Contents

[SRE (Site Reliability Engineering) 14](#_Toc35495158)

[DevOps Classroom Series- SRE – 22/Oct/2019 14](#_Toc35495159)

[Site Reliability Engineering – Intro 14](#_Toc35495160)

[How SRE implements DevOps 15](#_Toc35495161)

[DevOps Classroom Series – 23/Oct/2019 16](#_Toc35495162)

[Primary Reference 16](#_Toc35495163)

[SLI (Service Level Indicator) 16](#_Toc35495164)

[SLO (Service Level Objective) 16](#_Toc35495165)

[SLA (Service Level Agreements) 16](#_Toc35495166)

[Problem 17](#_Toc35495167)

[Risks 17](#_Toc35495168)

[Error Budget 17](#_Toc35495169)

[Error Budget Burndown 17](#_Toc35495170)

[Toil 17](#_Toc35495171)

[DevOps Classroom Series – SRE – 24/Oct/2019 17](#_Toc35495172)

[Monitoring and Alerting 18](#_Toc35495173)

[Observability 18](#_Toc35495174)

[Incident Mangement 18](#_Toc35495175)

[DevOps Classroom Series – 25/Oct/2019 18](#_Toc35495176)

[Metrics to measure Reliability 18](#_Toc35495177)

[Post-Mortem 19](#_Toc35495178)

[Ansible and Chef 19](#_Toc35495179)

[Docker, Kubernetes 19](#_Toc35495180)

[Monitoring 19](#_Toc35495181)

[DevOps Classroom Series – 27/Oct/2019 19](#_Toc35495182)

[System Monitoring 19](#_Toc35495183)

[Application Monitoring 20](#_Toc35495184)

[Terms 20](#_Toc35495185)

[Expectations 20](#_Toc35495186)

[SRE Expectations 20](#_Toc35495187)

[NAGIOS 20](#_Toc35495188)

[DevOps Classroom Series 29/Oct/2019 21](#_Toc35495189)

[System Monitoring using Nagios 21](#_Toc35495190)

[Terms 22](#_Toc35495191)

[Different Options of Nagios 22](#_Toc35495192)

[Status 22](#_Toc35495193)

[DevOps Classroom Notes 30 Oct 2019 – Elastic Stack Usecase 24](#_Toc35495194)

[Scenario 24](#_Toc35495195)

[Application-1 Architecture 25](#_Toc35495196)

[Enabling Log Monitoring 25](#_Toc35495197)

[Elastic Stack 26](#_Toc35495198)

[Elastic Search 26](#_Toc35495199)

[LogStash 26](#_Toc35495200)

[Kibana 26](#_Toc35495201)

[Beats 26](#_Toc35495202)

[Elastic Stack – Introduction and Core Concepts 27](#_Toc35495203)

[Elastic Stack 27](#_Toc35495204)

[Elastic Stack Components 27](#_Toc35495205)

[Elastic Search 28](#_Toc35495206)

[Kibana 31](#_Toc35495207)

[Core Concepts of Elastic Search 34](#_Toc35495208)

[Elastic Stack and Kibana Loading Data and API 35](#_Toc35495209)

[Mappings and Data types 36](#_Toc35495210)

[Index API 36](#_Toc35495211)

[**DevOps Classroom Series 01/Nov/2019** 41](#_Toc35495212)

[When to create indexes in Elastic Search Monitoring Use case 41](#_Toc35495213)

[To Create Indexes We need Index APIs 41](#_Toc35495214)

[Visualize in Kibana 41](#_Toc35495215)

[Beats 41](#_Toc35495216)

[**DevOps Classroom Series – 3/Nov/2019** 41](#_Toc35495217)

[Install Logstash on Centos7 using rpm 41](#_Toc35495218)

[Golden Signals 42](#_Toc35495219)

[Reason 42](#_Toc35495220)

[**DevOps Classroom Series- Ansible 05/November/2019** 44](#_Toc35495221)

[**Configuration Management** 44](#_Toc35495222)

[**Manual** 44](#_Toc35495223)

[**Shell/Powershell Scripts** 44](#_Toc35495224)

[**Configuration Management** 44](#_Toc35495225)

[**Architecture of Configuration Management** 45](#_Toc35495226)

[**PUSH** 45](#_Toc35495227)

[**PULL** 45](#_Toc35495228)

[**Ansible Falls under Push based Configuration Management** 46](#_Toc35495229)

[**What do the user need to provide to ansible** 46](#_Toc35495230)

[DevOps Classroom Series – Ansible Installation – 06/Nov/2019 47](#_Toc35495231)

[Ansible Installation 47](#_Toc35495232)

[Lab Setup 47](#_Toc35495233)

[Refer the following 48](#_Toc35495234)

[DevOps Classroom Series – 7/Nov/19 48](#_Toc35495235)

[How to think in Ansible 48](#_Toc35495236)

[DevOps Classroom Series – Ansible – 8/Nov/2019 48](#_Toc35495237)

[Ansible Workflow 48](#_Toc35495238)

[Sample-Application 48](#_Toc35495239)

[Spring-pet Clinic 49](#_Toc35495240)

[Student Courses Application 49](#_Toc35495241)

[Using Ansible For deployment of Sample Application 49](#_Toc35495242)

[YAML 50](#_Toc35495243)

[DevOps Classroom Series 10/Nov/2019 – Ansible 50](#_Toc35495244)

[How to automate application deployments using Ansible 50](#_Toc35495245)

[Playbooks Structure 50](#_Toc35495246)

[Simple Scenario: Installing Tree on all the nodes 51](#_Toc35495247)

[Module Categories In Ansible 52](#_Toc35495248)

[Execution in Ansible 52](#_Toc35495249)

[Adhoc Commands in Ansible 52](#_Toc35495250)

[Playbook in Ansible 52](#_Toc35495251)

[DevOps Classroom Series – Ansible – 11/Nov/2019 53](#_Toc35495252)

[Widely Used Modules 53](#_Toc35495253)

[Sample Deployment in Ansible 53](#_Toc35495254)

[DevOps Classroom Series 12/Nov/2019 57](#_Toc35495255)

[Ansible Playbook for lamp (Contd..) 57](#_Toc35495256)

[Script from Previous Series 57](#_Toc35495257)

[Goal is have one playbook for Redhat and Ubuntu 58](#_Toc35495258)

[Ansible Facts 59](#_Toc35495259)

[Ansible Conditionals 59](#_Toc35495260)

[Continuing with the Playbook combining facts and when statement 59](#_Toc35495261)

[Example of git installation 60](#_Toc35495262)

[Goal: To make the Playbook Generic 61](#_Toc35495263)

[DevOps Classroom Series -13/Nov/2019 61](#_Toc35495264)

[**Goal: Write Common Tasks and Handlers** 61](#_Toc35495265)

[DevOps Classroom notes – 15/Nov/2019 65](#_Toc35495266)

[**Ansible Variable Files** 65](#_Toc35495267)

[Ansible Variable Precedence 67](#_Toc35495268)

[Ansible Special Variables 67](#_Toc35495269)

[https://docs.ansible.com/ansible/latest/user\_guide/intro\_inventory.html#connecting-to-hosts-behavioral-inventory-parameters 67](#_Toc35495270)

[Adding a custom fail message for unsupported OS 67](#_Toc35495271)

[DevOps Classroom Series – 16/Nov/2019 68](#_Toc35495272)

[Ansible Files 68](#_Toc35495273)

[Exercise 69](#_Toc35495274)

[Solution to above Exercise 70](#_Toc35495275)

[Pre-Tasks and Post-Tasks 70](#_Toc35495276)

[How to Reuse Playbooks 70](#_Toc35495277)

[Ansible roles 71](#_Toc35495278)

[Creating Ansible Roles 71](#_Toc35495279)

[DevOps Classroom Series – 17/Nov/2019 72](#_Toc35495280)

[Ansible Parallelism 72](#_Toc35495281)

[Ansible custom facts 72](#_Toc35495282)

[How to make non idempotent modules idempotent 72](#_Toc35495283)

[Register Variables 73](#_Toc35495284)

[Vault 73](#_Toc35495285)

[Ansible Tower 73](#_Toc35495286)

[Dynamic Inventory 73](#_Toc35495287)

[DevOps Scenarios in Ansible 74](#_Toc35495288)

[Realizing this with CI/CD 75](#_Toc35495289)

[Realizing this with CI/CD and Infra Provisioning 75](#_Toc35495290)

[Exercise 76](#_Toc35495291)

[DevOps Classroom Series – Chef – 19/Nov/2019 78](#_Toc35495292)

[**Configuration Management** 78](#_Toc35495293)

[**Example Scenario** 78](#_Toc35495294)

[**Creating Environments** 78](#_Toc35495295)

[**Sample Java Based application** 78](#_Toc35495296)

[**Configuration Management Architecture** 79](#_Toc35495297)

[**PULL Based Configuration Management.** 79](#_Toc35495298)

[DevOps Classroom Series – Chef Architecture – 20/Nov/2019 80](#_Toc35495299)

[Chef Architecture and Components 80](#_Toc35495300)

[Chef Terminology 81](#_Toc35495301)

[Lab Setup 82](#_Toc35495302)

[DevOps Classroom Series – 21/Nov/2019 82](#_Toc35495303)

[Bootstrap Preparation 82](#_Toc35495304)

[Bootstrap Process 83](#_Toc35495305)

[DevOps Classroom Series – Chef – 22/Nov/19 84](#_Toc35495306)

[Visual Studio Code Configuration 84](#_Toc35495307)

[Automating Deployments using Chef 85](#_Toc35495308)

[Chefs Basic Workflow for cookbook automation 85](#_Toc35495309)

[Scenario-1: Install Git and tree 85](#_Toc35495310)

[DevOps Classroom Series – Chef – 24/Nov/2019 87](#_Toc35495311)

[Uploading Cookbooks to chef server 87](#_Toc35495312)

[Run List 87](#_Toc35495313)

[Developing a Cookbook for Java Application Deployment. 88](#_Toc35495314)

[Developing a cookbook for lamp stack 89](#_Toc35495315)

[DevOps Classroom Series Chef – 25/Nov/2019 93](#_Toc35495316)

[Lamp on Centos 7 93](#_Toc35495317)

[DevOps Classroom Series 28/Nov/2019 94](#_Toc35495318)

[Problems with Current Approach of Developing Cookbooks 94](#_Toc35495319)

[Test Kitchen 94](#_Toc35495320)

[Test Kitchen Setup for AWS Test Environments 95](#_Toc35495321)

[DevOps Classroom 29/Nov/2019 95](#_Toc35495322)

[Kitchen Docs 95](#_Toc35495323)

[Test Kitchen on Azure 95](#_Toc35495324)

[Accessing Node Object in Recipes 95](#_Toc35495325)

[DevOps Classroom Series – 01/Dec/2019 96](#_Toc35495326)

[Attributes 96](#_Toc35495327)

[Scenario: Lamp 96](#_Toc35495328)

[Making Cookbooks Reusable 98](#_Toc35495329)

[Chef Supermarket 98](#_Toc35495330)

[DevOps Classroom Series – 02/Dec/2019 99](#_Toc35495331)

[Chef-Client Supermarket cookbook 99](#_Toc35495332)

[Changing default convergance of a node to 2 hours. 99](#_Toc35495333)

[Scenario: 100](#_Toc35495334)

[DevOps Classroom Series – 03/Dec/2019 101](#_Toc35495335)

[Scenario 101](#_Toc35495336)

[Sample with Environments and Roles 102](#_Toc35495337)

[DevOps Classroom Series – 04/Dec/2019 103](#_Toc35495338)

[Environment File 103](#_Toc35495339)

[Attribute Precedence: 103](#_Toc35495340)

[Chef Templates 104](#_Toc35495341)

[DevOps Classroom Series – 05/Dec/2019 105](#_Toc35495342)

[Chef data bags 105](#_Toc35495343)

[Chef Unattended bootstrap 106](#_Toc35495344)

[Chef Server Installation 106](#_Toc35495345)

[Exercise Spring Boot application (Spring-petclinic) as a Service 106](#_Toc35495346)

[DevOps Classroom Series – 07/Dec/2019 108](#_Toc35495347)

[Chef Server Installation 108](#_Toc35495348)

[Chef Points for Professional Summary & Roles 109](#_Toc35495349)

[DevOps Classroom Series – 10/Dec/2019 111](#_Toc35495350)

[**Sample Pipeline** 111](#_Toc35495351)

[**Requirements for system to store code** 111](#_Toc35495352)

[**Architectures of VCS** 111](#_Toc35495353)

[DevOps Classroom Series – Git – 11/Dec/2019 113](#_Toc35495354)

[Installing Git 113](#_Toc35495355)

[Git Operations on Node (Local) 113](#_Toc35495356)

[Popular Cheatsheets 115](#_Toc35495357)

[DevOps Classroom Series – Git – 12/Dec/2019 115](#_Toc35495358)

[Git Operations 115](#_Toc35495359)

[Untracked file and modified file 115](#_Toc35495360)

[Handling multiple Changes 116](#_Toc35495361)

[Removing files from local repo. 117](#_Toc35495362)

[DevOps Classroom Series – 13/Dec/2019 118](#_Toc35495363)

[**Git History** 118](#_Toc35495364)

[**Revert the modified changes in the Working Tree** 119](#_Toc35495365)

[**Git Branches** 119](#_Toc35495366)

[DevOps Classroom Series – 16/Dec – Installing Jenkins 122](#_Toc35495367)

[Jenkins 122](#_Toc35495368)

[Important considerations for Working with Jenkins 122](#_Toc35495369)

[Installing Jenkins 122](#_Toc35495370)

[DevOps Classroom Series 17/Dec/2019 123](#_Toc35495371)

[**Jenkins Terms** 123](#_Toc35495372)

[**Simulating any Jenkins Job Execution** 123](#_Toc35495373)

[**Exercise** 123](#_Toc35495374)

[DevOps Classroom Series – 18/Dec/2019 Jenkins Configuration 124](#_Toc35495375)

[Configuring Jenkins 124](#_Toc35495376)

[Add Sudo access to Jenkins with NO Password 124](#_Toc35495377)

[Using git with jenkins 124](#_Toc35495378)

[Jenkins Projects 124](#_Toc35495379)

[Continuous Integration 126](#_Toc35495380)

[Continuous Delivery 126](#_Toc35495381)

[DevOps Classroom notes – 19/Dec/2019 127](#_Toc35495382)

[**Building Code and Unit Testing Using Jenkins** 127](#_Toc35495383)

[**Example: Building Docker Image** 127](#_Toc35495384)

[**Build/Compile/test Java Using Maven.** 127](#_Toc35495385)

[DevOps Classroom Series – Maven – 20/Dec/2019 129](#_Toc35495386)

[**Maven** 129](#_Toc35495387)

[**Installation** 129](#_Toc35495388)

[**Maven folder conventions** 129](#_Toc35495389)

[**pom.xml** 129](#_Toc35495390)

[**Goals** 129](#_Toc35495391)

[**Maven Repository Architecture** 130](#_Toc35495392)

[**Using Maven with Jenkins** 130](#_Toc35495393)

[DevOps Classroom Series – 22/Dec/2019 – Maven With Jenkins 131](#_Toc35495394)

[**Testing Types** 131](#_Toc35495395)

[**What is Measured From Tests** 131](#_Toc35495396)

[**Integrating Maven tests with Jenkins** 131](#_Toc35495397)

[**Archiving the Artefacts (Displaying the package built)** 132](#_Toc35495398)

[**Post Build Actions:** 132](#_Toc35495399)

[**Jenkins Plugins** 133](#_Toc35495400)

[**Day Builds** 133](#_Toc35495401)

[**Night Builds** 133](#_Toc35495402)

[DevOps Classroom Series – 22/Dec/2019 – Jenkins Distributed Build 134](#_Toc35495403)

[**Jenkins Build Triggers** 134](#_Toc35495404)

[**Build Periodically** 134](#_Toc35495405)

[**Poll SCM** 135](#_Toc35495406)

[**Build Environment** 135](#_Toc35495407)

[**Jenkins Logical Architecture** 135](#_Toc35495408)

[**Queue** 135](#_Toc35495409)

[**Executor** 135](#_Toc35495410)

[**Jenkins Distributed Builds** 135](#_Toc35495411)

[DevOps Classroom Series – Jenkins – 23/Dec/2019 136](#_Toc35495412)

[**Configuring Linux Nodes to Jenkins Master** 136](#_Toc35495413)

[**Adding Ubuntu Node to Jenkins Master** 136](#_Toc35495414)

[**Exercise-1** 139](#_Toc35495415)

[**Adding Downstream Jobs to Jenkins Job** 139](#_Toc35495416)

[DevOps Classroom Series – Jenkins Windows Node – 24/Dec/2019 140](#_Toc35495417)

[**Adding Windows Node to Jenkins Master** 140](#_Toc35495418)

[**Build Parameters** 141](#_Toc35495419)

[DevOps Classroom Series – Jenkins Pipeline – 25/Dec/2019 143](#_Toc35495420)

[Need For Pipelines 143](#_Toc35495421)

[Jenkins Pipeline 143](#_Toc35495422)

[DevOps Classroom Series – 26/Dec/2019 – Jenkins Pipeline 146](#_Toc35495423)

[Jenkins Pipeline Syntax 146](#_Toc35495424)

[Jenkins Pipeline – Important Definitions 146](#_Toc35495425)

[Quick Wins for generating pipelines. 147](#_Toc35495426)

[DevOps Classroom Series – 29/Dec/2019 149](#_Toc35495427)

[Scripted Pipeline 149](#_Toc35495428)

[Declarative Pipeline 149](#_Toc35495429)

[Configuring Tools to Jenkins 150](#_Toc35495430)

[Multi Branch Pipeline 150](#_Toc35495431)

[Writing Jenkins Declarative Pipelines 151](#_Toc35495432)

[DevOps Classroom Series – 29/Dec/2019 – Integrating Jenkins with Artifactory, Sonar Qube 153](#_Toc35495433)

[Integrating Jenkins with Sonar Qube 153](#_Toc35495434)

[Integrating Jenkins with Artifactory 153](#_Toc35495435)

[Integrating Jenkins with DevOps Tools like Ansible or Docker 153](#_Toc35495436)

[DevOps Classroom Series – Infra provisioning 31/Dec/2019 156](#_Toc35495437)

[Infra-Provisioning 156](#_Toc35495438)

[Packer 156](#_Toc35495439)

[Terraform 156](#_Toc35495440)

[Pre-requisites 157](#_Toc35495441)

[DevOps Classroom Series – 02/Jan/2020 – Packer 157](#_Toc35495442)

[Where Packer Fits in CI/CD 157](#_Toc35495443)

[Softwares Required 157](#_Toc35495444)

[Create a tomcat Server AMI in AWS 157](#_Toc35495445)

[Install Packer 159](#_Toc35495446)

[Packer Json 159](#_Toc35495447)

[Builders 159](#_Toc35495448)

[Provisioners 159](#_Toc35495449)

[DevOps Classroom Series – 03/Jan/2020 – Packer 160](#_Toc35495450)

[Creating AWS Image From Packer 160](#_Toc35495451)

[DevOps Classroom Series – Packer – 04/Jan/2020 162](#_Toc35495452)

[Provisioning in Packer 162](#_Toc35495453)

[Applying Shell Provisioner with script 162](#_Toc35495454)

[Variables in Packer 163](#_Toc35495455)

[DevOps Classroom Series – Packer – 05/Jan/2020 164](#_Toc35495456)

[Azure VM Image from Packer 164](#_Toc35495457)

[Manual Creation 164](#_Toc35495458)

[Using Packer 164](#_Toc35495459)

[Variables 165](#_Toc35495460)

[Useful Functions in Packer 165](#_Toc35495461)

[DevOps Classroom Series – 06/Jan/2020 – Terraform 167](#_Toc35495462)

[Terraform 167](#_Toc35495463)

[Installation 167](#_Toc35495464)

[Terraform in CD Pipelines (DevOps) 167](#_Toc35495465)

[Concepts 168](#_Toc35495466)

[DevOps Classroom Series – 07/Jan/2020 – Terraform 168](#_Toc35495467)

[Terraform Example for creating simple AWS Resource 168](#_Toc35495468)

[Setup Terraform DEV Environment 168](#_Toc35495469)

[DevOps Classroom Series – 08/Jan/2020 172](#_Toc35495470)

[Resource Dependencies 172](#_Toc35495471)

[DevOps Classroom Series – 09/Jan/2020 174](#_Toc35495472)

[Terraform file organization 174](#_Toc35495473)

[Adding some stuff to existing network 175](#_Toc35495474)

[Variable support to Terraform 176](#_Toc35495475)

[DevOps Classroom Series – 10/Jan/2020 – Terraform 178](#_Toc35495476)

[terraform cli 178](#_Toc35495477)

[Terraform outputs 178](#_Toc35495478)

[Terraform modules 178](#_Toc35495479)

[Next Sections 179](#_Toc35495480)

[DevOps Classroom Series – 20/Jan/2020 179](#_Toc35495481)

[Data-Sources 179](#_Toc35495482)

[Terraform provisioning 180](#_Toc35495483)

[Backends 180](#_Toc35495484)

[DevOps Classroom Series – 21/Jan/2020 181](#_Toc35495485)

[Terraform Provisioning 181](#_Toc35495486)

[Backends 183](#_Toc35495487)

[Exercise 183](#_Toc35495488)

[DevOps Classroom Series – 25/Jan/2020 (Azure DevOps) 185](#_Toc35495489)

[Azure DevOps (Visual Studio Team Services) 185](#_Toc35495490)

[VSTS Terms Equivalent in Jenkins 185](#_Toc35495491)

[DevOps Classroom Series – 23/Jan/2020 187](#_Toc35495492)

[Docker 187](#_Toc35495493)

[Evolution 187](#_Toc35495494)

[DevOps Classroom Series – 24/Jan/2020 189](#_Toc35495495)

[Microservices Deployments 189](#_Toc35495496)

[Container 189](#_Toc35495497)

[Role in Docker as a DevOps Engineer 189](#_Toc35495498)

[First Steps towards Containerization 190](#_Toc35495499)

[DevOps Classroom Series – 25/Jan/2020 190](#_Toc35495500)

[Docker Images and Registry 190](#_Toc35495501)

[Components of Docker 190](#_Toc35495502)

[What happens when we execute docker container run <image> 190](#_Toc35495503)

[How to create Docker Image? 191](#_Toc35495504)

[Instruction Based Approach to be followed. 191](#_Toc35495505)

[DevOps Classroom Series – 27/Jan/2020 191](#_Toc35495506)

[Manual Creation of Apache Server 191](#_Toc35495507)

[Creating a Docker Image for Apache Server 191](#_Toc35495508)

[Installing Docker on Linux Platforms 192](#_Toc35495509)

[Manual Creation of Spring-Petclinic (Spring Boot) 192](#_Toc35495510)

[Docker image creation for Spring-Petclinic 192](#_Toc35495511)

[Dockerfile Instructions 193](#_Toc35495512)

[DevOps Classroom Series – 28/Jan/2020 193](#_Toc35495513)

[Importance of Tags in Docker 193](#_Toc35495514)

[Dockerfile instructions 193](#_Toc35495515)

[Entrypoint 193](#_Toc35495516)

[Other instructions 194](#_Toc35495517)

[DevOps Classroom Series – 29/Jan/2020 195](#_Toc35495518)

[Docker references 195](#_Toc35495519)

[Docker Internals 195](#_Toc35495520)

[DevOps Classroom series – 30/Jan2020 195](#_Toc35495521)

[Container States 195](#_Toc35495522)

[Docker Container Creation modes 195](#_Toc35495523)

[Listing Containers 196](#_Toc35495524)

