers is abstracted
* The cloud service provider gives you a way to deploy and app use databases
* You are only responsible to manage the app and data
* Examples: AWS Elastic Beanstalk, Heroku, Google App Engine

### Software-as-a-Service

* With SaaS, everything is managed
* The cloud service provider gives you an application ready for use
* You are only responsible for using the application
* Examples: G-Mail, Microsoft Office 365



### Serverless

* Serverless is also known as Function-as-a-Service (FaaS)
* Serverless is different from traditional application architecture
* Everything is abstracted. You deploy small, single-purpose functions
* You pay for the compute resources used by your functions
* Examples: AWS Lambda, Azure Functions, Google Cloud Functions

## DevOps and Google Cloud Platform

### Google App Engine:

* **PaaS** – Deploy your code, don’t worry about the rest
* Built-in support for **microservices**
* Out-of-the-box **autoscaling**
* Certain configurations can be considered **serverless**

### Google Compute Engine

* IaaS – Deploy and orchestrate clusters of VMs on Google’s architecture
* Built-in orchestration
* Works with Google App Engine
* Can be managed with other tools like Ansible, Salt, Puppet and Chef

### Google Cloud Functions:

* Google’s FaaS/Serverless solution
* Quickly and easily create and deploy FaaS functions

### Google Cloud SDK

* An SDK (Software Development Kit) for interacting with GCP APIs
* Makes it easy to build your own tools and automations that interact with GCP

### Stackdriver

* GCP’s **monitoring** solution
* Monitoring, logging and diagnostics for your GCP services
* Also works with AWS

### Cloud Deployment Manager

* Declarative configuration for your GCP stack
* IaaC and automated deployment
* YAML-based

### Google Kubernetes Engine

* Orchestration on GCP with Kubernetes
* Do continuous integration with Jenkins on Kubernetes Engine

## Microsoft Azure DevOps Features

### Continuous Integration, Delivery and Deployment:

* Visual Studio Team Services – source control and CI
* Jenkins – CI for Java apps
* Continuous Deployment Triggers – automated deployment triggers integrated with CI

### Orchestration

* Azure Container Registry – repository of container images
* Azure Container Service – Kubernetes Orchestration
* Azure Web Apps – Cloud hosting for web apps integrated with DevOps pipeline

### Monitoring

**Azure Application Insights** – APM, diagnostics and analytics. Support Machine Learning of Application logs so that we come to know of issues much faster

### FaaS/Serverless

* Azure Functions – autoscaling, serverless functions in Azure

## Amazon Web Services DevOps Features

### EC2:

* IaaS
* Easily Scalable
* Full control over your cloud infra
* Integrates tons of tools, both AWS and 3rd Parry

### Elastic Beanstalk

* PaaS
* Out-of-the-box load balancing and autoscaling
* Can still access underlying AWS resources with full control

### Continuous Integration, Delivery, Deployment

* AWS CodeBuild – continuous Integration
* AWS CodeDeploy – continuous Deployment
* AWS CodePipeline – full code pipeline from build to deploy
* AWS CodeStar – integrates all parts of the process with Project Management tools and JIRA issue tracking

### Infrastructure-As-Code

* CloudFormation – Stack templating engine, YAML or JSON based
* OpsWorks – IaC with Chef

### Serverless/FaaS

* AWS Lambda – run serverless on AWS

### Monitoring

* CloudWatch – track metrics and logs, set alarms, automate responses to monitoring data

## Hands-On Lab

### CLOUD SERVER – Production Webserver

UN: cloud\_user PW: jlwwlqRBxL

Public IP Address of Production Web Server: 34.204.78.222

Private IP Address of Production Web Server: 10.0.1.125

### CLOUD SERVER – CI Server

UN: cloud\_user PW: jlwwlqRBxL

Public IP, Private IP: 3.93.14.157, 10.0.1.95

### CLOUD SERVER – Staging Webserver

UN: cloud\_user PW: jlwwlqRBxL

Public IP, Private IP: 3.90.39.166, 10.0.1.252

## Version Control

* Build version numbering

## Continuous Integration

* Build/compile
* Linting (Validation & Code Review) – Static Analysis, Security Analysis
* Unit Testing
* Integration Testing

﻿

Poulton, Nigel. Docker Deep Dive (p. 92). leanpub.com. Kindle Edition.