[Removing Containers 196](#_Toc35495525)

[Docker CLI 196](#_Toc35495526)

[DevOps Classroom Series – 31/Jan/2020 197](#_Toc35495527)

[Image Layers 197](#_Toc35495528)

[Commands used 197](#_Toc35495529)

[DevOps Classroom Series – 02/Feb/2020 - Morning 197](#_Toc35495530)

[Docker Storage Driver 197](#_Toc35495531)

[Docker Volumes 197](#_Toc35495532)

[Docker Volume Command line 197](#_Toc35495533)

[Bind mount volumes 199](#_Toc35495534)

[Dockerfile 199](#_Toc35495535)

[DevOps Classroom Series – 02/Feb/2020 - Evening 200](#_Toc35495536)

[Docker Essential Command line 200](#_Toc35495537)

[Dockerfile instructions 200](#_Toc35495538)

[Docker Single Host networking and Port forwarding 201](#_Toc35495539)

[DevOps Classroom Series – 04/Feb/2020 203](#_Toc35495540)

[Docker Network Drivers 203](#_Toc35495541)

[Creating a new bridge network 203](#_Toc35495542)

[DevOps Classroom Series – 05/Feb/2020 204](#_Toc35495543)

[Multi-host Docker networking 204](#_Toc35495544)

[DevOps Classroom Series – 06/Feb/2020 204](#_Toc35495545)

[Docker Swarm 204](#_Toc35495546)

[DevOps Classroom Series – 09/Feb/2020 206](#_Toc35495547)

[Dockerfile instructions (others) 206](#_Toc35495548)

[CI/CD Pipeline with Docker containers 206](#_Toc35495549)

[Registry 206](#_Toc35495550)

[Sample Scenario: Push gameoflife to docker hub 206](#_Toc35495551)

[multistage builds 207](#_Toc35495552)

[DevOps Classroom Series – 07/Feb/2020 209](#_Toc35495553)

[Kubernetes 209](#_Toc35495554)

[Where is it used? 209](#_Toc35495555)

[Why Kubernetes 209](#_Toc35495556)

[Basic Working Style on Kubernetes 209](#_Toc35495557)

[Kubernetes Architecture (First View) 209](#_Toc35495558)

[Pet vs Cattle 210](#_Toc35495559)

[DevOps Classroom Series – 08/Feb/2020 211](#_Toc35495560)

[Why Kubernetes 211](#_Toc35495561)

[Kubernetes Architecture 211](#_Toc35495562)

[Kubernetes Single Master Cluster 211](#_Toc35495563)

[DevOps Classroom Series – 10/Feb/2020 212](#_Toc35495564)

[Kubernetes Objects 212](#_Toc35495565)

[Basic Workflow 212](#_Toc35495566)

[Pod 212](#_Toc35495567)

[DevOps Classroom Series – 11/Feb/2020 214](#_Toc35495568)

[Important Concepts of K8s 214](#_Toc35495569)

[Workloads 214](#_Toc35495570)

[Pod 214](#_Toc35495571)

[Controllers 214](#_Toc35495572)

[Replication Controller 214](#_Toc35495573)

[Daemon Set 215](#_Toc35495574)

[DevOps Classroom Series – 12/Feb/2020 215](#_Toc35495575)

[Controllers 215](#_Toc35495576)

[Service 216](#_Toc35495577)

[DevOps Classroom Series – 14/Feb/2020 218](#_Toc35495578)

[**Kubernetes Deployments** 218](#_Toc35495579)

[DevOps Classroom Series – 15/02/2020 220](#_Toc35495580)

[Storage in K8s 220](#_Toc35495581)

[Kubernetes Storage 220](#_Toc35495582)

[Persistent Volumes Workflow 220](#_Toc35495583)

[Networking in K8s 221](#_Toc35495584)

[DevOps Classroom 16/Feb/2020- K8s Start to End 222](#_Toc35495585)

[Journey of Openmrs-core from code to K8s 222](#_Toc35495586)

[OpenMRS-Core Architecture 222](#_Toc35495587)

[Dockerfile Creation 222](#_Toc35495588)

[Setting up AKS k8s Cluster 223](#_Toc35495589)

[K8s Specifications 223](#_Toc35495590)

[How to enable communications between pods 226](#_Toc35495591)

[DevOps Classroom Series – 22/Feb/2020 229](#_Toc35495592)

[Openshift 229](#_Toc35495593)

[Openshift Product Lines 229](#_Toc35495594)

[Openshift Exercise 229](#_Toc35495595)

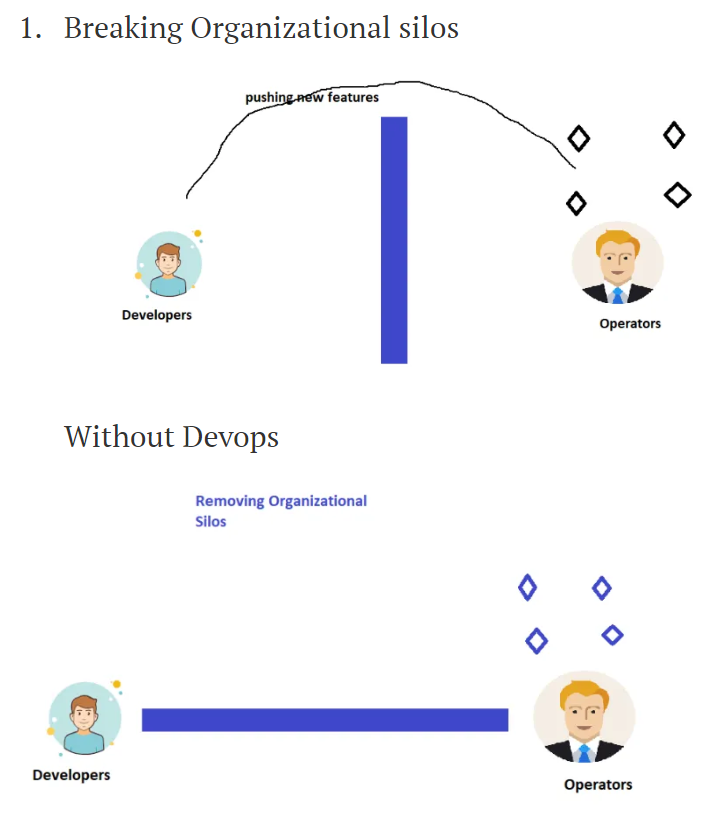
# SRE (Site Reliability Engineering)

**OCTOBER 22, 2019**

## DevOps Classroom Series- SRE – 22/Oct/2019

### Site Reliability Engineering – Intro

**DevOps**



With DevOps

2. Making frequent & smaller releases

3. Its ok to fail (Errors are normal)

4. Leverage Automation tools

5. Measure Everything

**SRE**

* Class SRE implements DevOps

### How SRE implements DevOps

1. Breaking Organizational silos:
   * SRE in the team along with developers
   * Developers & SRE’s use same tools for deployments
2. Making frequent & smaller releases
   * Risk
3. Its ok to fail (Errors are normal)
   * We define Error Budgets
4. Leverage Automation tools
   * Toil
5. Measure Everything
   * Observability , Alerting etc

* Metrics:
  + SLI (Service Level Indicator)
    - At current point of time
    - eg. is service up or down
  + SLO (Service Level Objective)
    - At combine points of time
    - Is your service up for 99.9% time during last week/month
  + SLA (Service Level Agreement)
    - Service will up and running for 99.5% over a period of one month

**OCTOBER 23, 2019**

## DevOps Classroom Series – 23/Oct/2019

### Primary Reference

* [Overview](https://directdevops.blog/2019/10/22/site-reliability-engineering-overview/)

### SLI (Service Level Indicator)

* Indicator of Availability of your application/service
* Sample Indicators:
  + Latency of the home page over last 5 minutes will be less than 300ms for 99.9% of requests

### SLO (Service Level Objective)

* SLI binds over a period of time
* Sample:
  + Latency of homepage over a year will be less than 300 ms for 99.9% of the requests

### SLA (Service Level Agreements)

* Sample:
  + Customer will be offered free credits if 99.5% of the requests over a year fail to achieved the latency of less than 300 ms

### Problem

* We build systems and they fail at some point.
* What’s the SRE approach towards failures

### Risks

* You can make aggressive deployments as long as you are within Error Budget.
* If Error Budget is exceeded no more deployments

### Error Budget

* Allowed time in minutes or hours of failure.
* Sample:
  + SLO : Latency will be less than 300 ms for 99.9% of request over the year
  + ERROR budget is what is left of total time after removing SLO (100-99.9) \* 365 \* 24 \* 60/ 100 = 525.6 minutes/year
  + SLO : Latency will be less than 300 ms for 99.99% of request over the year
  + ERROR Budget (100-99.99) \* 365 \* 24 \* 60 /100 = 52.5 minutes/year

### Error Budget Burndown

* Error Budget used
* Fast Burndowns
* Slow Burndowns

### Toil

* Repetitive manual work that can be automated
* Focus on Toils which are more frequent than infrequent ones

**OCTOBER 24, 2019**

## DevOps Classroom Series – SRE – 24/Oct/2019

### Monitoring and Alerting

* Alert on SLIs or SLOs
* Turn off all the other alerts

### Observability

* Three different things to observe
  + Logs
  + Metrics
  + Traces
* Monitoring system with High level failures which navigates to
  + Logs
  + Metrics
  + Traces

### Incident Mangement

* Clear cut Ways of Working defined by SRE.
* Following Roles are available to deal the situation
* Incident Commander Role is allocated when the incident is recorded.
* Incident Commander has following activiteis
  + Plan the Work to Resolve incident or delegate to Planning Lead (New Role create for incident)
  + Do Operations to Resolve or delegate to Operations Lead (New Role create for incident)
  + Make necessary Communications or delegate to Communications Lead (New Role create for incident)
  + Once the incident is resolved Create Postmortem Documnet

**OCTOBER 25, 2019**

## DevOps Classroom Series – 25/Oct/2019

### Metrics to measure Reliability

* MTTF (Mean Time To Failure):
  + Time from the begining of deployment till the first failure is reported
* MTBF (Mean Time Between Failures):
  + Time between Two Failures
* MTTR (Mean Time To Repair):
  + Time Taken to repair (resolve the incident)

### Post-Mortem

* Blameless
* Every Person involved in Incident Respose will have to prepare this document
* Initial Draft of this document will/can be from Incident Commander
* In Post-mortem unlike RCA (Root Cause Analysis), document can speak about multiple factors contribute the Error.
* All the actions taken, what can be done to eliminate the same problem from reocurrance/
* [Sample Postmortem Doc](https://landing.google.com/sre/sre-book/chapters/postmortem/)

### Ansible and Chef

* Enable Debug Logs
* Integrate Ansible/Chef Logs to Monitoring System
* Production Deployments should be Canary

### Docker, Kubernetes

* Enable Logging Drivers
* Change K8s deployment strategy to be Canary

### Monitoring

* Make monitoring Systems Observations

**OCTOBER 27, 2019**

## DevOps Classroom Series – 27/Oct/2019

**Monitoring**

### System Monitoring

* Monitor Basic System Details
  + Is Server Up
  + Is Http Page responding
  + Is Datbase Query Responding
  + Whats the free disk space at that moment
  + What is CPU Utilization at that moment
  + What is Network load at that moment
  + What is Disk IO Activity at that moment
* It can be completely our responsibility (DevOps/SRE)
* Tools:
  + Nagios
  + Zabbix
  + Icinga

### Application Monitoring

* Detailed monitoring of your application
  + How much memory is my application consuming
  + Whats the current number of concurrent users on my application
  + What are my applications logs speaking
  + What are my applications traces telling
  + What are failure patterns
* Need collaboration with Dev to accomplish this monitoring
* Tools:
  + Elastic Stack
  + Splunk
  + App Dynamics
  + Application Insights

### Terms

* Metric: Some measurement in terms of units of System/Application. Eg CPU Utilization, Newtork In/Out
* Charts: Charts are metrics aggregated over time
* Logs:
  + System Logs
  + Application Logs
* Dashboard: A unified view of every thing that matters.
* Alert: concern about a system raised to person(s)

### Expectations

* Health of the System
* Identify Failure Patterns
* Do analytics to suggest better customer experience.

### SRE Expectations

* Reduce Noise and Alert on SLIs or SLOs
* Make your Monitoring Observable

# NAGIOS

**Nagios**

* Has two Versions
  + Core
  + Enterprise
* Nagios Core Installation:
  + Involves downloading Nagios Code
  + Building the Nagios Code
  + Configuring Nagios
  + For the above nagios uses **make**
* Nagios
  + Plugins
  + Commands

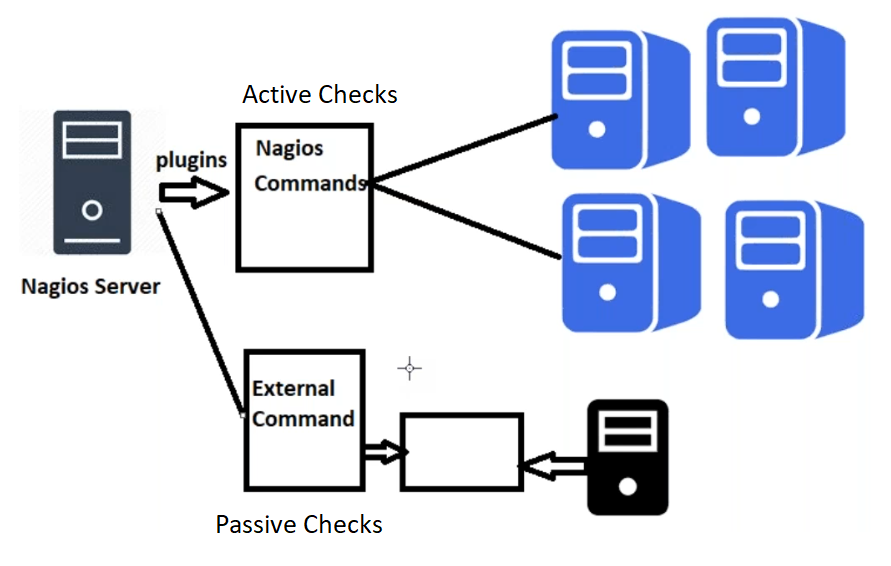
**OCTOBER 29, 2019**

## DevOps Classroom Series 29/Oct/2019

### System Monitoring using Nagios

### Terms

* Checks:
  + Active
  + Passive
* Servers: Host
* Commands:
* Contacts:
* Alerts:



### Different Options of Nagios

* Nagios Core: Open Source
* Nagios XI : Commercial Version

### Nagios Core

* Installation:
  1. Install Nagios by downloading the code and compiling
  2. Install Nagios Plugins by downloading the code and compiling
* References
  1. [Nagios Core](https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/quickstart.html)
  2. [Nagios Core on Ubuntu](https://support.nagios.com/kb/article/nagios-core-installing-nagios-core-from-source-96.html#Ubuntu)

### Configuration files

* In Nagios the main configuration file is **nagios.cfg**. This consits of nagios configurations like logs, other config locations, status update intervals.
* **commands.cfg** consists of reusable commands used to check the status and notify
* **timeperiods.cfg** consists of reusable timeperiods to run the checks
* **contacts.cfg** consists of user/group communication details

### Status

* Failure
  + Soft Failure
  + Hard Failure
* OK
* Pending

# ELASTIC STACK

**(Elasticsearch, Kibana, Beats, Logstash)**

**OCTOBER 30, 2019**

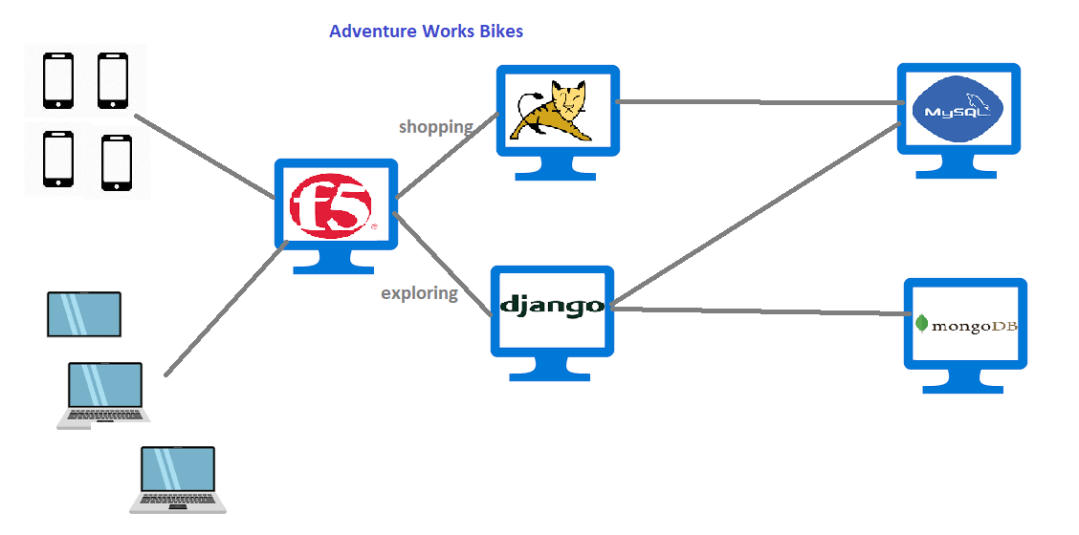
## DevOps Classroom Notes 30 Oct 2019 – Elastic Stack Usecase

### Scenario

* To understand Elastic Stack, We assume the organization as **AdventureWorks** in this series.
* Lets see different application/db servers used by **Adventure Works**



### Application-1 Architecture

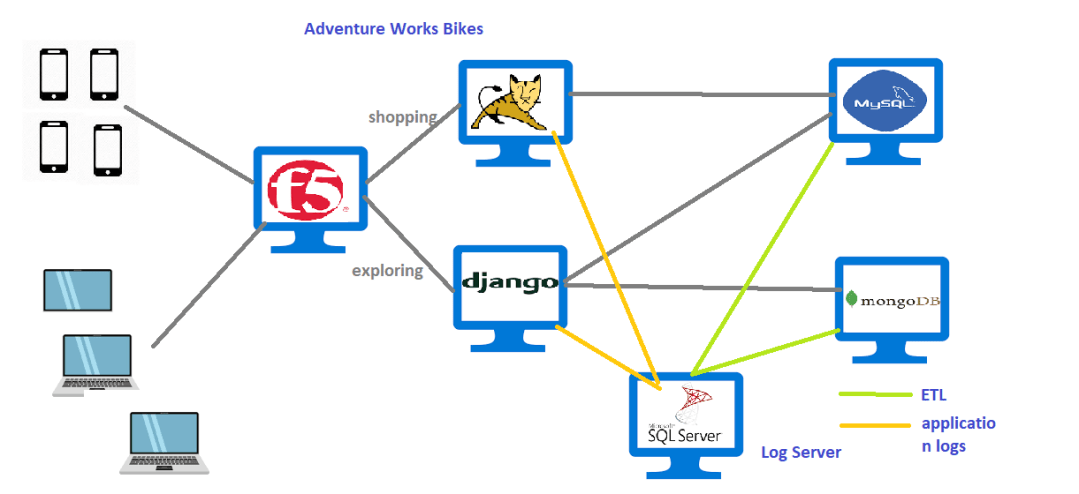


* To make the above application observable, we need to monitor
  + Logs
  + Traces
  + Metrics

### Enabling Log Monitoring

* It is very difficult to login into each server for the logs
* We need a central location for looking into logs of each applciation/server
* To acheive this, we have the following options
  + Store all the logs into Database like mysql/oracle
  + Use Syslog / Event log services

### Lets Examine SQL Approach



* In this Approach we send the logs of application to the Central SQL Server
* To send the logs of mysql and mongodb we might use ETL (Extract, Transform and Load) to see the logs in SQL Server
* It looks like the problem is solved.
* Problems with this approach:
  + Logs in the SQL server needs to be queried. But logs are text. Text searching in SQL is not that great.
  + Logs of all the different servers generally will be in different formats

### Possible Solution

* We need a system which can process different log formats and do the necessary adjustments
* We also need a system which can search text quickly
* Elastic Stack comes to the rescue

### Elastic Stack

* It is suite of Products
* In our Application’s Scenario we use Elastic Stack as our monitoring/logging platform.
* Elastic Stack = ELBK

#### Elastic Search

* Effecient Text Searching Distributed System.

#### LogStash

* Log Parsing and Conversions

#### Kibana

* Building Dashboards

#### Beats

* Agents which run on nodes which export logs to LogStash or ElasticSearch and/or report metrics.



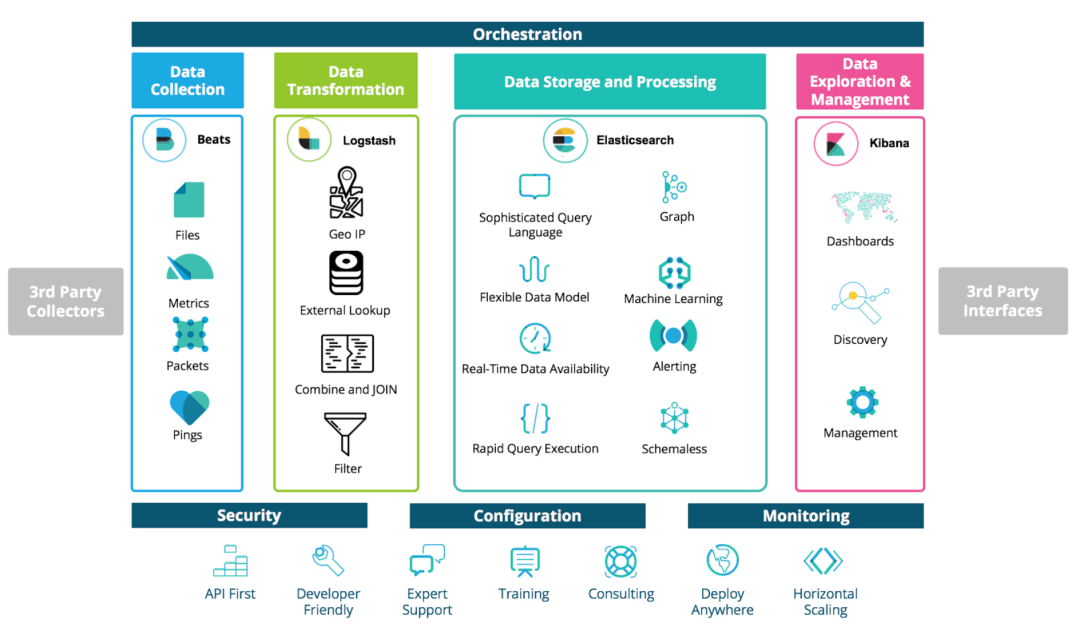
**OCTOBER 30, 2019**

## DevOps Classroom Notes 30 Oct 2019 – Elastic Stack – Introduction and Core Concepts (Notes from directdevops.blog)

### Elastic Stack

* Is suite of products Elastic Search, Kibana, Beats and Logstash
* They Reliably and securely taken data into from any source, in any format, then search, analyze and visualize in real time
* [Refer Here](https://www.elastic.co/what-is/elk-stack) for the short history about elastic stack
* [Refer Here](https://directdevops.blog/2019/10/30/devops-classroom-notes-30-oct-2019-elastic-stack-usecase/) for basic overview of Elastic Stack in Monitoring/Logging use case

### Elastic Stack Components



note: This image is from elastic.co blogs

### Elastic Search

* Core of the Elastic Stack.
* Is used to store and perform analytics on the data.
* Built on a [Apache Lucene](https://lucene.apache.org/)

### Benefits

* Schema-less and document-oriented:
  + No restrictions on strict structure, Any Json documents can be stored.
  + A document is mostly equivalent to a record in relational database table.
* Searching:
  + Elastic Search is superior at full-text searches
  + Full-text search is searching through all the terms of all the documents available in database. (Much like how Google Search Works rather than a database SELECT query)
* Analytics:
  + Elastic Search supports a wide variety of aggregations for analytics
* API and Client Libraries:
  + Elastic Search has client libraries in almost all the popular languages
  + Elastic Search has a very Rich REST API which works on http protocol.
* Horizontal Scaling:
  + Supports easily scaling number of Elastic search nodes.
  + Adding a new node is as easy as creating a new node in same network, with virtually no extra configuration.
* Fault-tolerant:
  + Clusters can continue running even when there is a failure.
  + In Node failure scenarios, data is replicated to another node in the cluster
  + In Network failure scenarios, new node is elected as master to keep cluster running.

### Installation of elastic Search

* In this Series I would be using an Centos-7 Machine launched in AWS CLoud with 2 vCpus and 8 GB of RAM. The version of Elastic Search will be 7.4. Installation Method would be RPM Based.
* Create an EC2 machine with Centos 7 AMI and ensure you have atleast 9200 and 22 ports opened in Security Groups.
* Once the machine is created, ssh into the centos 7 Machine
* Installation steps are [referred from here](https://www.elastic.co/guide/en/elasticsearch/reference/7.4/rpm.html)
* Execute the following Commands

sudo rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch

wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.4.1-x86\_64.rpm

wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.4.1-x86\_64.rpm.sha512

shasum -a 512 -c elasticsearch-7.4.1-x86\_64.rpm.sha512

sudo rpm --install elasticsearch-7.4.1-x86\_64.rpm

* Remember one important stuff, 50% of the system’s memory should be allocated to JVM. For that edit the file /etc/elasticsearch/jvm.options and change **-Xms1g** to **-Xms4g**and **-Xmx1g** to **-Xmx4g**. Please find the [related documentation](https://www.elastic.co/guide/en/elasticsearch/reference/current/heap-size.html)

-Xms4g

-Xmx4g

* Now change the node name and the network configurations in **/etc/elasticsearch/elasticsearch.yml** by setting the following and remove # in the beginning to uncomment the line
  + cluster.name: direct-devops
  + node.name: node1
  + network.host: \_site\_
  + http.port: 9200
  + discovery.seed\_hosts: ["127.0.0.1", "[::1]", "<private ip of ec2>"]
* Find the whole elasticsearch.yml file below

# ======================== Elasticsearch Configuration =========================

#

# NOTE: Elasticsearch comes with reasonable defaults for most settings.

# Before you set out to tweak and tune the configuration, make sure you

# understand what are you trying to accomplish and the consequences.

#

# The primary way of configuring a node is via this file. This template lists

# the most important settings you may want to configure for a production cluster.

#

# Please consult the documentation for further information on configuration options:

# https://www.elastic.co/guide/en/elasticsearch/reference/index.html

#

# ---------------------------------- Cluster -----------------------------------

#

# Use a descriptive name for your cluster:

#

cluster.name: direct-devops

#

# ------------------------------------ Node ------------------------------------

#

# Use a descriptive name for the node:

#

node.name: node1

#

# Add custom attributes to the node:

#

#node.attr.rack: r1

#

# ----------------------------------- Paths ------------------------------------

#

# Path to directory where to store the data (separate multiple locations by comma):

#

path.data: /var/lib/elasticsearch

#

# Path to log files:

#

path.logs: /var/log/elasticsearch

#

# ----------------------------------- Memory -----------------------------------

#

# Lock the memory on startup:

#

#bootstrap.memory\_lock: true

#

# Make sure that the heap size is set to about half the memory available

# on the system and that the owner of the process is allowed to use this

# limit.

#

# Elasticsearch performs poorly when the system is swapping the memory.

#

# ---------------------------------- Network -----------------------------------

#

# Set the bind address to a specific IP (IPv4 or IPv6):

#

network.host: \_site\_

#

# Set a custom port for HTTP:

#

http.port: 9200

#

# For more information, consult the network module documentation.

#

# --------------------------------- Discovery ----------------------------------

#

# Pass an initial list of hosts to perform discovery when this node is started:

# The default list of hosts is ["127.0.0.1", "[::1]"]

#

discovery.seed\_hosts: ["127.0.0.1", "[::1]", "172.31.23.231"]

#

# Bootstrap the cluster using an initial set of master-eligible nodes:

#

#cluster.initial\_master\_nodes: ["node-1", "node-2"]

#

# For more information, consult the discovery and cluster formation module documentation.

#

# ---------------------------------- Gateway -----------------------------------

#

# Block initial recovery after a full cluster restart until N nodes are started:

#

#gateway.recover\_after\_nodes: 3

#

# For more information, consult the gateway module documentation.

#

# ---------------------------------- Various -----------------------------------

#

# Require explicit names when deleting indices:

#

#action.destructive\_requires\_name: true

* For more info on Configurations [refer here](https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html)
* Now execute the following commands to start the elastic search service
* Note: Open the Firewall port in Kibana for CentOS as below

# firewall-cmd --zone=public --add-port=9200/tcp --permanent

# firewall-cmd --zone=public --add-port=22/tcp --permanent

# firewall-cmd --reload

sudo systemctl daemon-reload

sudo systemctl restart elasticsearch

* If the elasticsearch has started successfully, execute the following statements in the ssh terminal and observe the results

curl -XGET '<private ip>:9200/\_cluster/health?pretty'

curl -XGET '<private ip>:9200/\_cluster/stats?human&pretty'

Directly, hit the below url from browser also

http://192.168.1.7:9200/\_cluster/health?pretty

http://192.168.1.7:9200/\_cluster/stats?human&pretty

* To continue further on understanding Elastic Search, lets install Kibana Console UI.

### Kibana

* As we are already familiar that Kibana is a visualization tool, lets continue to install Kibana

#### Kibana Installation

* I will be using Centos 7 ec2 machine with t2.micro (1 vCpu 1 GB RAM)
* Kibana installation is [referred from here](https://www.elastic.co/guide/en/kibana/current/rpm.html)
* Login into the created ec2 machine and execute the following commands on the terminal

sudo rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch

wget https://artifacts.elastic.co/downloads/kibana/kibana-7.4.1-x86\_64.rpm

shasum -a 512 kibana-7.4.1-x86\_64.rpm

sudo rpm --install kibana-7.4.1-x86\_64.rpm

* Now configure Kibana config file from **/etc/kibana/kibana.yml**change elasticsearch.hosts to ["http://<private ip of elastic search>:9200"] and server.host to "0.0.0.0" . For More info on configuration [refer here](https://www.elastic.co/guide/en/kibana/7.4/settings.html) Find the kibana.yml used by me below

# Kibana is served by a back end server. This setting specifies the port to use.

#server.port: 5601

# Specifies the address to which the Kibana server will bind. IP addresses and host names are both valid values.

# The default is 'localhost', which usually means remote machines will not be able to connect.

# To allow connections from remote users, set this parameter to a non-loopback address.

server.host: "0.0.0.0"

# Enables you to specify a path to mount Kibana at if you are running behind a proxy.

# Use the `server.rewriteBasePath` setting to tell Kibana if it should remove the basePath

# from requests it receives, and to prevent a deprecation warning at startup.

# This setting cannot end in a slash.

#server.basePath: ""

# Specifies whether Kibana should rewrite requests that are prefixed with

# `server.basePath` or require that they are rewritten by your reverse proxy.

# This setting was effectively always `false` before Kibana 6.3 and will

# default to `true` starting in Kibana 7.0.

#server.rewriteBasePath: false

# The maximum payload size in bytes for incoming server requests.

#server.maxPayloadBytes: 1048576

# The Kibana server's name. This is used for display purposes.

#server.name: "your-hostname"

# The URLs of the Elasticsearch instances to use for all your queries.

elasticsearch.hosts: ["http://172.31.23.231:9200"]

# When this setting's value is true Kibana uses the hostname specified in the server.host

# setting. When the value of this setting is false, Kibana uses the hostname of the host

# that connects to this Kibana instance.

#elasticsearch.preserveHost: true

# Kibana uses an index in Elasticsearch to store saved searches, visualizations and

# dashboards. Kibana creates a new index if the index doesn't already exist.

#kibana.index: ".kibana"

# The default application to load.

#kibana.defaultAppId: "home"

# If your Elasticsearch is protected with basic authentication, these settings provide

# the username and password that the Kibana server uses to perform maintenance on the Kibana

# index at startup. Your Kibana users still need to authenticate with Elasticsearch, which

# is proxied through the Kibana server.

#elasticsearch.username: "kibana"

#elasticsearch.password: "pass"

# Enables SSL and paths to the PEM-format SSL certificate and SSL key files, respectively.

# These settings enable SSL for outgoing requests from the Kibana server to the browser.

#server.ssl.enabled: false

#server.ssl.certificate: /path/to/your/server.crt

#server.ssl.key: /path/to/your/server.key

# Optional settings that provide the paths to the PEM-format SSL certificate and key files.

# These files validate that your Elasticsearch backend uses the same key files.

#elasticsearch.ssl.certificate: /path/to/your/client.crt

#elasticsearch.ssl.key: /path/to/your/client.key

# Optional setting that enables you to specify a path to the PEM file for the certificate

# authority for your Elasticsearch instance.

#elasticsearch.ssl.certificateAuthorities: [ "/path/to/your/CA.pem" ]

# To disregard the validity of SSL certificates, change this setting's value to 'none'.

#elasticsearch.ssl.verificationMode: full

# Time in milliseconds to wait for Elasticsearch to respond to pings. Defaults to the value of

# the elasticsearch.requestTimeout setting.

#elasticsearch.pingTimeout: 1500

# Time in milliseconds to wait for responses from the back end or Elasticsearch. This value

# must be a positive integer.

#elasticsearch.requestTimeout: 30000

# List of Kibana client-side headers to send to Elasticsearch. To send \*no\* client-side

# headers, set this value to [] (an empty list).

#elasticsearch.requestHeadersWhitelist: [ authorization ]

# Header names and values that are sent to Elasticsearch. Any custom headers cannot be overwritten

# by client-side headers, regardless of the elasticsearch.requestHeadersWhitelist configuration.

#elasticsearch.customHeaders: {}

# Time in milliseconds for Elasticsearch to wait for responses from shards. Set to 0 to disable.

#elasticsearch.shardTimeout: 30000

# Time in milliseconds to wait for Elasticsearch at Kibana startup before retrying.

#elasticsearch.startupTimeout: 5000

# Logs queries sent to Elasticsearch. Requires logging.verbose set to true.

#elasticsearch.logQueries: false

# Specifies the path where Kibana creates the process ID file.

#pid.file: /var/run/kibana.pid

# Enables you specify a file where Kibana stores log output.

#logging.dest: stdout

# Set the value of this setting to true to suppress all logging output.

#logging.silent: false

# Set the value of this setting to true to suppress all logging output other than error messages.

#logging.quiet: false

# Set the value of this setting to true to log all events, including system usage information

# and all requests.

#logging.verbose: false

# Set the interval in milliseconds to sample system and process performance

# metrics. Minimum is 100ms. Defaults to 5000.

#ops.interval: 5000

# Specifies locale to be used for all localizable strings, dates and number formats.

# Supported languages are the following: English - en , by default , Chinese - zh-CN .

#i18n.locale: "en"

* Note: Open the Firewall port in Kibana for CentOS as below

# firewall-cmd --zone=public --add-port=5601/tcp --permanent

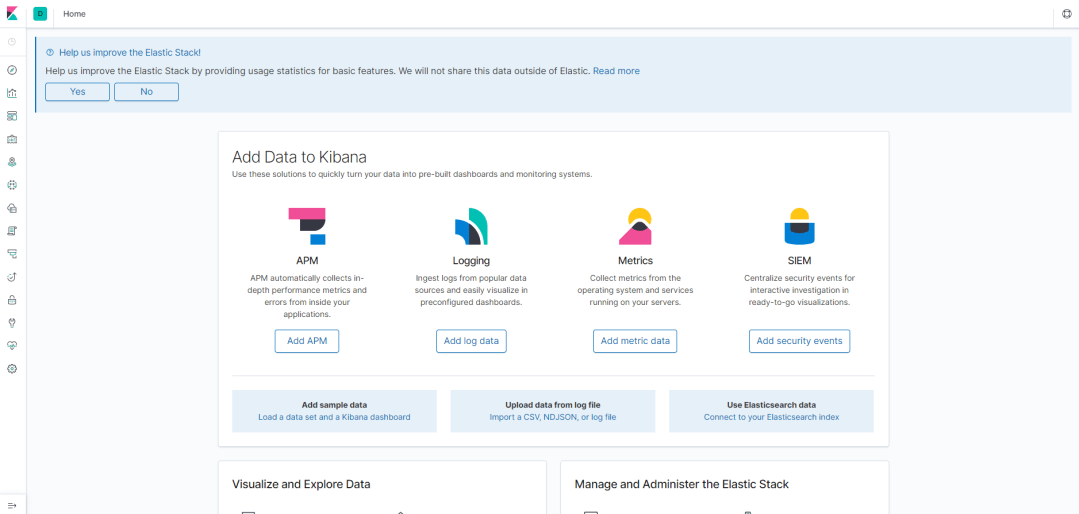
# firewall-cmd --zone=public --add-port=22/tcp --permanent

# firewall-cmd --reload

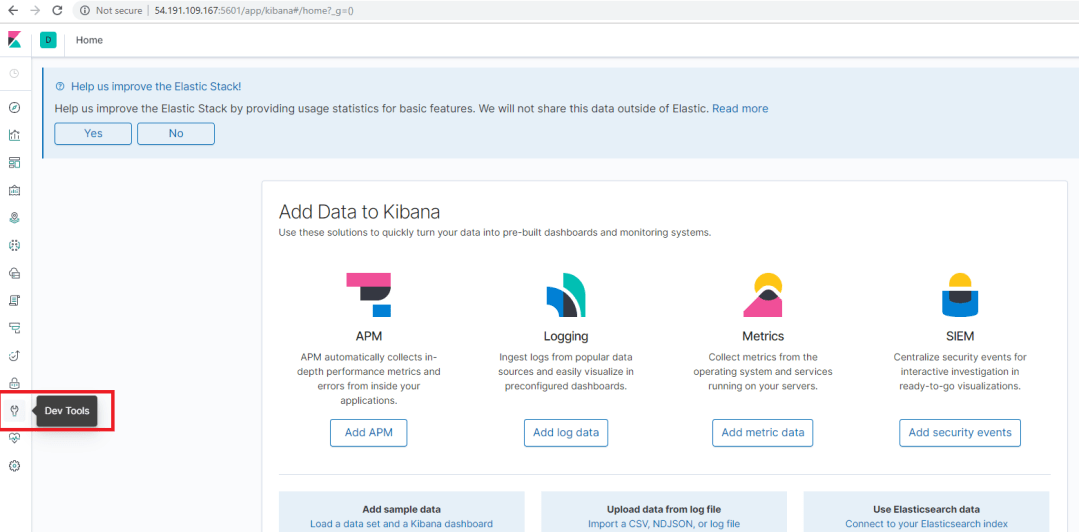
* Start kibana by executing

sudo systemctl start kibana

* Now Navigate to http://<public ip kibana>:5601 and you should see the home page



* We cannot do any visualizations yet, because we don’t have any data so far
* We can still use the **dev tools** section to test the elastic search apis. Click on DevTools Sections as shown in the image below and you will be navigated to console



* We will be using this Console for understanding the Elastic Search

### Core Concepts of Elastic Search

* Relational Databases have rows, columns, tables and schemas where as Elastic Search is a document-oriented store which has Json documents. These Json-documents are organized into different indexes, types.

#### Document

* Data in elastic search is stored in the format of JSON
* Document is very similar to record in Relational Database
* A Sample Document

{

"id": 1,

"name": "khaja",

"blog": "https://directdevops.blog",

"organization": "QualityThought",

"courses": ["AWS", "Azure", "DevOps", "Linux", "Windows", "Python"],

}

* In addition to field send by the user, Elastic Search stores the internal fields for metadata. These fields are as follows
  + \_id: unique identifier of document, just like primary key in a database. This can be auto generated or specified by the user
  + \_type: Type of the document
  + \_index: index name of the document

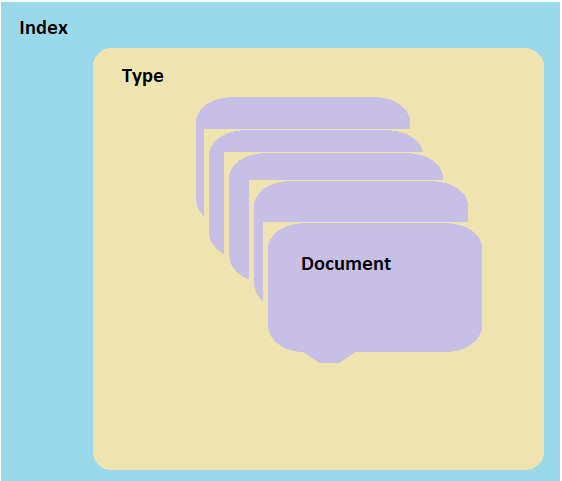
#### Indexes

* Index is a container that stores and manages documents of single type in Elastic Search.

#### Types

* Documents with mostly common set of fields are grouped under one Type.

#### Relation Between Index, Type and Documents



#### Nodes and Cluster

* Since Elastic Search is a Distributed System, it has nodes and clusters
* Node is a single server of Elastic Search
* A cluster is formed by one or more nodes, Every Elastic Search node is always part of cluster.

#### Shards and replicas

* And index consists of documents of some type, Shards helps in distributing an index over the cluster.
* Shards help in dividing documents of single index over multiple nodes.
* Process of dividing data among shards is **Sharding**. **Sharding** is Elastic Searches way of scaling and parallelism.
* By default every index is configured to have five shards in Elastic Search.
* You can specify the number of Shards while creating an index
* Since systems might fail in a cluster replica shards or replicas are created and stored in cluster.
* Despite the failure, Elastic Search runs due to this feature.
* Replicas are also support querying

**OCTOBER 31, 2019**

## DevOps Classroom Notes 31 Oct 2019 – Elastic Stack and Kibana Loading Data and API

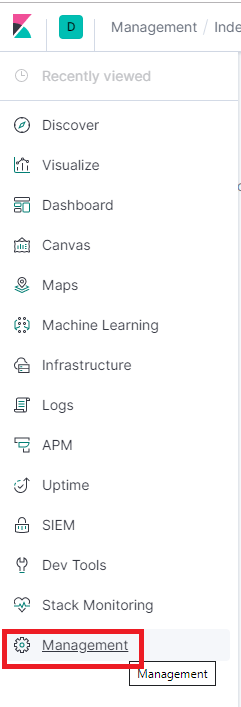
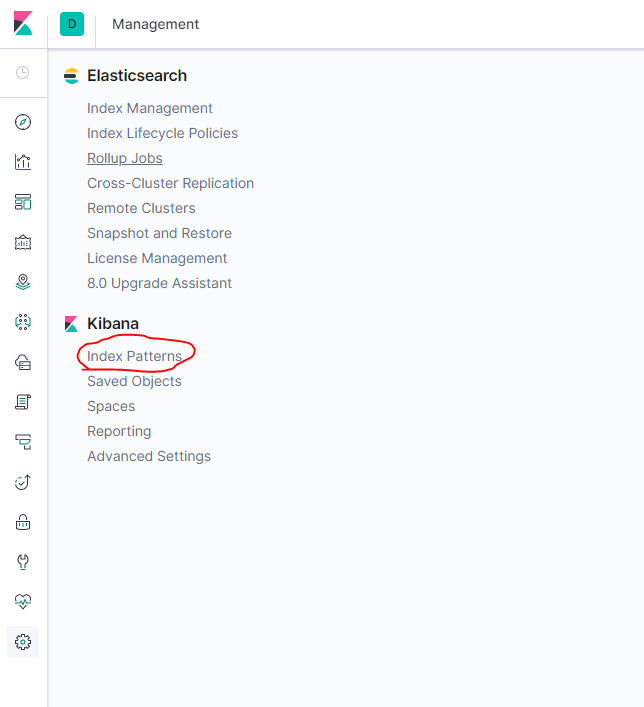
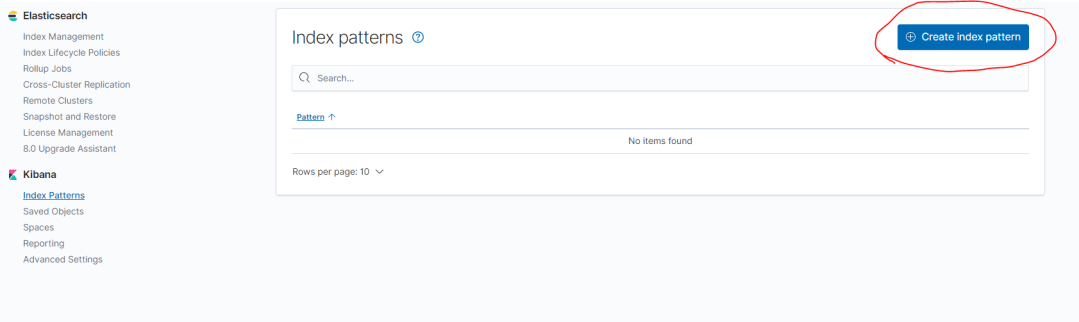
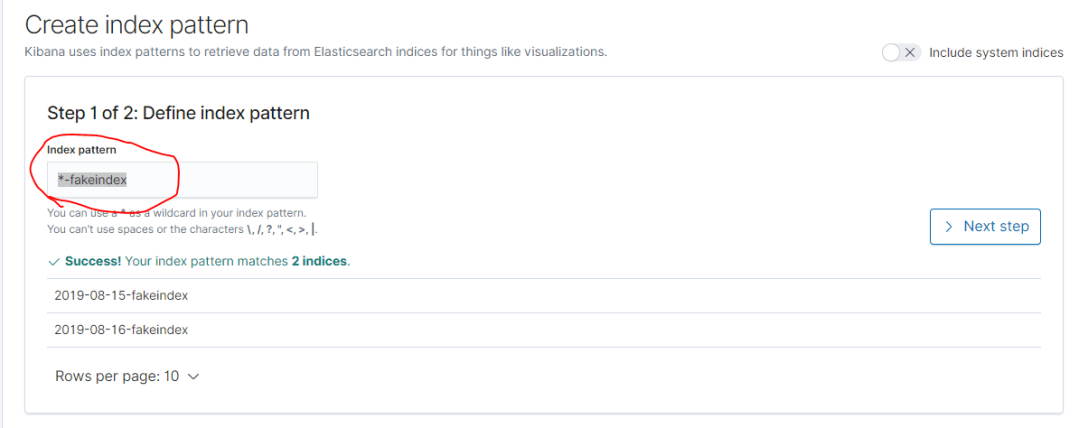
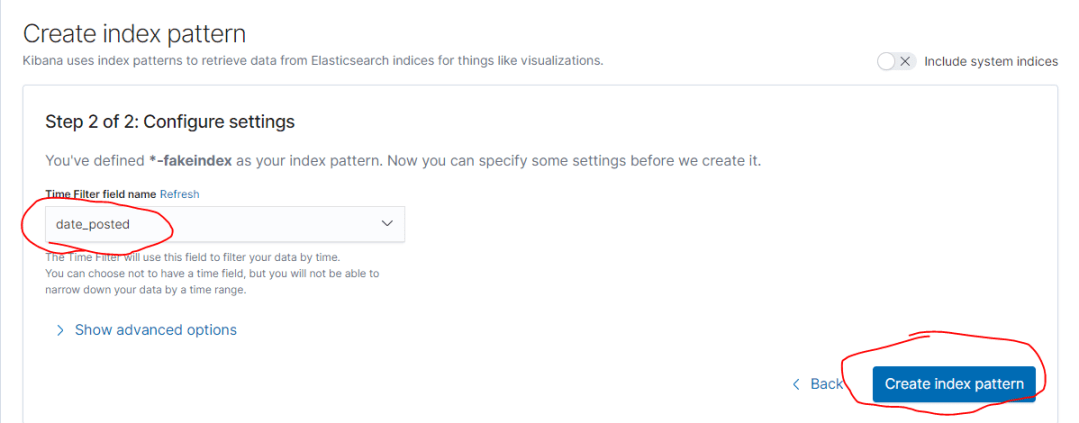
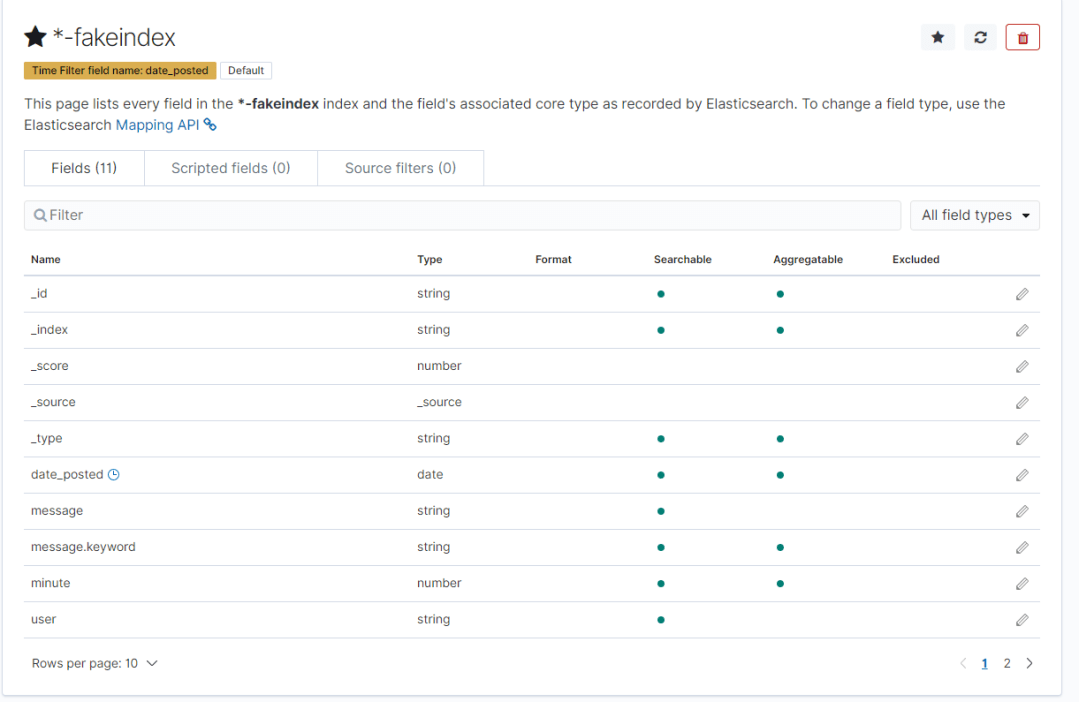
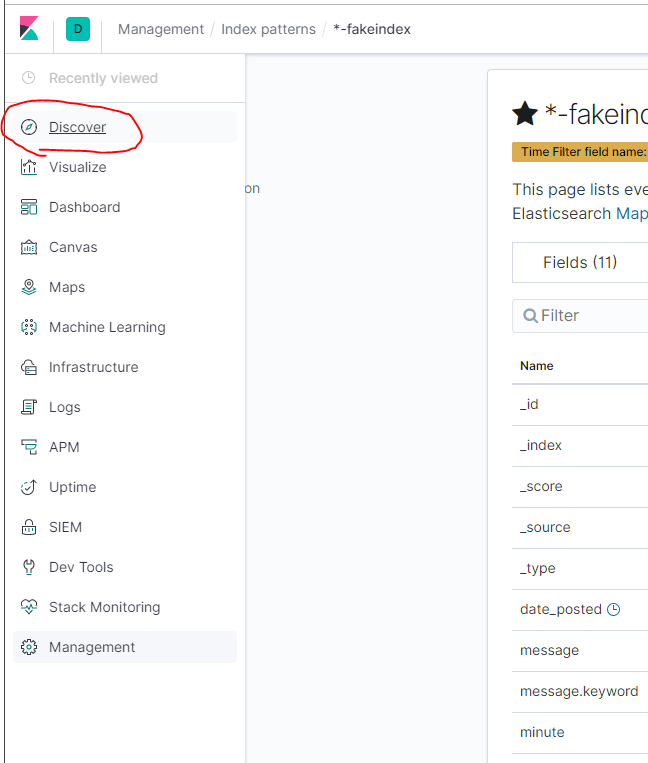
### Mappings and Data types

* Elasticsearch supports a wide variety of data types for different situations where you want to store text data, numbers, booleans, binary objects, arrays, objects, geo-points, geo-shapes, and many other data types, such as IP addresses.
* [Refer here](https://www.elastic.co/guide/en/elasticsearch/reference/current/sql-data-types.html) for complete list of Datatypes
* Mapping is the process of defining how document are indexed.
* Two Kinds of Mappings can be done
  + Dynamic
  + Explicit
* [Refer here](https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html) for complete documentation on Mappings

### Index API

* [Refer here](https://www.elastic.co/guide/en/elasticsearch/reference/current/indices.html) for Index APIs
* Lets use Kibana Console to experiment with the Index APIs
* Basic structure of the Index APIs are

http://<elasticsearch host>:<port>/<index>/<type>/<document id>

* To experiment with elastic search we need to generate fake data. For that i have written a very simple python program.
* Make sure you make changes to Ip address and execute pip install requests before you run this program
* Program can be found [here](https://gist.github.com/shaikkhajaibrahim/accb5653472d0d0fe8ab17374f24c5ba#file-elasticfakedata-py)
* This program generates 10 days of log data, with each date being in a index
* Execute the program with the following command python <filename>.py
* Wait for some time and Open Kibana UI.
* Navigate to Kibana Management 
* Select the Index Patterns 
* Click on Create Index Pattern 
* Enter **\*-fakeindex** as shown in image below and click on next step 
* Select the time field as date\_posted and Create Index Pattern 
* You should be able to see the field mappings 
* Now click on Discover 
* Select Time period to be 1 year and you should be able to (Program starts publishing log data from Aug 15 2019)
* Note: There is one more program with fake cpu values [here](https://gist.github.com/shaikkhajaibrahim/ff985def8117dea9867f0a8a846dc08e#file-elasticfakecpu-py)

**NOVEMBER 1, 2019**

## DevOps Classroom Series 01/Nov/2019

### When to create indexes in Elastic Search Monitoring Use case

* We create index for similar records daily.
* Example: Webserver indexes might be apache-2019-08-31, apache-2019-09-01

### To Create Indexes We need Index APIs

* [Index APIs](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index_.html)

### Visualize in Kibana

* [Refer](https://directdevops.blog/2019/10/31/elastic-stack-and-kibana-loading-data-and-api/)

### Beats

* Beats are agents which can export information to elastic search/logstash
* We have Beats for various purposes
  + Metric Beats
  + File Beats
  + Packet Beat
  + Heart Beat
* [Refer](https://www.elastic.co/downloads/beats)

**NOVEMBER 3, 2019**

## DevOps Classroom Series – 3/Nov/2019

### Install Logstash on Centos7 using rpm

* Create a centos 7 machine
* Connect to it using ssh
* install java 8 using

sudo yum install java-1.8.0-openjdk-devel -y

java -version

* Download binaries from elastic site and install. In this series i will be using 7.4 version

wget https://artifacts.elastic.co/downloads/logstash/logstash-7.4.2.rpm

sudo rpm -i logstash-7.4.2.rpm

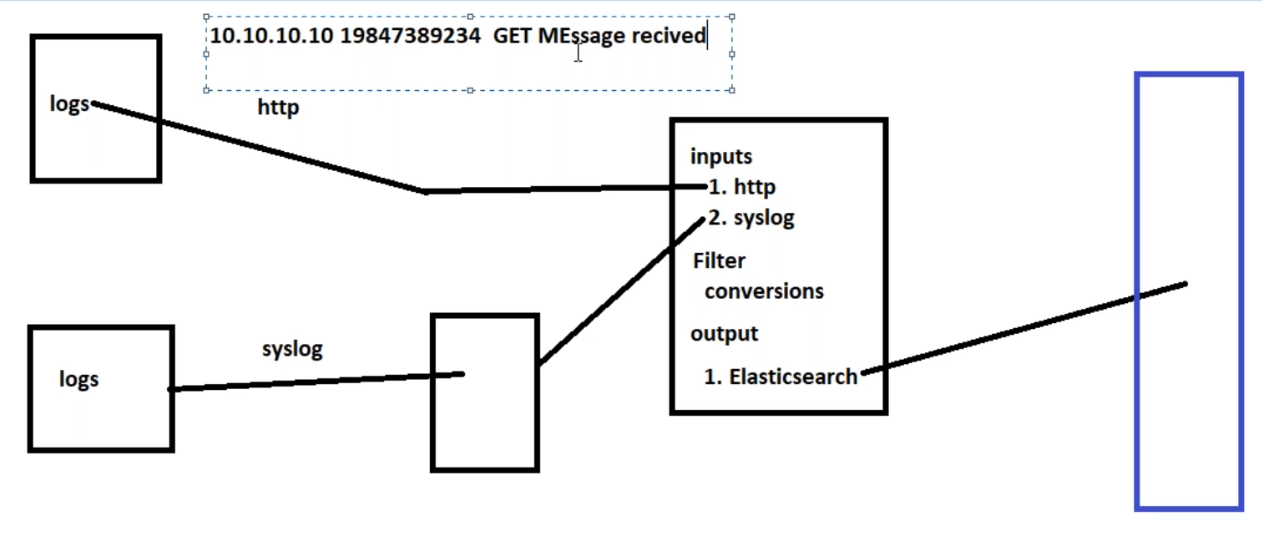
* Configuration files of logstash are located @ /etc/logstash
* Logstash works with configuration files where you write rules. Basic configuration files syntax would be

input

filter

output

* Logstash can receive the inputs from input plugins for whole list [refer here](https://www.elastic.co/guide/en/logstash/current/input-plugins.html)
* [Output Plugins](https://www.elastic.co/guide/en/logstash/current/output-plugins.html)
* [Filters](https://www.elastic.co/guide/en/logstash/current/filter-plugins.html)



### Golden Signals

* Request per Second
* Error Rate
* Latency
* Saturation
* Utilzation

### Reason

Making your system observable reduces MTTR (Mean Time TO Repair)

### Exercise

* Create Dashboards representing requests pers second of Apache Server using logstash, elastic search and kibana

# ANSIBLE

**NOVEMBER 5, 2019**

## DevOps Classroom Series- Ansible 05/November/2019

### Configuration Management

* For any application to work we need some softwares.
* Options for installing necessary softwares and deploying application

### Manual

* Administrator will install the necessary softwares
* Admin will deploy the application
* Problems:
  + Slow
  + Error Prone

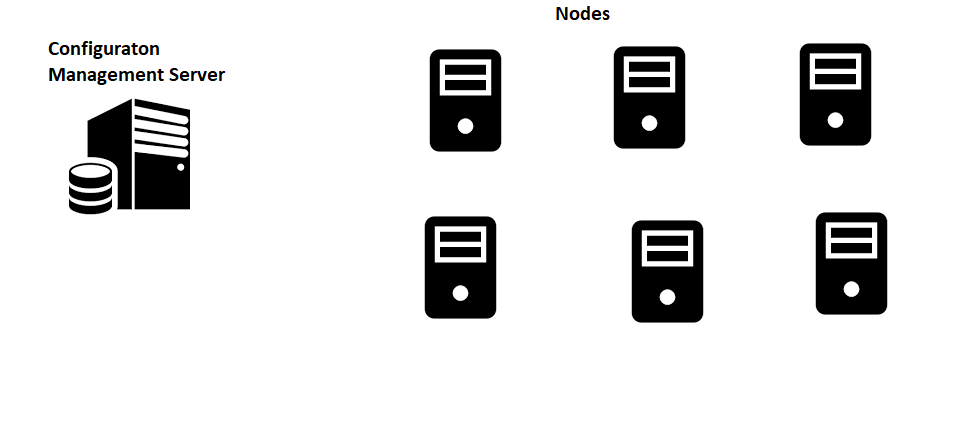
### Shell/Powershell Scripts

* Create a shell/powershell script to perform deployment
* Problems:
  + Multiple ways of doing the same activity. Because it impacts Readability and Maintainability.
  + Every script has assumptions
  + Scripts are Procedural (How it has to done)
* Example:
  + Create a file:
    - Create a file using touch command
    - files path should be at /home/ubuntu/test.txt

### Configuration Management

* In this user will write the desired state
* Scripts are Declarative (What has to be done)
* The major theme is to maintain desired state.
* CM has an important property call as **idempotance**
* Example:
  + Create a file
    - I want an empty file at /home/ubuntu/test.txt
* CM can be done using many tools, some of them are
  + Ansible
  + Chef
  + Salt
  + Puppet
  + CFEngine
  + PowerShell DSC

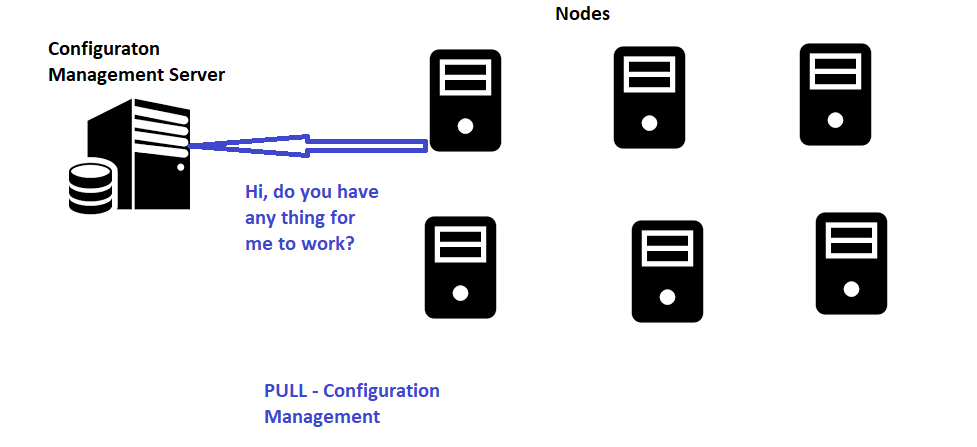
### Architecture of Configuration Management



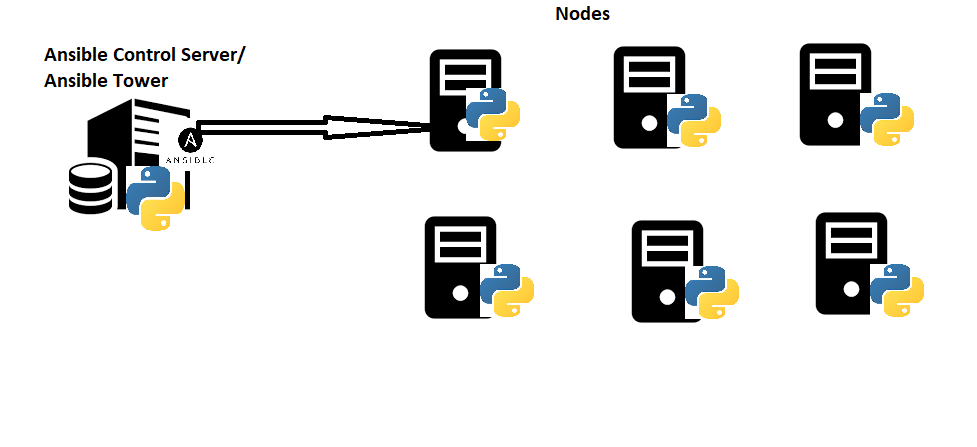
#### PUSH



#### PULL



### Ansible Falls under Push based Configuration Management



* Ansible is push based CM tool by default.
* Software requirements of Ansible
  + Ansible Control Server:
    - Python
    - Ansible
  + Nodes:
    - Python

### What do the user need to provide to ansible

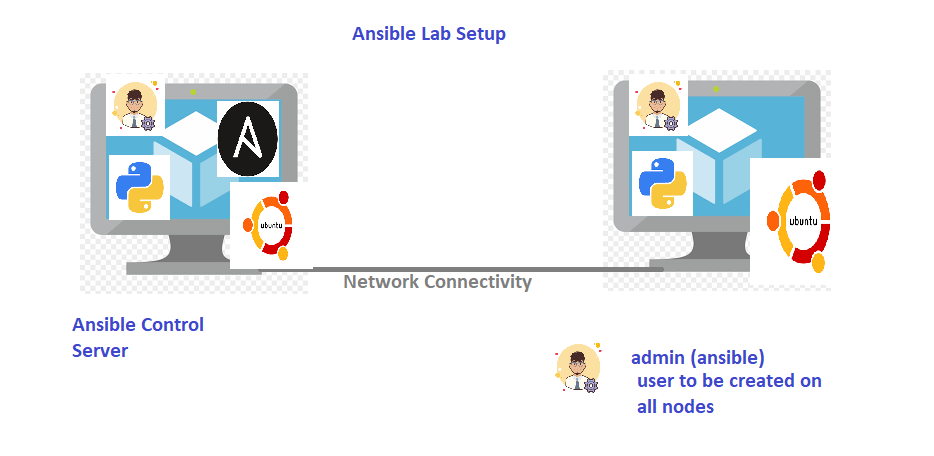
* Playbook:
  + Written in **YAML**
  + YAML is all about data representation
* Inventory:
  + List of nodes to be used by Ansible Control

**NOVEMBER 6, 2019**

## DevOps Classroom Series – Ansible Installation – 06/Nov/2019

### Ansible Installation

### Lab Setup



1. Create two ec2 machines with t2.micro in AWS
2. Tag one machine as ACS and other as Node-1
3. Login into ACS and become root user user
4. Enable Password Based Authentication

vi /etc/ssh/sshd\_config

change the contents

# PasswordAuthentication no => PasswordAuthentication yes

service sshd restart

1. Create a user **devops** and give sudo permissions

adduser devops

visudo

# in the file opened

devops ALL=(ALL:ALL) NOPASSWD:ALL

# To test this

su devops

sudo apt-get update

# It should not ask for password and it should complete the command execution

1. If python is not installed on ACS, install python
   * To install ansible on Ubuntu [Refer Here](https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#latest-releases-via-apt-ubuntu)
   * On the node execute the following lines

sudo apt update

sudo apt install software-properties-common

sudo apt-add-repository --yes --update ppa:ansible/ansible

sudo apt-get install python -y

1. Install Ansible on ACS
2. Repeat the steps from 3 to 6 on **node-1**
3. Login into ACS as **devops** and configure password less login into node-1

ssh devops@<acspublicip>

ssh-keygen

ssh-copy-id devops@<node1privateip>

1. Do the following to test connectivity

sudo mv /etc/ansible/hosts /etc/ansible/hosts.orig

sudo vi /etc/ansible/hosts

# add node1-private ip and save the file

# check by executing

ansible -m ping all

### Refer the following

* [VIM](https://www.openvim.com/)

**NOVEMBER 7, 2019**

## DevOps Classroom Series – 7/Nov/19

### How to think in Ansible

* Given any task, list down all the steps
* In Ansible you need express them with Desired State
* For that we start with Playbook (YAML File)
* Each Step ~> task in ansible playbook
* In Ansible, Task are executed by **Modules**
* **Modules** are atomic units of Ansible which perform execution/automation.
* In Ansible There are loads of modules. [Refer Here](https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html)

**NOVEMBER 8, 2019**

## DevOps Classroom Series – Ansible – 8/Nov/2019

### Ansible Workflow

1. Make a list of all the commands/steps for deploying the applciation
2. Execute them manually.
3. Write a Ansible Playbook with the task for each command/step.

### Sample-Application

#### Spring-pet Clinic

* This application is based out of java spring framework
* Prereqs:
  + Java 8 should be installed.
* Download the Spring-petclinic.jar file file from [here](https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/spring-petclinic.jar)
* once application is downloaded execute the following command in the terminal

java -jar <path to spring-petclinic.jar>

#### Manual Steps

* Deploy Spring Pet Clinic which by default runs on 8080 port

sudo apt-get update

sudo apt-cache search jdk | less

sudo apt-get install openjdk-8-jdk -y

java -version

wget https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/spring-petclinic.jar

java -jar spring-petclinic.jar

* Access your application using http://<publicip>:8080

### Student Courses Application

* This is application written in Python Flask
* Prereqs:
  + Python 3
  + pip-3
* Steps

sudo apt-get update

sudo apt-get install python3 python3-pip git -y

git clone https://github.com/DevProjectsForDevOps/StudentCoursesRestAPI.git

cd StudentCoursesRestAPI

pip3 install -r requirements

python3 app.py

* Access your application using http://<publicip>:8080

### Using Ansible For deployment of Sample Application

* This can be done by writing Playbooks.
* To Write effective Playbooks, we need to know
  + YAML
  + Modules
  + Tasks
  + Ansible-ENvironment Variables
  + Inventory
  + Ansible Configuration

### YAML

* Is much like JSON.
* It is collection of name value pairs
* [Refer here](https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html) for YAML Syntax

**NOVEMBER 10, 2019**

## DevOps Classroom Series 10/Nov/2019 – Ansible

### How to automate application deployments using Ansible

* Ansible Expects to automate using Playbooks.
* Playbooks is YAML with predefined Structure.

### Playbooks Structure

* Hosts: On which nodes defined in **Inventory** do you want to execute the playbook
* Syntax of Hosts: name value pair

hosts: all

* Remote\_user: User which ansible executes your automation.
* Become: Do you want run the playbook using sudo (set Boolean value)

hosts: all

become: yes

* Tasks: list of individual task
* Task: writing some of task properties

hosts: all

become: yes

tasks:

- name: <value>

<module\_name>:

<module\_param1>: <module\_value1>

..

..

..

<module\_paramn>: <module\_valuen>

state: <module supported state>

* Example Playbook:

- hosts: all

become: yes

tasks:

- name: install git

package:

name: git

state: present

- name: uninstall git

package:

name: git

state: absent

### Simple Scenario: Installing Tree on all the nodes

* Manual Execution:

sudo apt-get update

sudo apt-get install tree -y

### Expressing the above Manual Execution in Ansible

1. Inventory: i need a list with ip addresses/dns names(fqdn)
2. Playbook: lets write the basic structure

- hosts: all

become: yes

tasks:

- name: update ubuntu packages

- name: install tree

1. Now we need Modules to automate. Lets define modules. Modules are smallest unit of work in Ansible. There are predefined Modules. You can also define custom modules.
2. To automate the activity lets search modules. lets build a google query around this

apt-get update in Ansible

1. Based of results, lets fill the modules in the above playbook

- hosts: all

become: yes

tasks:

- name: update ubuntu packages and install tree

apt:

name: tree

update\_cache: yes

state: present

1. Create an inventory file with ip addresses in each line
2. create a yaml file called as tree-playbook.yml with the following content

- hosts: all

become: yes

tasks:

- name: update ubuntu packages and install tree

apt:

name: tree

update\_cache: yes

state: present

1. Explore ansible-playbook command line by using ansible-playbook --help | less
2. Command to execute in this case is

ansible-playbook -i test-inventory tree-playbook.yml

### Module Categories In Ansible

* [Refer Here](https://docs.ansible.com/ansible/latest/modules/modules_by_category.html) for Ansible Module Categories

### Execution in Ansible

* Two possibilities:
  + Playbook:
    - Must if you are automating and also if it is frequent activity
    - Can execute multiple modules from one playbook file
  + Adhoc-Commands:
    - Infrequently used activities use adhoc commands.
    - Can execute only one module per command.

### Adhoc Commands in Ansible

* [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html) for complete documentation.
* Syntax:

ansible --help

ansible -i <inventory path> -m <modulename> -a "param1=value1 .. .. paramn=valuen" [-b] <all>

### Playbook in Ansible

* Collection of Plays.
* Each play

- hosts: all

become: yes

tasks:

..

..

- hosts: all

tasks:

..

..

* Each Play in a playbook defines the phase/kind of activities. Generally playbooks are divided into following plays as best practice
  + PRE-DEPLOYMENT: Installing necessary Software’s
  + DEPLOYMENT: Deploying the newly built package to the server
  + POST-DEPLOYMENT: Mostly cleanup activites
* Ansible modules like command, bash, shell etc.. are not idempotent.
* Any module which directly executes os commands is not idempotent and it is not recommended to use these modules until you have no other options.
* If you are using this module, idempotance is your responsibility.

**NOVEMBER 11, 2019**

## DevOps Classroom Series – Ansible – 11/Nov/2019

### Widely Used Modules

* Installations:
  + apt
  + yum
  + package
* Services:
  + service
  + systemd
* Files:
  + file
  + get\_url
* Templating
  + template

### Sample Deployment in Ansible

* We will try to deploy Apache with PHP modules
* Ubuntu Steps

sudo apt-get update

sudo apt-get install apache2 -y

sudo systemctl enable apache2

sudo apt-get install php libapache2-mod-php php-mysql php-cli -y

sudo vi /var/www/html/info.php

<?php

phpinfo();

?>

sudo systemctl restart apache2

* Centos Steps:

sudo yum install httpd -y

sudo systemctl enable httpd

sudo yum install php php-mysql

sudo vi /var/www/html/info.php

<?php

phpinfo();

?>

sudo systemctl restart httpd

* Initial Playbook for Ubuntu looks as shown below

---

- hosts: all

become: yes

tasks:

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

- name: enable and restart apache2 service

service:

name: apache2

enabled: yes

state: restarted

- name: install php

apt:

name: php

state: present

- name: install libapache2-mod-php

apt:

name: libapache2-mod-php

state: present

- name: install php-mysql

apt:

name: php-mysql

state: present

- name: php-cli

apt:

name: php-cli

state: present

- name: restart apache2

service:

name: apache2

state: restarted

* Playbook for Redhat/Centos looks as shown below

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

- name: enable and restart httpd

service:

name: httpd

enabled: yes

state: restarted

- name: install php

yum:

name: php

state: present

- name: install php-mysql

yum:

name: php-mysql

state: present

- name: restart httpd

service:

name: httpd

state: restarted

* Ansible Handlers: [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#handlers-running-operations-on-change)
* Lets change our playbooks by adding handlers.
* Ubuntu Playbook looks as shown below

---

- hosts: all

become: yes

tasks:

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

notify:

- restart and enable apache2

- name: install php

apt:

name: php

state: present

- name: install libapache2-mod-php

apt:

name: libapache2-mod-php

state: present

- name: install php-mysql

apt:

name: php-mysql

state: present

- name: php-cli

apt:

name: php-cli

state: present

notify:

- restart and enable apache2

handlers:

- name: restart and enable apache2

service:

name: apache2

enabled: yes

state: restarted

* RedHat Playbook looks as shown below

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

notify:

- restart and enable httpd

- name: install php

yum:

name: php

state: present

- name: install php-mysql

yum:

name: php-mysql

state: present

notify:

- restart and enable httpd

handlers:

- name: restart and enable httpd

service:

name: httpd

enabled: yes

state: restarted

* Now in the playbooks we can write it in a simplified way using looping constructs [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#with-items)
* After applying loops ubuntu playbook looks as shown below

---

- hosts: all

become: yes

tasks:

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

notify:

- restart and enable apache2

- name: install php modules

apt:

name: "{{ item }}"

state: present

loop:

- php

- libapache2-mod-php

- php-mysql

- php-cli

notify:

- restart and enable apache2

handlers:

- name: restart and enable apache2

service:

name: apache2

enabled: yes

state: restarted

* After applying loops redhat/centos playbook looks as shown below

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

notify:

- restart and enable httpd

- name: install php modules

yum:

name: "{{ item }}"

state: present

loop:

- php

- php-mysql

notify:

- restart and enable httpd

handlers:

- name: restart and enable httpd

service:

name: httpd

enabled: yes

state: restarted

**NOVEMBER 12, 2019**

## DevOps Classroom Series 12/Nov/2019

### Ansible Playbook for lamp (Contd..)

### Script from Previous Series

* Ubuntu:

---

- hosts: all

become: yes

tasks:

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

notify:

- restart and enable apache2

- name: install php modules

apt:

name: "{{ item }}"

state: present

loop:

- php

- libapache2-mod-php

- php-mysql

- php-cli

notify:

- restart and enable apache2

handlers:

- name: restart and enable apache2

service:

name: apache2

enabled: yes

state: restarted

* RedHat

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

notify:

- restart and enable httpd

- name: install php modules

yum:

name: "{{ item }}"

state: present

loop:

- php

- php-mysql

notify:

- restart and enable httpd

handlers:

- name: restart and enable httpd

service:

name: httpd

enabled: yes

state: restarted

### Goal is have one playbook for Redhat and Ubuntu

* Combining both the playbooks into one

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

notify:

- restart and enable httpd

- name: install php modules

yum:

name: "{{ item }}"

state: present

loop:

- php

- php-mysql

notify:

- restart and enable httpd

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

notify:

- restart and enable apache2

- name: install php modules

apt:

name: "{{ item }}"

state: present

loop:

- php

- libapache2-mod-php

- php-mysql

- php-cli

notify:

- restart and enable apache2

handlers:

- name: restart and enable httpd

service:

name: httpd

enabled: yes

state: restarted

- name: restart and enable apache2

service:

name: apache2

enabled: yes

state: restarted

* Things to be resolved are
  + Run apt tasks only on ubuntu and yum tasks only on redhat machines.
* To do this ansible facts come to the rescue.

### Ansible Facts

* Facts are information about the node.
* Facts can be gathered using setup module
* Facts can be used in the playbooks directly fact\_name or by using expression "{{ fact\_name }}"
* For Setup module [Refer Here](https://docs.ansible.com/ansible/latest/modules/setup_module.html)

### Ansible Conditionals

* use When in the tasks. [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#the-when-statement)

### Continuing with the Playbook combining facts and when statement

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

notify:

- restart and enable httpd

when: ansible\_facts['os\_family'] == "RedHat"

- name: install php modules

yum:

name: "{{ item }}"

state: present

loop:

- php

- php-mysql

notify:

- restart and enable httpd

when: ansible\_facts['os\_family'] == "RedHat"

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

notify:

- restart and enable apache2

when: ansible\_facts['os\_family'] == "Debian"

- name: install php modules

apt:

name: "{{ item }}"

state: present

loop:

- php

- libapache2-mod-php

- php-mysql

- php-cli

notify:

- restart and enable apache2

when: ansible\_facts['os\_family'] == "Debian"

handlers:

- name: restart and enable httpd

service:

name: httpd

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "RedHat"

- name: restart and enable apache2

service:

name: apache2

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "Debian"

* Need to optimize further by having less tasks and handlers bcoz they almost do the same job.

### Example of git installation

* Manual Steps

#centos

sudo yum install git -y

# ubuntu

sudo apt-get install git -y

* Playbook in ansible

---

- hosts: all

become: yes

tasks:

- name: install git

package:

name: git

state: present

### Goal: To make the Playbook Generic

* We need to use Anisble variables.

**NOVEMBER 13, 2019**

## DevOps Classroom Series -13/Nov/2019

### Goal: Write Common Tasks and Handlers

* From Previous Series this playbook will be our starting point

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

notify:

- restart and enable httpd

when: ansible\_facts['os\_family'] == "RedHat"

- name: install php modules

yum:

name: "{{ item }}"

state: present

loop:

- php

- php-mysql

notify:

- restart and enable httpd

when: ansible\_facts['os\_family'] == "RedHat"

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

notify:

- restart and enable apache2

when: ansible\_facts['os\_family'] == "Debian"

- name: install php modules

apt:

name: "{{ item }}"

state: present

loop:

- php

- libapache2-mod-php

- php-mysql

- php-cli

notify:

- restart and enable apache2

when: ansible\_facts['os\_family'] == "Debian"

handlers:

- name: restart and enable httpd

service:

name: httpd

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "RedHat"

- name: restart and enable apache2

service:

name: apache2

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "Debian"

* To replace apt or yum we can use package module. In the current Scenario, Package names are different for RedHat and Ubuntu
* Lets use Ansible Variables to remove the package name hard coding.
* Approach 1: Define Variables in inventory files.
* Initial Inventory looks as shown below

172.31.26.196

172.31.25.93

* Lets add Groups to the inventory to something like below in inventory file

[ubuntu]

172.31.26.196

[redhat]

172.31.25.93

[webservers]

172.31.26.196

172.31.25.93

* Lets add the variable package\_name to webserver group as shown below

[ubuntu]

172.31.26.196

[redhat]

172.31.25.93

[webservers]

172.31.26.196 package\_name=apache2

172.31.25.93 package\_name=httpd

* For now lets comment the install php modules section

---

- hosts: all

become: yes

tasks:

- name: install httpd

yum:

name: httpd

state: present

notify:

- restart and enable httpd

when: ansible\_facts['os\_family'] == "RedHat"

#- name: install php modules

# yum:

# name: "{{ item }}"

# state: present

# loop:

# - php

# - php-mysql

# notify:

# - restart and enable httpd

# when: ansible\_facts['os\_family'] == "RedHat"

- name: install apache2 and update packages

apt:

name: apache2

update\_cache: yes

state: present

notify:

- restart and enable apache2

when: ansible\_facts['os\_family'] == "Debian"

#- name: install php modules

# apt:

# name: "{{ item }}"

# state: present

# loop:

# - php

# - libapache2-mod-php

# - php-mysql

# - php-cli

# notify:

# - restart and enable apache2

# when: ansible\_facts['os\_family'] == "Debian"

handlers:

- name: restart and enable httpd

service:

name: httpd

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "RedHat"

- name: restart and enable apache2

service:

name: apache2

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "Debian"

* Now Replace apt and yum with package module and also replace all in hosts with webservers

---

- hosts: webservers

become: yes

tasks:

- name: install apache

package:

name: "{{ package\_name }}"

state: present

notify:

- restart and enable apache

when: ansible\_facts['os\_family'] == "RedHat" or ansible\_facts['os\_family'] == "Debian"

handlers:

- name: restart and enable apache

service:

name: "{{ package\_name }}"

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "RedHat" or or ansible\_facts['os\_family'] == "Debian"

* Now lets write the same inventory in YAML format with modules included

all:

children:

webservers:

hosts:

172.31.26.196:

package\_name: apache2

php\_modules:

- php

- libapache2-mod-php

- php-mysql

- php-cli

172.31.25.93:

package\_name: httpd

php\_modules:

- php

- php-mysql

* Now lets use php\_modules variable in playbook to install modules
* Ensure you use a debug module to print variables
* Right now playbook is

---

- hosts: webservers

become: yes

tasks:

- name: install apache

package:

name: "{{ package\_name }}"

state: present

notify:

- restart and enable apache

when: ansible\_facts['os\_family'] == "RedHat" or ansible\_facts['os\_family'] == "Debian"

- name: printing php modules

debug:

var: php\_modules

- name: install php modules

package:

name: "{{ item }}"

state: present

notify:

- restart and enable apache

when: ansible\_facts['os\_family'] == "RedHat" or ansible\_facts['os\_family'] == "Debian"

loop: "{{ php\_modules }}"

handlers:

- name: restart and enable apache

service:

name: "{{ package\_name }}"

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "RedHat" or ansible\_facts['os\_family'] == "Debian"

* php-modules is not recognized as variable, find the resolution.

**NOVEMBER 15, 2019**

## DevOps Classroom notes – 15/Nov/2019

### Ansible Variable Files

* Rather than having inventory and variables defined in same file, lets use Variable files.
* Variable files are of two categories
  + group\_vars
  + host\_vars
* Directory Structure should
  + Playbook
  + Inventory
  + group\_vars
    - group\_name
  + host\_vars
    - host\_name
* Group variables applies to all members of the group where as host variable applies to only that particular host.
* Lets design the directory structure for the following inventory

[ubuntu]

172.31.26.196

[redhat]

172.31.25.93

[webservers]

172.31.26.196

172.31.25.93

* The directory structure would be
  + lamp.yml
  + hosts
  + group\_vars
    - webservers
    - redhat
    - ubuntu
  + host\_vars
    - 172.31.26.196
    - 172.31.25.93
* Create files in the group\_vars and host\_vars

mkdir newplays

cd newplays

vi hosts

[ubuntu]

172.31.31.239

[redhat]

172.31.30.86

[webservers]

172.31.31.239

172.31.30.86

mkdir group\_vars

touch group\_vars/webservers

vi group\_vars/webservers

---

package\_name: httpd

php\_modules:

- php

mkdir host\_vars

touch group\_vars/172.31.31.239

vi group\_vars/172.31.31.239

---

package\_name: apache2

vi lamp.yml

---

- hosts: webservers

become: yes

tasks:

- name: install apache

package:

name: "{{ package\_name }}"

state: present

notify:

- restart and enable apache

when: ansible\_facts['os\_family'] == "RedHat" or ansible\_facts['os\_family'] == "Debian"

- name: printing php modules

debug:

var: php\_modules

- name: install php modules

package:

name: "{{ item }}"

state: present

notify:

- restart and enable apache

when: ansible\_facts['os\_family'] == "RedHat" or ansible\_facts['os\_family'] == "Debian"

loop: "{{ php\_modules }}"

handlers:

- name: restart and enable apache

service:

name: "{{ package\_name }}"

enabled: yes

state: restarted

when: ansible\_facts['os\_family'] == "RedHat" or ansible\_facts['os\_family'] == "Debian"

Run:

ansible-playbook –i hosts lamp.yml

* Refer this sample over [here](https://github.com/ansible/ansible-examples/blob/master/lamp_haproxy/group_vars/dbservers)

https://github.com/ansible/ansible-examples/blob/master/lamp\_haproxy/group\_vars/dbservers

* Lets execute this command

ansible-playbook -i hosts -e package\_name=stress lamp.yml

### Ansible Variable Precedence

* [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable)
* In majority of Cases variables are defined
  + In Playbook
  + In Inventory
  + In Inventory group\_vars and host\_vars
  + set fact
  + Roles
  + extra vars

### Ansible Special Variables

* [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#connecting-to-hosts-behavioral-inventory-parameters) (<https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#connecting-to-hosts-behavioral-inventory-parameters>)

### Adding a custom fail message for unsupported OS

* After adding a fail message for unsupported os, the playbook looks like

---

- hosts: webservers

become: yes

tasks:

- name: fail if it is unsupported os

fail:

msg: This playbook is supported only on redhat and debain variants

when: ansible\_facts['os\_family'] != "RedHat" and ansible\_facts['os\_family'] != "Debian"

- name: install apache

package:

name: "{{ package\_name }}"

state: present

notify:

- restart and enable apache

- name: printing php modules

debug:

var: php\_modules

- name: install php modules

package:

name: "{{ item }}"

state: present

loop: "{{ php\_modules }}"

- name: Download info php page

get\_url:

url: https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/info.php

dest: /var/www/html/info.php

notify:

- restart and enable apache

handlers:

- name: restart and enable apache

service:

name: "{{ package\_name }}"

enabled: yes

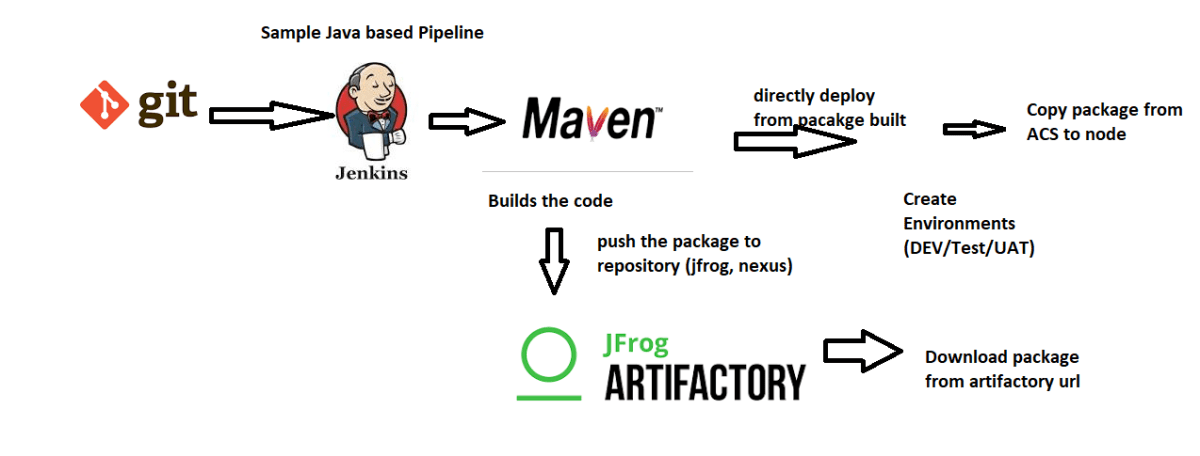
state: restarted

* In our next series we will cover about
  + files
  + templates
  + Roles
  + set facts (pre and post tasks)
  + Ansible tower

**NOVEMBER 17, 2019**

## DevOps Classroom Series – 16/Nov/2019

### Ansible Files



* In DevOps Pipelines, it will necessary to download/copy the package built by build system (Maven, Msbuild, make) to the Node.
* To Download the file into node use module **get\_url**
* To copy the file from ACS to node use **copy**
* Now lets use the same lamp playbook and create a file called as info.php with following contnets

<?php

phpinfo();

?>

* Now lets add the copy module to the existing playbook

---

- hosts: webservers

become: yes

tasks:

- name: fail if it is unsupported os

fail:

msg: This playbook is supported only on redhat and debain variants

when: ansible\_facts['os\_family'] != "RedHat" and ansible\_facts['os\_family'] != "Debian"

- name: install apache

package:

name: "{{ package\_name }}"

state: present

notify:

- restart and enable apache

- name: printing php modules

debug:

var: php\_modules

- name: install php modules

package:

name: "{{ item }}"

state: present

loop: "{{ php\_modules }}"

- name: copy the home page

copy:

src: info.php

dest: /var/www/html/info.php

notify:

- restart and enable apache

#- name: Download info php page

# get\_url:

# url: https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/info.php

# dest: /var/www/html/info.php

# notify:

# - restart and enable apache

handlers:

- name: restart and enable apache

service:

name: "{{ package\_name }}"

enabled: yes

state: restarted

### Exercise

* Create a file called as Readme.txt in the root folder.
* The contents of the file should be

Create by Ansible

Os family is <os >

Os distribution is <distribution>

version is <version>

### Solution to above Exercise

* Whenever you have dynamic values, ansible has a templating system which is widely used in python based developments. That templating system is Jinja
* Jinja templates are the files with extension **.j2**
* Lets try to solve the above situation
* Create a Readme.txt.j2 file with following content

Create by Ansible

Os family is {{ ansible\_os\_family }}

Os distribution is {{ ansible\_distribution }}

version is {{ ansible\_distribution\_version }}

* Write a simple playbook with following content

---

- hosts: all

tasks:

- name: copy the readme

template:

src: Readme.txt.j2

dest: "{{ ansible\_user\_dir }}/Readme.txt"

* Run this playbook and check the results
* In the J2 file facts and variables (user defined) can be used.
* Individual find and replaces can be done with a module **lineinfile**

### Pre-Tasks and Post-Tasks

* Pre-Tasks are the tasks which get executed before execution of tasks
* Post-Tasks are the tasks which get executed after execution of tasks

---

- hosts: all

become: yes

pre\_tasks:

- name: write a message

debug:

msg: "Starting Execution of Playbook"

tasks:

- name: install tree

package:

name: tree

state: present

post\_tasks:

- debug:

msg: "Playbook execution completed"

### How to Reuse Playbooks

* You already have lamp.yaml, now you want to use same lamp installation along with some other configurations
* You can call existing playbooks using include or import playbooks. [Refer](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_includes.html)
* Lets write a simple playbook to call other playbook

---

- hosts: all

become: yes

tasks:

- name: call lamp playbook

import\_playbook: lamp.yaml

- name: install mysql

package:

name: mysql-server

state: present

### Ansible roles

* Its all about reusability
* Using others automation efforts in ansible is simplified using roles
* Any user can share the work by converting playbook to ansible role and submit to ansible community. Ansible community which shares these roles is **Ansible Galaxy**
* As an example lets install java using Ansible Galaxy role. [Refer Here](https://galaxy.ansible.com/geerlingguy/java)
* Download the Ansible Role by executing ansible-galaxy install geerlingguy.java
* Now to install java lets write a playbook

---

- hosts: all

become: yes

roles:

- role: geerlingguy.java

### Creating Ansible Roles

* Sample folder structure
  + playbooks
    - roles
      * rolename
* Login into the ACS and create a new directory called roles and execute the following command

ansible-galaxy init lamp

* Now use lamp playbook to copy tasks in to tasks\main.yml, and do the same for handlers
* if you have any files or templates copy them in files or templates folder
* Now cd into playbooks folder and create one more folder called as environments
  + playbooks
    - roles
      * lamp (folder)
    - environments
      * DEV (file)
      * TEST (file)
      * UAT (file)
    - myplaybook.yml
* Edit myplaybook.yml file with

---

- hosts: webservers

become: yes

roles:

- role: lamp

* To create DEV Environment execute

ansible-playbook -i environments/DEV myplaybook.yml

**NOVEMBER 18, 2019**

## DevOps Classroom Series – 17/Nov/2019

### Ansible Parallelism

* Parallelism can be set in Ansible using an argument **–fork**.
* Default value is 5
* Scenario:
  + 100 nodes are in inventory and I run ansible-playbook test.yml, this execution happens in 20 batches
  + 100 nodes are in inventory and I run ansible-playbook -f 50 test.yml, this execution happens in 2 batches
  + 100 nodes are in inventory and I run ansible-playbook -f 500 test.yml, this execution happens in 1 batch

### Ansible custom facts

* I have a playbook, inside this playbook I need to decide on some module execution depending on other module execution.

Eg: only when you create a file, then only restart service and do further executions

---

- hosts: all

become: yes

tasks:

- name: create a file

file:

path: /tmp/test.txt

notify:

- create a fact

- name: restart service

service:

name: test

state: restarted

when file\_created == yes

handlers:

- name: create a fact

set\_fact:

file\_created: yes

* In any case if you want to store the value only till your playbook execution is completed, you can use a module called as **set\_fact**
* For more info [Refer Here](https://docs.ansible.com/ansible/latest/modules/set_fact_module.html)

https://docs.ansible.com/ansible/latest/modules/set\_fact\_module.html

### How to make non idempotent modules idempotent

* To make your non idempotent modules idempotent create some flags
* One easier implementation is create and check for file existence
* You can also try environmental variables

- hosts: appservers

become: yes

tasks:

- name: download spring pet clinic

get\_url:

url:

- name: execute spring pet clinic

command: java -jar spring-petclinic.jar

notify:

- create a flag

when: file\_path.stat.exists == False

handlers:

- name: create a flag

stat:

path: "/usr/share/test/iwashere"

register: file\_path

when: file\_path.stat.exists == False

### Register Variables

* [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#registering-variables)

https://docs.ansible.com/ansible/latest/user\_guide/playbooks\_variables.html#registering-variables

---

- hosts: all

become: yes

tasks:

- name: contents of file

command: cat /home/ansible/Readme.txt

register: readme\_output

- name: display contents

debug:

var: readme\_output

### Vault

* Configuration Management has to deal with passwords or sensitive contents
* We need a mechanism to encrypt the sensitive content and thats exactly what ansible-vault does.
* [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/vault.html#providing-vault-passwords)

https://docs.ansible.com/ansible/latest/user\_guide/vault.html#providing-vault-passwords

* Execution in ansible requires vault password or password file to passed

ansible-playbook -i hosts --vault-password-file /var/lib/password test.yaml

### Ansible Tower

* Tower provides authentication and reporting features to ansible control server
* Tower is web based platform for doing what you have done with ACS (Command line), we would have an user interface and REST API.
* Terms
  + Projects => Where your ansible playbooks are?
  + Inventory
  + Job
  + Schedule Jobs
  + Job Template
* In this series we will use Ansible Trail Version on AWS

### Dynamic Inventory

* If the node list is changing consistently, then how to handle this in inventory.
* Dynamic Inventory can be used.
* Dynamic Inventory is any script in any language which returns information about your nodes in json format specified by Ansible.
* Json structure should look like

{

"group001": {

"hosts": ["host001", "host002"],

"vars": {

"var1": true

},

"children": ["group002"]

},

"group002": {

"hosts": ["host003","host004"],

"vars": {

"var2": 500

},

"children":[]

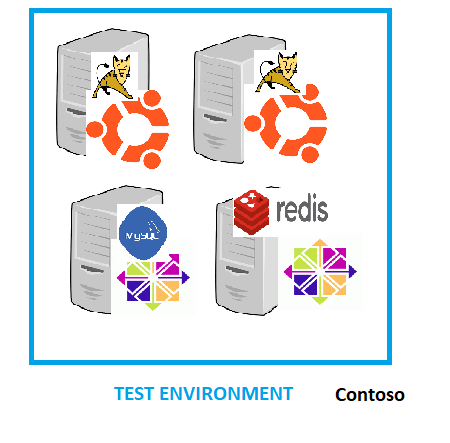
}

}

* [Refer Here](https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#inventory-script-example-aws-ec2) for sample AWS scenario.

https://docs.ansible.com/ansible/latest/user\_guide/intro\_dynamic\_inventory.html#inventory-script-example-aws-ec2

### DevOps Scenarios in Ansible

* Consider the belowArchitecture
* 
* To create this deployment, we can go with three roles
  + tomcat => tomcat on servers
  + mysql => mysql on server
  + redis => redis on server
* Then you write your playbook => contoso.yml

---

- hosts: appservers

become: yes

roles:

- role: tomcat\_deploy

- hosts: dbservers

become: yes

roles:

- role: mysql\_deploy

- hosts: cacheservers

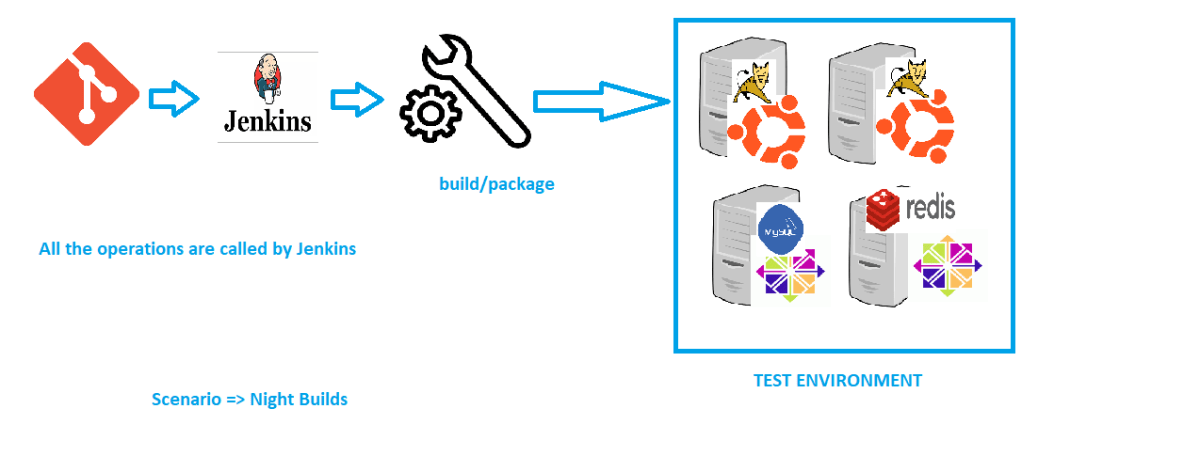
become: yes

roles:

- role: redis\_deploy

#### Realizing this with CI/CD

* Servers are already available with OS installed.
* Create an inventory with servers
* Create a Jenkins Pipeline/Freestyle Project with following steps
  + Get latest code from GIT
  + Build the code
  + Call ansible-playbook to deploy the configuration



#### Realizing this with CI/CD and Infra Provisioning

* Starting Point : Servers are not available
* Create a Jenkins Pipeline/Freestyle Project with following steps
  + Get latest code from GIT
  + Build the code
  + Call Terraform to create servers and then terraform calls ansible to do configuration management

#### Exercise

* Write ansible-playbook to deploy any of the sample application [from here](https://directdevops.blog/2019/11/08/devops-classroom-series-ansible-8-nov-2019/)

https://directdevops.blog/2019/11/08/devops-classroom-series-ansible-8-nov-2019/

* Try to make java -jar spring-petclinic.jar as a Linux service [Refer Here](https://medium.com/@manjiki/running-spring-boot-applications-as-system-services-on-linux-5ea5f148c39a)

https://medium.com/@manjiki/running-spring-boot-applications-as-system-services-on-linux-5ea5f148c39a

# CHEF

**NOVEMBER 19, 2019**

## DevOps Classroom Series – Chef – 19/Nov/2019

### Configuration Management

### Example Scenario

* Developer => submits code to git (VCS)
* CI/CD Engine picks up the code and does build/package
* CI/CD Engine Executes the Unit Tests
* CI/CD Engine needs to create Test Environments (System Test/Load Test/Performance Test) and execute the test cases
* Once tests are successful then post approval , you go live.

### Creating Environments

* Creating Environments is Deploying applications into servers.
* It might also include creating virtual Servers

### Sample Java Based application

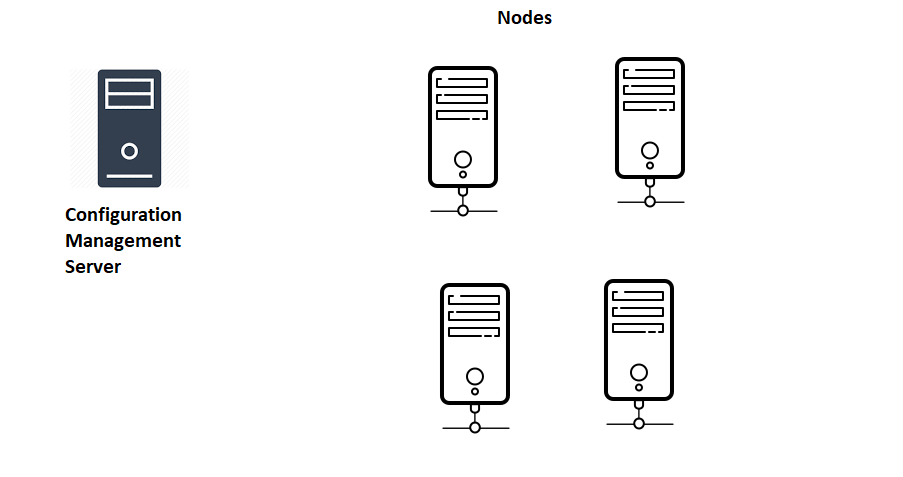
* Developer commits the Code
* Build and package is done by build tools like maven mvn package
* Result of Build/Package will be a file with jar/war extension
* To deploy application in server
  + Pre-requisites:
    - JDK 1.8 needs to be Installed
    - tomcat 8 needs to be installed
  + Deployment
    - Copy the package into some folder
    - restart tomcat
  + Post-Deployment:
    - Clean up temp directories

### Possible ways of deployment

* Manual:
  + Administrator executes the pre-requisites, deployment and post-deployment steps
  + Problems:
    - Time Consuming
    - Humans are not good at repetition
    - We cannot guarantee all admins will give same result
* Scripting:
  + We tend to write shell/PowerShell scripts to solve the deployment
  + In scripts we write instructions about how the deployment has to be done
  + Problems:
    - Scripting is generally based out of assumptions
    - Scripting is mostly unreadable.
* Configuration Management:
  + We use Declarative Way, which means you will write **what has to done** but not **how it has to done**
  + CM scripts are idempotent. Executing CM Scripts will always give the same state
  + In CM you write the desired state to be achieved.
  + Problems:
    - Learning Curve

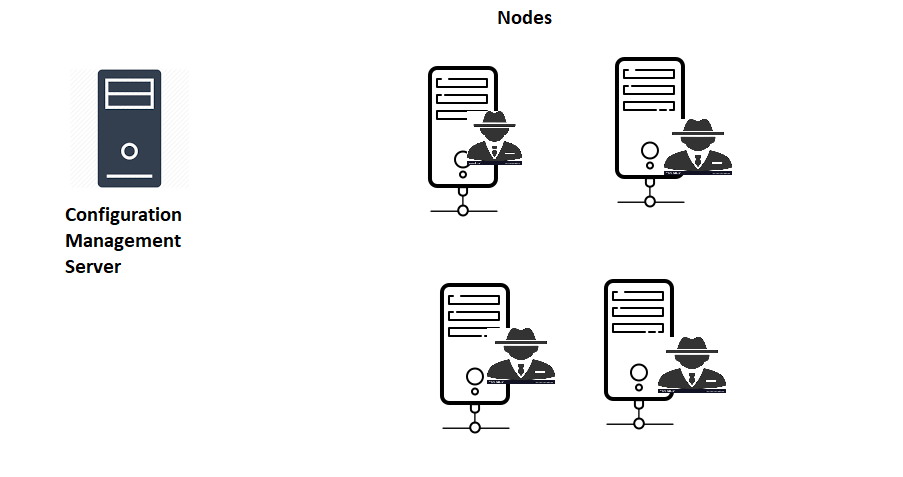
### Configuration Management Architecture

* Configuration Management Server:
  + This server has CM scripts all the necessary infrastructure.
* Nodes:
  + These are servers/machines where you want to apply configuration
* Models:
  + Push: In case of Push based CM, Server will initiate communications to the nodes
  + PULL: In this case, nodes will initiated communications with the server.



### PULL Based Configuration Management.

* Agent:
  + Agent has to installed on all the nodes.
  + Agent will have information about server
  + Agent will communicate with the Server
  + This agent needs to verified by server, we will have some security mechanism.
  + Once Agent is authorized, the conversation might look like
  + Do i have any work to be done

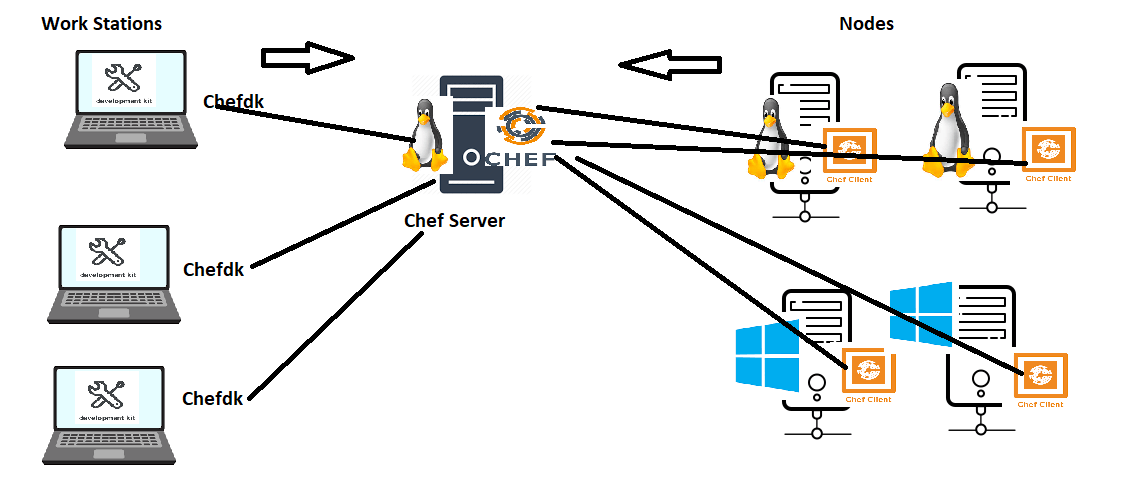


* Bootstrap:
  + Process of installing agent on node is called as Boot-straping
* Convergence:
  + Is the time (periodic) when agent communicates to the server and gets the CM done.
* Configuration Drift:
  + Deviation between desired state and actual state on the node.

**NOVEMBER 20, 2019**

## DevOps Classroom Series – Chef Architecture – 20/Nov/2019

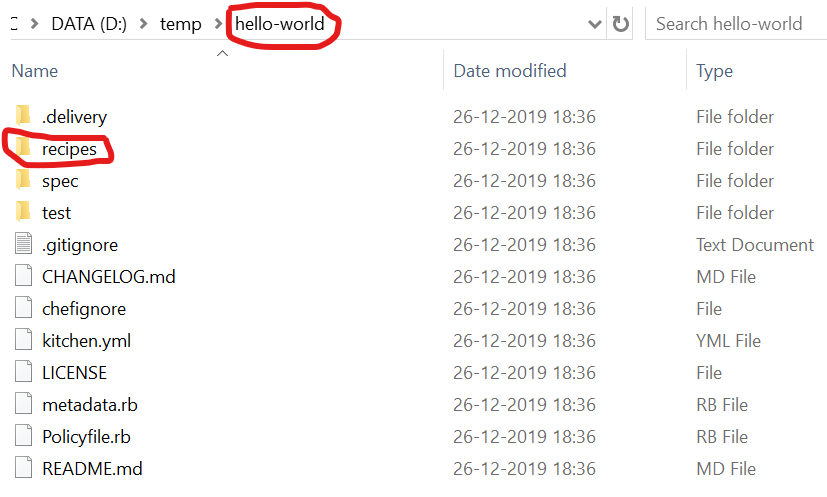
### Chef Architecture and Components



* Chef Server:
  + This is Ruby on Rails Application.
  + In this we have many components
    - Bookshelf
    - Message Queue
    - Postgres
  + We can also multiple Chef Servers for High Availability.
  + It stores
    - Reports on every execution
    - Collects the details about nodes
  + Server has to be Linux machine (RedHat, Ubuntu, SUSE etc…)
* Chef Client:
  + Agent running on Nodes.
  + Agent Executes Chef Cookbooks/recipes
  + Agent can be installed on Linux, Windows and Mac
* Chef Workstation:
  + Development tools installed on the Chef-Developers workstation

### Chef Terminology

* Cookbook:
  + Recipes organized in specific order
  + Cookbooks are organized in directory
* Recipe:
  + Resources organized in specific orders
  + Each recipe is a file.
* Resource:
  + Smallest unit to specify declarative syntax
  + Sample Cookbook structure



### Lab Setup

1. Chef Server:

Initially we will be using hosted chef Server. [Refer Here](https://manage.chef.io/login) (https://manage.chef.io/login)

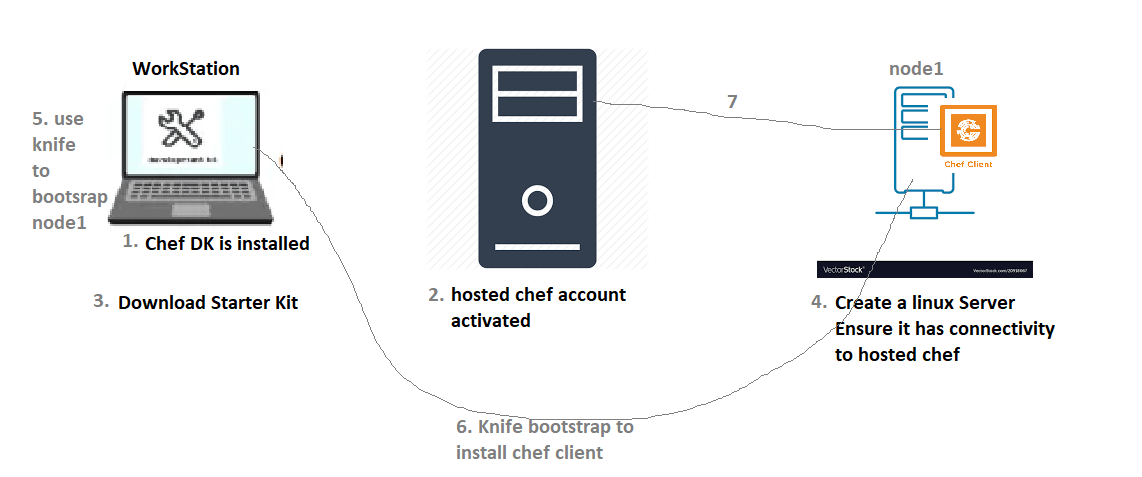
* + Create an account and then an organization.

1. Workstation:
   * Chocolatey:
     + Only on Windows
     + [Refer Here](https://chocolatey.org/docs/installation) (https://chocolatey.org/docs/installation)
   * Git Bash
     + Only on Windows
     + Launch Powershell and execute choco install git -y
     + If choco is not configured install git directly from [here](https://git-scm.com/download/win) (https://git-scm.com/download/win)
   * Chef DK
     + Can be installed from [here](https://downloads.chef.io/chefdk/) (https://downloads.chef.io/chefdk/)
     + If you prefer choco choco install chefdk -y
   * Visual Studio Code:
     + Download from [here](https://code.visualstudio.com/) (https://code.visualstudio.com/)
     + If you prefer choco choco install vscode -y
2. Node:
   * Create a Linux Machine(Ubuntu) in any Cloud(AWS).

**NOVEMBER 21, 2019**

## DevOps Classroom Series – 21/Nov/2019

### Bootstrap Preparation

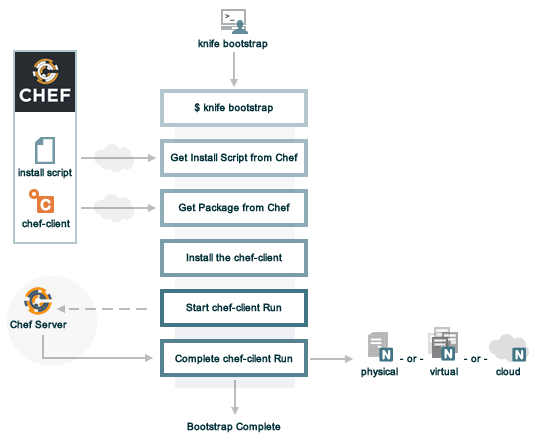
* Ensure Chefdk is installed on Workstation
* Login into Chef Server and Download Starter Kit (One time Job)
* Starter Kit consists of
  + Basic Folder Structure for Cookbooks, Roles and Environments
  + Authentication Mechanism between Workstation and Chef Server
  + Information about Server.
* Create a Linux Node on AWS/Azure/VirtualBox. In this series I will be using AWS. 

### Bootstrap Process

* Knife:
  + Utility which is installed as part of chefdk
  + Remember to run knife commands with your present working directory in Starter Kit
* Node Bootstrap:
  + [Refer Here](https://docs.chef.io/install_bootstrap.html) (<https://docs.chef.io/install_bootstrap.html>) for official chef docs.
  + In the current scenario, we will be using knife bootstrap

knife bootstrap <publicip> -U <username> -i <pathtoidentify> --sudo -N <fqdn>

* For bootstrap commandline options [Refer Here](https://docs.chef.io/knife_bootstrap.html) (<https://docs.chef.io/knife_bootstrap.html>)



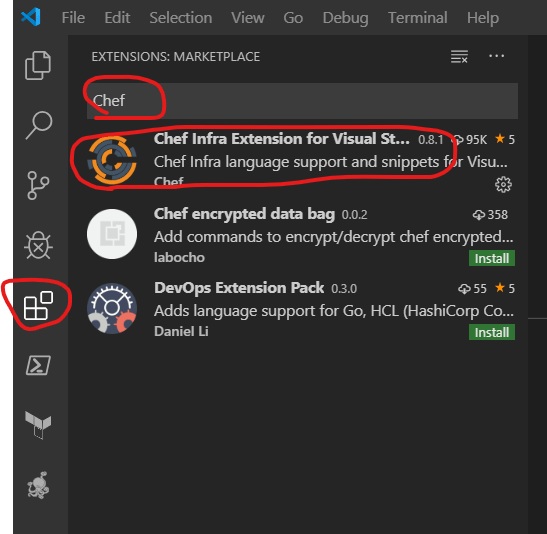
note: Referred from Chef documentation.

**NOVEMBER 22, 2019**

## DevOps Classroom Series – Chef – 22/Nov/19

### Visual Studio Code Configuration

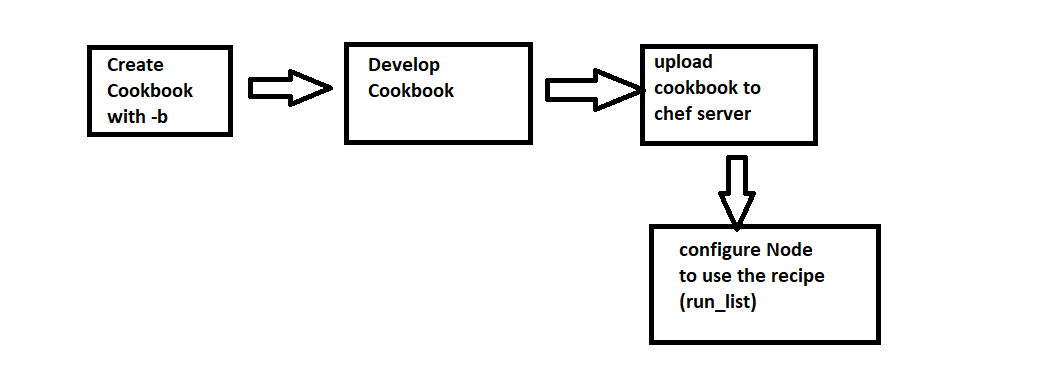
### Install Chef Extension into Visual Studio Code.



### Automating Deployments using Chef

* Ensure you have manual steps/commands for whatever you want to automate.
* Chef Organizes declarative syntaxes into
  + Cookbooks
    - Recipes
      * Resources

### Chefs Basic Workflow for cookbook automation



#### Scenario-1: Install Git and tree

1. Manual Execution (Ubuntu):

sudo apt-get update

sudo apt-get install git -y

sudo apt-get install tree -y

1. Manual Execution (RedHat):

sudo yum install git -y

sudo yum install tree -y

1. Creating a cookbook. For creating cookbook as part of chef dk installation, we get generators. Lets use cookbook generator

cd <starterkit-folder>\cookbooks\

chef generate --help

# since we want to generate cookbook

chef generate cookbook --help

# generate a cookbook called exercise1

chef generate cookbook exercise1 -b

1. Observe the folder structure of exercise1 and you should be able to see default.rb in recipes folder which is default recipe.
2. [Refer Here](https://docs.chef.io/resource_reference.html) (<https://docs.chef.io/resource_reference.html>) for Chef Resources
3. Basic Resource syntax is

<resource type> '<name of resource>' do

property1 value1

..

..

propertyn valuen

action <some actions>

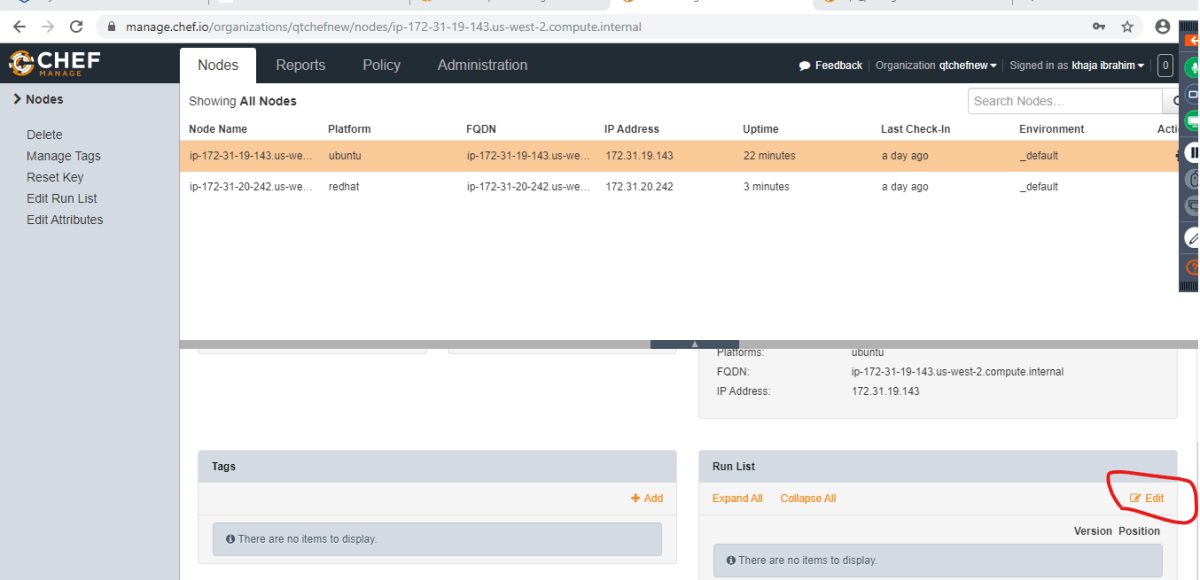
end

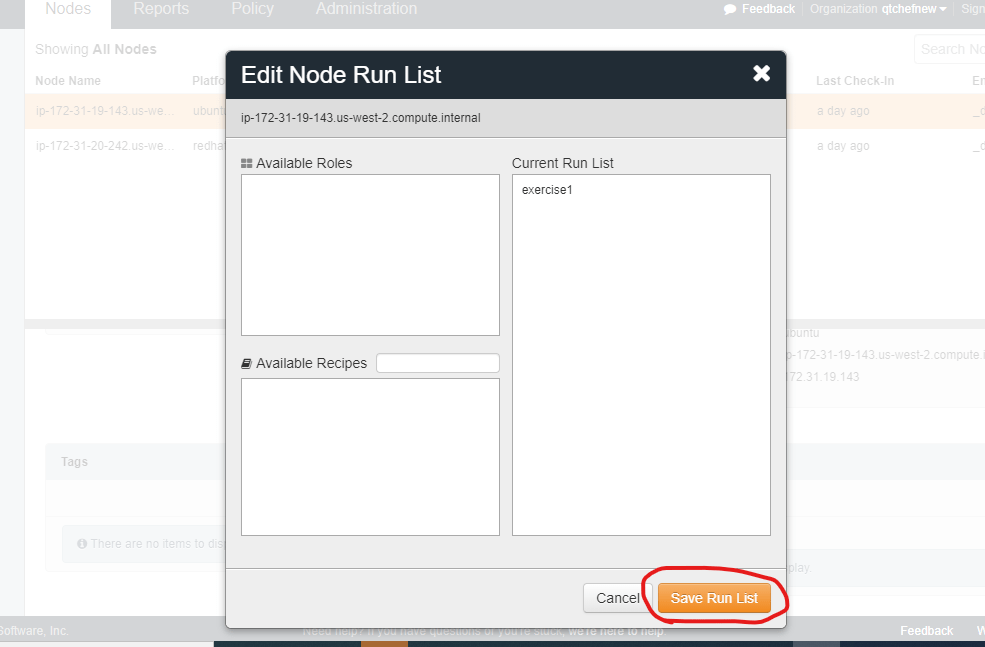
1. Upload the cookbook. Uploading the cookbook in this case will be done using a tool called berkshelf (berks).

cd <starterkit>\cookbooks\<cookbook>

berks install

berks upload

1. Applying cookbook to node. In chef every node will have list of recipes to be executed. This list is called as run\_list. Go to your node and change the run\_list and add your recipe. 



1. Once recipes are added to run list wait for Convergence. In this lets manually force Convergence. Login into node and execute chef-client

**NOVEMBER 24, 2019**

## DevOps Classroom Series – Chef – 24/Nov/2019

### Uploading Cookbooks to chef server

* Cookbooks can be uploaded to chef server using
  + Knife
    - Same cookbook can be uploaded multiple times and every time it overrites the existing cookbook
  + berkshelf
    - Allows cookbook with one version uploaded only once.

### Run List

* List of Recipes to be applied/executed on the node.
* This list mainitained by chef-server and during convergence passed to node.
* Each Recipe in the runlist has the following format

<cookbook-name>::<recipe-name>

## examples

exercise1::default

exercise1::redhat

* Just writing a cookbook name in runlist is equivalent to calling default recipe

exercise1

or

exercise1::default

### Developing a Cookbook for Java Application Deployment.

* Manual Steps in Ubuntu

sudo apt-get update

sudo apt-get install openjdk-8-jdk -y

wget https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/spring-petclinic.jar

java -jar spring-petclinic.jar

* Execute the following command chef generate cookbook -b springpetclinic
* Create a new recipe for java installation and updating ubuntu packages

chef generate recipe . java8

* [Refer Here](https://docs.chef.io/resource_package.html) (<https://docs.chef.io/resource_package.html>)  for package resource.
* Please refer below for java8 recipe

#

# Cookbook:: .

# Recipe:: java8

#

# Copyright:: 2019, The Authors, All Rights Reserved.

# updating ubuntu packages

apt\_update 'update' do

ignore\_failure true

action :update

end

package 'openjdk-8-jdk' do

action :install

end

* Now to download the spring petclinic and execute it lets create a new recipe called as spc with following content

#

# Cookbook:: .

# Recipe:: spc

#

# Copyright:: 2019, The Authors, All Rights Reserved.

remote\_file '/home/ubuntu/spring-petclinic.jar' do

source 'https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/spring-petclinic.jar'

action :create

end

## todo automation for java -jar spring-petclinic.jar

* Also change default.rb in recipes folder to call the recipes in the execution order (so that user can include default recipe in run list and whole cookbook executes)

#

# Cookbook:: springpetclinic

# Recipe:: default

#

# Copyright:: 2019, The Authors, All Rights Reserved.

include\_recipe 'springpetclinic::java8'

include\_recipe 'springpetclinic::spc'

* Upload the cookbook to chef server using berks upload
* Change the runlist of the ubuntu node and wait for convergence (or manually force it using sudo chef-client)

### Developing a cookbook for lamp stack

* [Refer Here](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04) (<https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04>)
* Manual Steps for ubuntu

sudo apt-get update

sudo apt-get install apache2 -y

sudo systemctl restart apache2

sudo apt-get install php libapache2-mod-php php-mysql php-cli -y

sudo systemctl restart apache2

sudo nano /var/www/html/info.php

<?php

phpinfo();

?>

sudo systemctl restart apache2

* Create a cookbook called as apachewithphp

chef generate cookbook -b apachewithphp

* Change metadata.rb file with your information

name 'apachewithphp'

maintainer 'DirectDevOps'

maintainer\_email 'qtdevops@gmail.com'

license 'All Rights Reserved'

description 'Installs/Configures apache and php'

long\_description 'Installs/Configures apachewithphp'

version '0.1.0'

chef\_version '>= 14.0'

* Create a recipe to install apache with php

chef generate recipe . apachephp

* Ensure your recipe has the following content in it

#

# Cookbook:: .

# Recipe:: apachephp

#

# Copyright:: 2019, The Authors, All Rights Reserved.

apt\_update 'update pacakges' do

ignore\_failure true

action :update

end

package 'apache2' do

action :install

end

service 'apache2' do

action [:enable, :restart]

end

php\_packages = ["php", "libapache2-mod-php", "php-mysql", "php-cli"]

php\_packages.each do |package\_name|

package package\_name do

action :install

end

end

service 'mypache2' do

service\_name 'apache2'

action :restart

end

remote\_file '/var/www/html/info.php' do

source 'https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/info.php'

action :create

end

service 'mypache2again' do

service\_name 'apache2'

action :restart

end

* Problems in this recipe so far are
  + service is appearing multiple times.
  + File info.php is downloaded but not created.
  + Service is restarted multiple times from second convergence.
* First lets try to resolve multipler services. For that we would use notifications in chef. [Refer Here](https://docs.chef.io/resource_common.html#notifications) (<https://docs.chef.io/resource_common.html#notifications>)  for more info.
* Using notifies updated recipe as shown below

#

# Cookbook:: .

# Recipe:: apachephp

#

# Copyright:: 2019, The Authors, All Rights Reserved.

apt\_update 'update pacakges' do

ignore\_failure true

action :update

end

package 'apache2' do

action :install

notifies :enable, 'service[apache2]'

end

php\_packages = ["php", "libapache2-mod-php", "php-mysql", "php-cli"]

php\_packages.each do |package\_name|

package package\_name do

action :install

end

end

remote\_file '/var/www/html/info.php' do

source 'https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/info.php'

action :create

notifies :restart, 'service[apache2]'

end

service 'apache2' do

action :nothing

end

* Change the version. Upload the cookbook using berks upload
* Now lets try to create a file info.php rather than downloading it.
* Generate a file in your cookbook using chef generate file . info.php and you can observe a folder called files gets created in your cookbook with a file info.php. In this file add the contents to info.php

<?php

phpinfo();

?>

* To copy this file to your node, use chef resource **cookbook\_file**. This changes the recipe as shown below

#

# Cookbook:: .

# Recipe:: apachephp

#

# Copyright:: 2019, The Authors, All Rights Reserved.

apt\_update 'update pacakges' do

ignore\_failure true

action :update

end

package 'apache2' do

action :install

notifies :enable, 'service[apache2]'

end

php\_packages = ["php", "libapache2-mod-php", "php-mysql", "php-cli"]

php\_packages.each do |package\_name|

package package\_name do

action :install

end

end

#remote\_file '/var/www/html/info.php' do

# source 'https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/info.php'

# action :create

# notifies :restart, 'service[apache2]'

#end

cookbook\_file '/var/www/html/info.php' do

source 'info.php'

action :create

notifies :restart, 'service[apache2]'

end

service 'apache2' do

action :nothing

end

* Change the version of the cookbook and upload the recipe and manually force convergence.
* The problems still open are
  + uploading multiple versions of under developed cookbooks to chef server.
* Next Steps:
  + Doing the developement for the same lampstack on centos
  + Avoiding uploading of under development cookbooks.
  + Uploading only stable versions to Chef Server.

**NOVEMBER 25, 2019**

## DevOps Classroom Series Chef – 25/Nov/2019

### Lamp on Centos 7

* Manual Steps for Centos 7

sudo yum install httpd -y

sudo systemctl enable httpd.service

sudo systemctl start httpd.service

sudo yum install php php-mysql

sudo systemctl restart httpd.service

sudo vi /var/www/html/info.php

<?php phpinfo(); ?>

* Reference from [here](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-centos-7) (<https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-centos-7>)
* Create a new recipe called as **httpdphp** for redhat flavor

chef generate recipe . httpdphp

* Recipe will be as shown below

#

# Cookbook:: .

# Recipe:: httpdphp

#

# Copyright:: 2019, The Authors, All Rights Reserved.

package 'httpd' do

action :install

notifies :enable, 'service[httpd]'

end

package\_names = ["php", "php-mysql"]

package\_names.each do |package\_name|

package package\_name do

action :install

end

end

cookbook\_file '/var/www/html/info.php' do

source 'info.php'

action :create

notifies :restart, 'service[httpd]'

end

service 'httpd' do

action :nothing

end

* If you compare recipes of redhat and ubuntu, they are almost similar so trying to write one recipe for both.
* This requires to understand couple of concepts in chef
  + Node Object
  + Attributes
* Node Object:
  + Information collected about node during convergence.
  + This information is collected by **Ohai**
  + Information Collected by ohai will be stored as Node Object.
  + Accessing Node objects can be done in recipes by using following syntax node[key][childkey1]
* Next Steps:
  + Use Node Objects and User Defined Attributes to write one recipe for lamp stack which runs on ubuntu and centos.

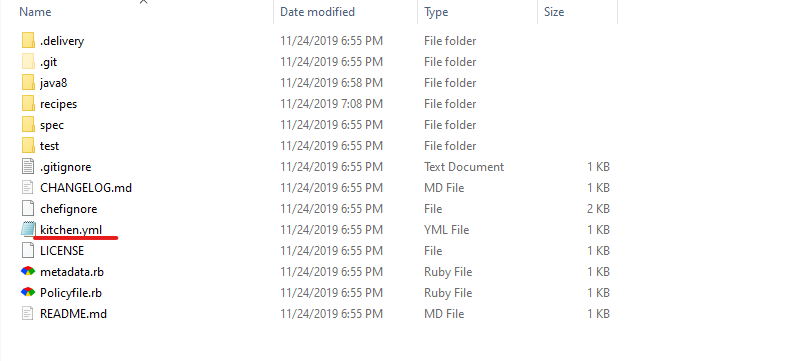
**NOVEMBER 28, 2019**

## DevOps Classroom Series 28/Nov/2019

### Problems with Current Approach of Developing Cookbooks

* Every time a correction is made to the cookbook, version is changed and new version is uploaded to chef server
* This leads to incorrect cookbooks for some versions.
* Solution is to upload only stable cookbook versions to chef server.
* The Answer to solve this is Test Kitchen.

### Test Kitchen

* This is testing environment for chef cookbook development.
* Workflow
  + Develop Cookbook
  + Test cookbook
  + Uplod cookbook to chef server
* Test Kitchen has drivers. Driver is the virtual environment where the servers are created. Driver Examples are
  + AWS
  + Azure
  + Docker
  + Vagrant
* In Test Kitchen,
  + a test vm gets created using the driver configure.
  + In this Test VM Chef Zero/Chef solo (Chef server and chef node on same machine)
  + Run list gets executed and logs, results are shared.
* Whenever you create a cookbook, kitchen.yml file is created. This is the configuration file for test kitchen. 

**In kitchen.yml file the default configuration will be using driver vagrant.**

### Test Kitchen Setup for AWS Test Environments

* To configure Test Kitchen for AWS [Refer Here](https://directdevops.blog/2019/04/02/test-kitchen-setup-with-aws/) <https://directdevops.blog/2019/04/02/test-kitchen-setup-with-aws/>

**NOVEMBER 29, 2019**

## DevOps Classroom 29/Nov/2019

### Kitchen Docs

* [Refer Here](https://kitchen.ci/) (<https://kitchen.ci/>)

### Test Kitchen on Azure

* Install azure cli [from here](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest) (<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>)
* Launch any terminal (Powershell) and execute

az login

* Generate a cookbook and cd into that.
* Install chef gem using command

chef gem install kitchen-azurerm

* [Refer Here](https://directdevops.blog/2019/04/03/test-kitchen-setup-with-azure/)(<https://directdevops.blog/2019/04/03/test-kitchen-setup-with-azure/>)

### Accessing Node Object in Recipes

* Node object is available inside recipes as **node**.
* Type of node is ruby hash

**DECEMBER 1, 2019**

## DevOps Classroom Series – 01/Dec/2019

### Attributes

* Attributes are user variables
* Attributes can be defined at multiple places
  + Recipe
  + Attribute File
  + Roles
  + Environments
* Attributes also have types
  + default
  + force\_default
  + normal
  + overrride
  + force\_override
  + automatic

### Scenario: Lamp

* As of now we have lamp stack cookbook with one recipe for ubuntu and one recipe for redhat.
* Lets try to write one recipe for both.
* Lets generate the attribute

chef generate attribute --help

chef generate attribute . default

* In the default.rb file generated in attributes folder, add the following

default[<cookbookname>][<variable-name>] = <variable-value>

default['apachewithphp']['package\_name'] = 'apache2'

* default.rb with all the attributes defined will be as shown below

if node['platform'] == 'ubuntu'

# package name for apache server for ubuntu machines

default['apachewithphp']['package\_name'] = 'apache2'

# php packages for ubuntu machines

default['apachewithphp']['php\_packages'] = ["php", "libapache2-mod-php", "php-mysql", "php-cli"]

else

# package name for apache server for redhat machines

default['apachewithphp']['package\_name'] = 'httpd'

# php packages for ubuntu machines

default['apachewithphp']['php\_packages'] = ["php", "php-mysql"]

end

* Using the attribute defined in attributes file can be acheived by using **node** object. syntax is

# attribute definition

default[<cookbookname>][<variable-name>] = <variable-value>

#attribute usage

node[<cookbookname>][<variable-name>]

* If we apply the above attributes then recipe will be as shown below

#

# Cookbook:: .

# Recipe:: phpapache

#

# Copyright:: 2019, The Authors, All Rights Reserved.

if node['platform'] == 'ubuntu'

apt\_update 'update pacakges' do

ignore\_failure true

action :update

end

end

package\_name = node['apachewithphp']['package\_name']

package package\_name do

action :install

notifies :enable, "service[#{package\_name}]"

end

php\_packages = node['apachewithphp']['php\_packages']

php\_packages.each do |package\_name|

package package\_name do

action :install

end

end

#remote\_file '/var/www/html/info.php' do

# source 'https://qt-s3-new-testing.s3-us-west-2.amazonaws.com/info.php'

# action :create

# notifies :restart, "service[#{mypackage}]"

#end

cookbook\_file '/var/www/html/info.php' do

source 'info.php'

action :create

notifies :restart, "service[#{mypackage}]"

end

service package\_name do

action :nothing

end

* Lets apply **only\_if** gaurd as mentioned over [here](https://docs.chef.io/resource_common.html#only-if-examples) (<https://docs.chef.io/resource_common.html#only-if-examples>)
* Also look at **not\_if** over [here](https://docs.chef.io/resource_common.html#not-if-examples) (<https://docs.chef.io/resource_common.html#not-if-examples>)
* Lets add the logic to fail the recipe execution if it is executed on unsupported platforms. The following code is added to default.rb in recipes folder.

if node['platform'] == 'ubuntu' or node['platform'] == 'centos'

include\_recipe 'apachewithphp::phpapache'

else

raise 'This cookbook is supported only on ubuntu and centos'

end

* Try testing this cookbook by changing kitchen.yaml file for ubuntu and centos flavloure
* Now change the version in the metadata.rb file and upload the cookbook using berks upload.

### Making Cookbooks Reusable

* There will be lot of cookbooks already written, rather than rewriting cookbooks, it easy to reuse.
* Ways of reusing cookbooks
  + Reusing recipes
  + Making your cookbooks reusable by writing custom resources.
* Supermarket:
  + Community cookbooks with reusable recipes or reusable custom resources.

### Chef Supermarket

* Lets create a cookbook to install java and postgresql on ubuntu machine and centos using chef supermarket cookbooks.
* Lets create a cookbook called supermarketdemo
* Now lets find a supermarket cookbook for postgres. Result is over [here](https://supermarket.chef.io/cookbooks/postgresql#berkshelf) (<https://supermarket.chef.io/cookbooks/postgresql#berkshelf>)
* To use this cookbook we need to define dependency. Dependency is defined in metadata.rb file using the following syntax

depends '<cookbookname>', '<symbol> <version>'

* Symbols in versions define restrictions on versions
  + <
  + >
  + <=
  + =
  + ~>
* berks install command will download all the dependencies of your cookbook defined in metadata.rb
* Now lets try to install postgres server by reading the documentation of cookbook, the way is as following. use the custom resource.

postgresql\_server\_install 'Setup my PostgreSQL 9.6 server' do

password 'MyP4ssw0rd'

port 5433

action :create

end

* Try to execute this cookbook located at [here](https://github.com/asquarezone/ChefZone/tree/master/Exercise/Supermarket) (<https://github.com/asquarezone/ChefZone/tree/master/Exercise/Supermarket>)

**DECEMBER 2, 2019**

## DevOps Classroom Series – 02/Dec/2019

### Chef-Client Supermarket cookbook

* Using this cookbook from supermarket
  + Convergence can be changed
  + Scheduled jobs can be created
  + Chef-Client log locations can be changed.

### Changing default convergance of a node to 2 hours.

* search for chef-client cookbook in Chef Supermarket. [Refer Here](https://supermarket.chef.io/cookbooks/chef-client) (<https://supermarket.chef.io/cookbooks/chef-client>)
* Create a cookbook called as changecovnergence.
* Add depends 'chef-client', '~> 11.4.0' to metadata.rb
* Execute berks install
* create a attribute file called as default using chef generate attribute . default
* In the attributes/default.rb add the following line

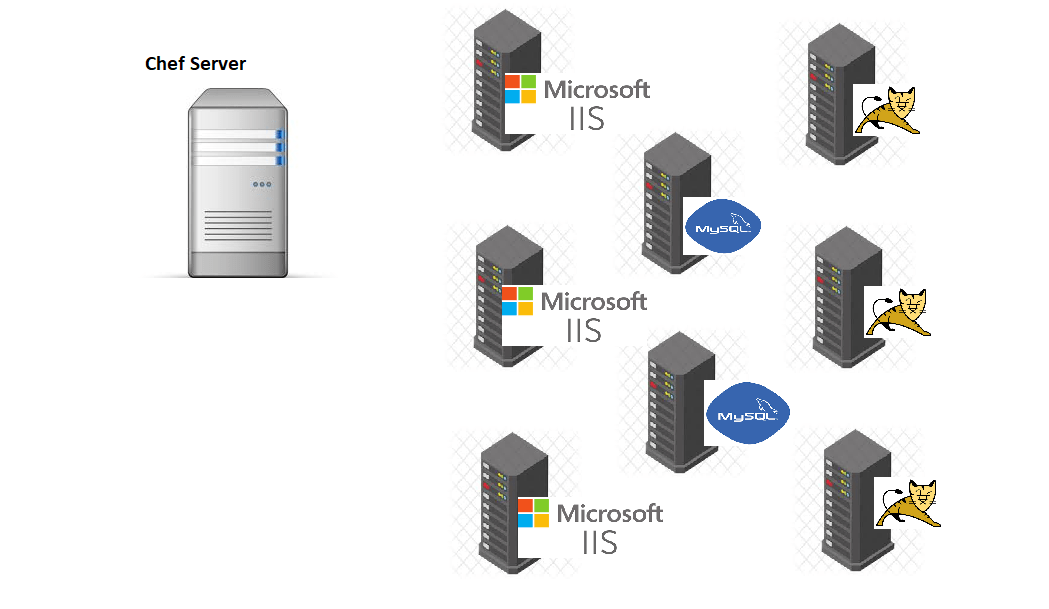
default['chef\_client']['interval'] = 7200

* In recipes/default.rb add the following line

include\_recipe 'chef-client::default'

* Now upload this cookbook to chef server using berks upload and add this recipe to run\_list of any node will change convergence from 30 minutes to 2 hours.

### Scenario:

* The Servers look as shown below 
* How do you design this deployment using chef
* Create 3 cookbooks
  + Mysql
  + tomcat
  + IIS Server
* use kitchen to verify cookbooks are working
* upload the cookbooks to chef server
* Ensure all the 8 servers are bootstrapped.
* Change the run\_list of each server and add the necessary recipe or recipe list

#### Pain points

* Bootstrapping multiple servers
* Changing run list of every server

#### Change in scenario.

* Lets assume you are asked to install git on every node.
* For this we need to change the run list of every node.

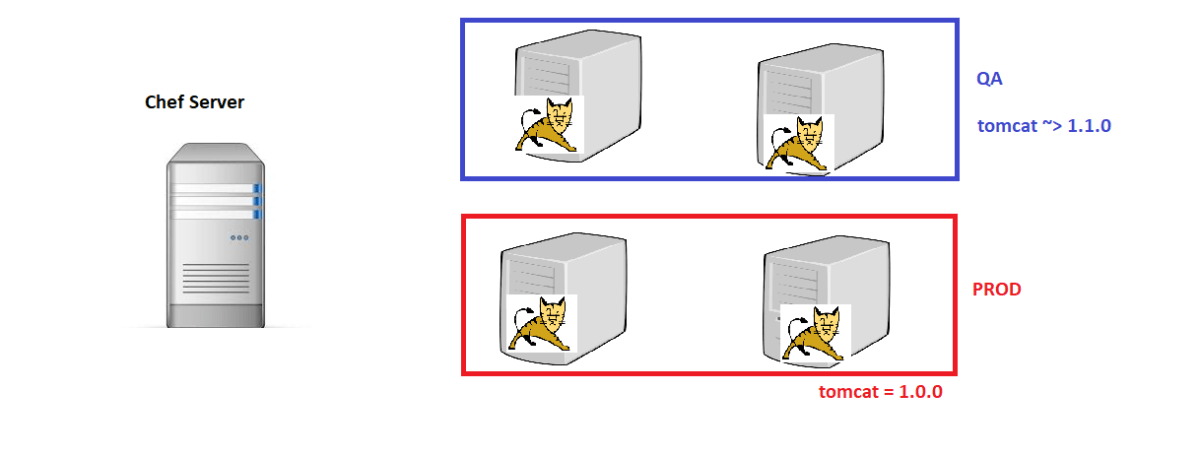
#### Lets try to minimize efforts in Changing run\_lists

* Chef has a concept called as role.
* Role is nothing but list of recipes.
* We create 3 roles
  + mysql-role => mysql::default
  + tomcat-role => tomcat::default
  + iis-role => iis::default
* In all the server where i need mysql to be installed in run\_list i will add mysql-role
* In all the server where i need tomcat to be installed in run\_list i will add tomcat-role
* In all the server where i need iis to be installed in run\_list i will add iis-role
* If i need to add git to every node, there is no need to change run\_list just edit the roles.
  + mysql-role => mysql::default, git::default
  + tomcat-role => tomcat::default, git::default
  + iis-role => iis::default, git::default
* Its always a best practice to have roles like
  + db-role
  + app-role
  + web-role

**DECEMBER 3, 2019**

## DevOps Classroom Series – 03/Dec/2019

### Scenario

* Important info:
  + By default chef always picks the latest version of cookbook 
* All the nodes are deployed using
  + tomcat cookbook
    - version 1.0.0
    - java => 8
* Now you have been asked to change the following
  + JDK8 to JDK9
  + tomcat7 to tomcat8
* Ideally you should be making changes to QA Servers first and then to Production
* Now lets assume you have changed tomcat cookbook to install tomcat8 and jdk9 with version 1.1.0
* During next convergance all the servers will get update with 1.1.0 which can be a disaster
* For this in chef we have concept called as Environments
* Environment impose restrictions on cookbook versions.
* It is necessary that every node belongs to one environment. The default environment is \_default.
* To solve the above problem
  + Create a new environment called as QA
    - tomcat = 1.1.0
  + Create a new environment called as Prod
    - tomcat = 1.0.0 

### Sample with Environments and Roles

* Create a simple cookbook with name envroledemo
* In the first version 1.0.0 write some resources. In this series i will be using file resource to create a file with some content
* Recipe => java in 1.0.0

file '/home/ubuntu/java' do

content 'jdk8'

action :create

end

* Recipe => java in 1.1.0

file '/home/ubuntu/java' do

content 'jdk9'

action :create

end

* Creating Roles: Roles can be created
  + On Chef Manage UI
  + From ruby or json files in Starter Kit.
* [Refer Here](https://docs.chef.io/roles.html#role-formats) (<https://docs.chef.io/roles.html#role-formats>) for role formats
* In this series i prefer ruby files. I have create a tomcat.rb file in roles folder with following content

name "tomcat"

description "tomcat role"

run\_list "recipe[envroledemo::java]"

**DECEMBER 4, 2019**

## DevOps Classroom Series – 04/Dec/2019

### Environment File

* In the chef-repo file create folder called as environments
* In the environments create two files called as qa.rb and uat.rb
* qa.rb contents are

name "qa"

description "This is the qa version"

cookbook\_versions({

'envroledemo'=>'= 1.1.0'

})

* uat.rb contents are

name "uat"

description "This is the uat version"

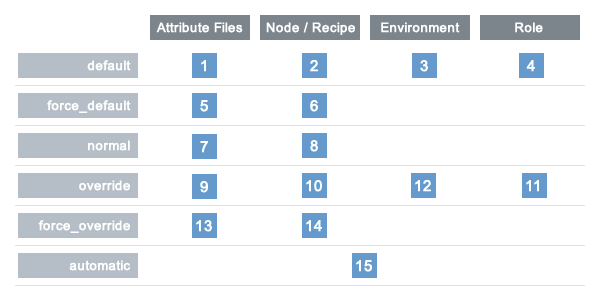
cookbook\_versions({

'envroledemo'=>'= 1.0.0'

})

* Upload environments to chef server using knife environment from file <filename>
* Apply environment to nodes

### Attribute Precedence:

* Attributes can be defined in
  + recipe
  + Attributes files
  + role
  + environment
* Each Attribute has type.
  + default
  + force\_default
  + normal
  + override
  + force\_override
* Chef defines priority for each position in the combination of Place of Attribute definition and Attribute Type 
* [Refer](https://directdevops.blog/wp-content/uploads/2019/01/Attributes.pdf) (<https://directdevops.blog/wp-content/uploads/2019/01/Attributes.pdf>)

### Chef Templates

* Templates are used for configuration files.
* Consider this scenario.
* Install apache server. Create a file called as index.html in /var/www/html/index.html with following content

Hello ,

Operating System is <os>

ip address is <ipadress>

* Generate cookbook

chef generate cookbook -b templatedemo

* Now generate a template

chef generate template . index.html

* A new folder called as templates with index.html.erb file gets created.
* In this file add the following content

Hi,

Operating System is <%= node['platform'] %>

Ip address is <%= node['ipaddress'] %>

* In the recipes\default.rb file add the template resource

template 'home/ubuntu/index.html' do

source 'index.html.erb'

action :create

end

* upload and test this cookbook in kitchen to find the dynamic values in the index.html on the node.

**DECEMBER 5, 2019**

## DevOps Classroom Series – 05/Dec/2019

### Chef data bags

* Is to encrypt sensitive information.
* [Refer Here](https://docs.chef.io/data_bags.html) (<https://docs.chef.io/data_bags.html>)
* Creating databag manually [refer here](https://docs.chef.io/data_bags.html#manually) (<https://docs.chef.io/data_bags.html#manually>)
* In the chef-repo folder

mkdir data\_bags

cd data\_bags

mkdir admins # admins is data bag name

knife data bag create admins # This command creates the data bag on the server

* Create a file called as mysql.json in admins folder with following content

{

"id": "mysql",

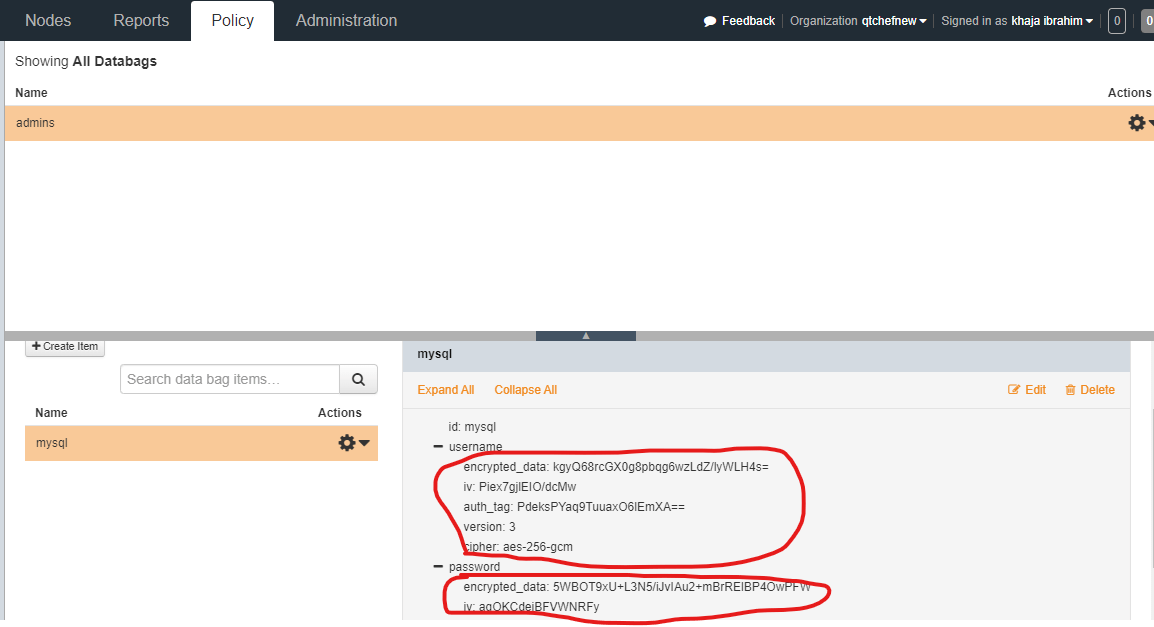
"username": "root",

"password": "rootroot"

}

* Data bag is used to encrypt the content and while reading from recipe it will decrypt
* Upload the file with secret text or secret file, so that the data is encrypted on the server

knife data bag from file admins .\mysql.json --secret <yoursecret>

* Now look at data bag item in Chef management Console 
* To encrypt the local file in your cook book use the following comamnd

knife data bag from file admins .\mysql.json --secret <yoursecret> --localmode

* When the user wants to access the data from data bag inside recipe or templates use the following syntax

mysqldatabag\_item = data\_bag\_item('admins','mysql', '<secret>')

username = mysqldatabag\_item['username']

password = mysqldatabag\_item['password']

### Chef Unattended bootstrap

* [Refer Here](https://docs.chef.io/install_bootstrap.html#unattended-installs) (<https://docs.chef.io/install_bootstrap.html#unattended-installs>)
* If you are using Azure VMs [Refer Here](https://docs.microsoft.com/en-us/azure/virtual-machines/extensions/chefx) (<https://docs.microsoft.com/en-us/azure/virtual-machines/extensions/chefx>)

### Chef Server Installation

* [Refer Here](https://docs.chef.io/install_server.html) (<https://docs.chef.io/install_server.html>)

### Exercise Spring Boot application (Spring-petclinic) as a Service

* Steps for installing java and downloading spring petclinic

sudo apt-get update

sudo apt-get install openjdk-8-jdk -y

sudo wget https://learningspcfromqt.s3.us-east-2.amazonaws.com/spring-petclinic.jar

* Write a script to run this application. (This script will run continuously as long as application is running)

sudo vi my-webapp.sh

#!/bin/bash

java -jar spring-petclinic.jar

* To avoid this long running script, make spring-petclinic a linux daemon(service).
* sudo vi /etc/systemd/system/springpetclinic.service

[Unit]

Description=My Webapp

[Service]

User=ubuntu

#change this to your workspace

WorkingDirectory=/home/ubuntu

#path to executable.

#executable is a bash script which calls jar file

ExecStart=/bin/bash /home/ubuntu/my-webapp.sh

SuccessExitStatus=143

TimeoutStopSec=10

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

* To reload the deamon “`sudo systemctl daemon-reload“
* enable the service for every boot “`sudo systemctl enable springpetclinic.service“
* start the service sudo systemctl start springpetclinic
* check the application is running or not sudo systemctl status springpetclinic
* For centos the create a file /etc/systemd/system/springpetclinic.service

[Unit]

Description=My Webapp

[Service]

User=centos

#The configuration file application.properties should be here:

#change this to your workspace

WorkingDirectory=/home/centos

#path to executable

#executable is a bash script which calls jar file

ExecStart=/bin/bash /home/centos/my-webapp.sh

SuccessExitStatus=143

TimeoutStopSec=10

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

**DECEMBER 7, 2019**

## DevOps Classroom Series – 07/Dec/2019

### Chef Server Installation

* Create a t2.large ubuntu machine in AWS.
* Installation Steps are referred from [here](https://docs.chef.io/install_server.html)(<https://docs.chef.io/install_server.html>)
* The current mode of installation is Standalone. Version used in this documentation is **13.1.13**
* Package used is downloaded from https://packages.chef.io/files/stable/chef-server/13.1.13/ubuntu/18.04/chef-server-core\_13.1.13-1\_amd64.deb
* Login into ec2 instance.

cd /tmp

wget https://packages.chef.io/files/stable/chef-server/13.1.13/ubuntu/18.04/chef-server-core\_13.1.13-1\_amd64.deb

sudo dpkg -i /tmp/chef-server-core\_13.1.13-1\_amd64.deb

sudo chef-server-ctl reconfigure

* Create an administrator and organization. In this series the following will be the assumed info

USERNAME shaikkhajaibrahim

FIRSTNAME khaja

LASTNAME ibrahim

EMAIL qtdevops@gmail.com

PASSWORD 'india@123'

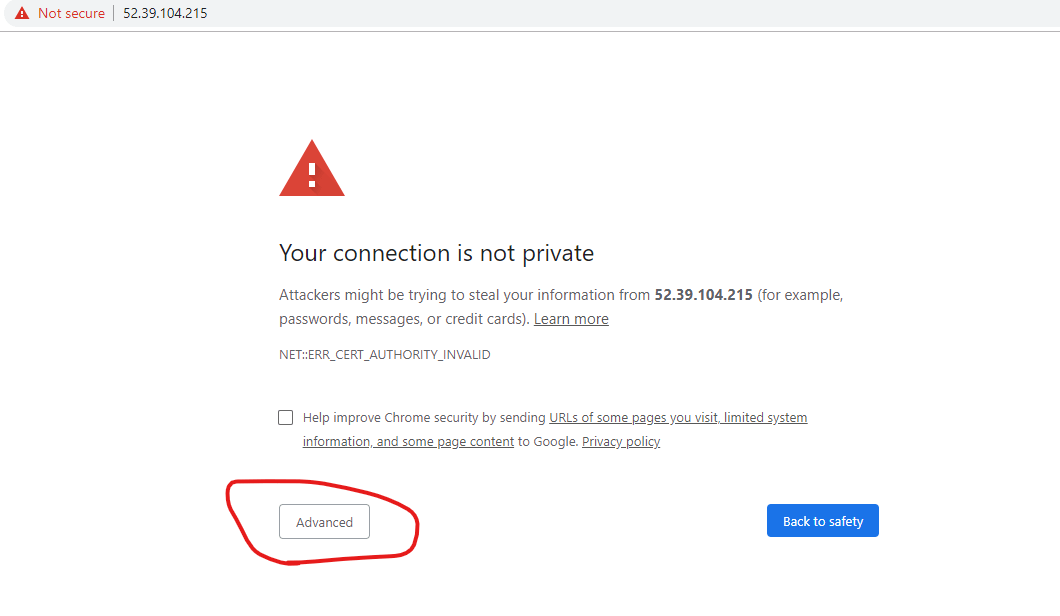
ORG NAME DIRECT DEVOPS

* The command to create user is sudo chef-server-ctl user-create USER\_NAME FIRST\_NAME LAST\_NAME EMAIL 'PASSWORD' --filename FILE\_NAME and now i fill it with above info

sudo chef-server-ctl user-create shaikkhajaibrahim khaja ibrahim qtdevops@gmail.com 'india@123' --filename /home/ubuntu/shaikkhajaibrahim.pem

* To create an organization command is “` sudo chef-server-ctl org-create short\_name ‘full\_organization\_name’ –association\_user user\_name –filename ORGANIZATION-validator.pem“ and now lets fill with above info

sudo chef-server-ctl org-create directdevops 'DIRECT DEVOPS' --association\_user shaikkhajaibrahim --filename /home/ubuntu/directdevops-validator.pem

* Access the url https://<publicip>. On error click on advanced an go to <public ip> 
* Log in to the Ubuntu machine and execute the following commands

sudo chef-server-ctl install chef-manage

sudo chef-server-ctl reconfigure

sudo chef-manage-ctl reconfigure

* Once this is done, chef server is exactly like hosted chef. in this case whenever you are executing knife commands do add –node-ssl-verify-mode none during bootstrap. you might need add ssl exceptions
* Two other addons can be added

sudo chef-server-ctl install opscode-reporting

sudo chef-server-ctl install opscode-push-jobs-server

### Chef Points for Professional Summary & Roles

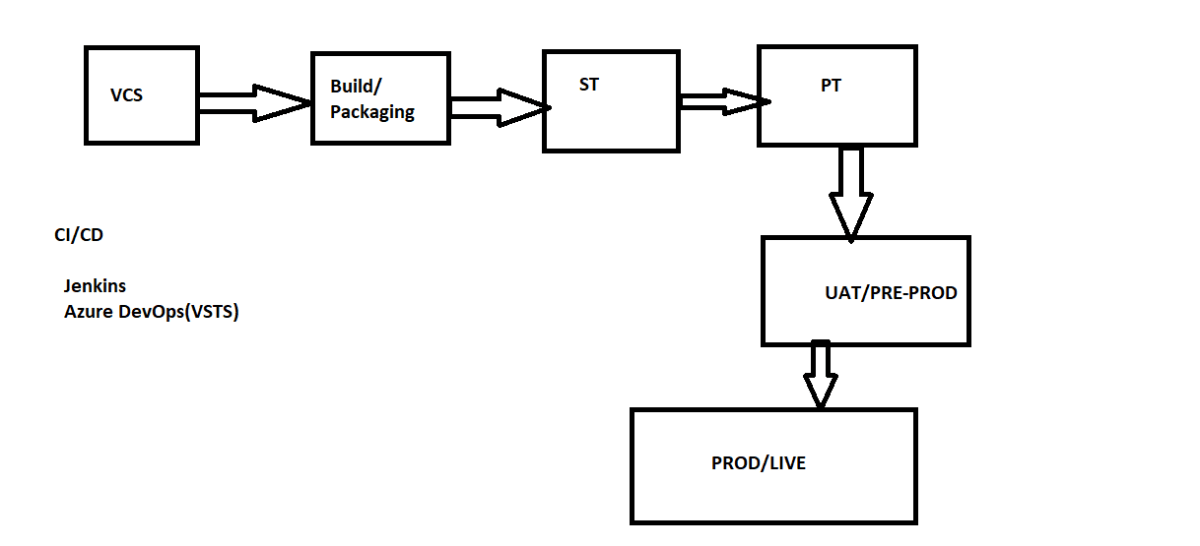
* Hands on Experience in Developing and testing chef cookbooks
* Deployed configurations to various environmets using Chef.
* Manged multiple nodes and frequent changes using chef
* Created reusable chef cookbooks and hands on experience in using and fine-tuning chef supermarket cookbooks
* Use chef as a provisioner in Terraform and Packer.
* Use Chef Extension on Azure VMs to deploy Configurations.

# GIT

**DECEMBER 10, 2019**

## DevOps Classroom Series – 10/Dec/2019

### Sample Pipeline

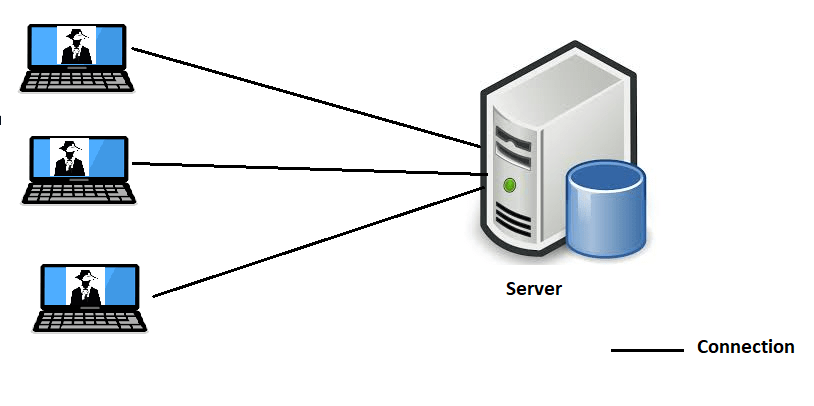


### Requirements for system to store code

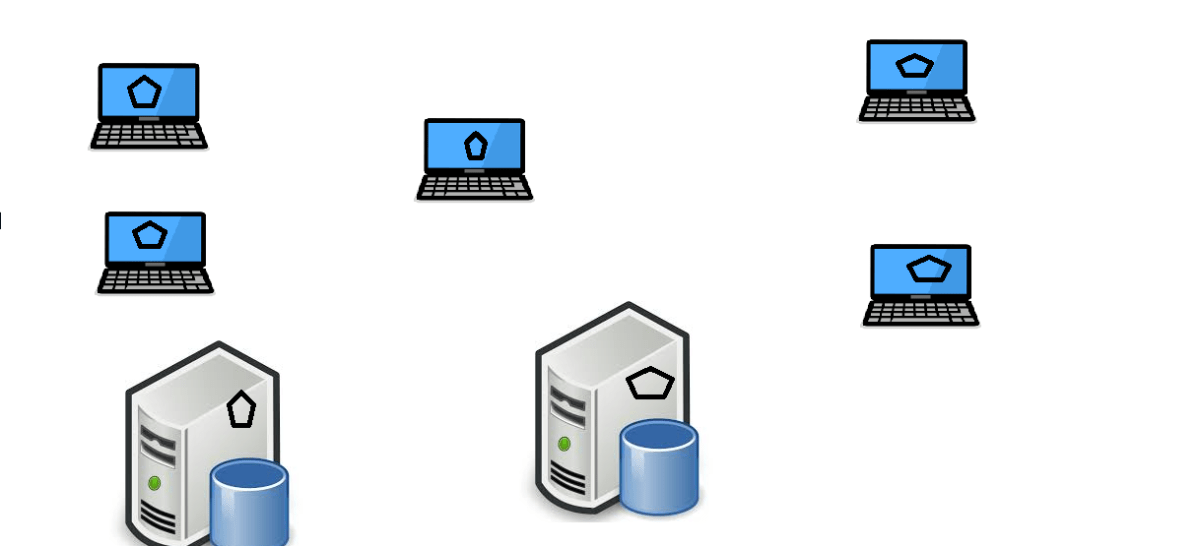
* Common Server/Application/System to store the code from muliple developers
* Version should be maintained for every change.
* Multiple users should be allowed to work on same files/code
* Any system which satisfies above three points is a Version Control System
* VCS:
  + CVS
  + VSS
  + SVN
  + ClearCase
  + GIT

### Architectures of VCS

* Client-Server Architecture:
  + Connected:
    - Clients should be connected to servers all the time
    - Administrators are required to take backups and also to replicate to multi-sites
    - Network connectivity issues could disrupt developers in doing the work



* + Dis-Connected:
    - Clients needed to be connected to servers only for getting the latest code or to submit work. Clients can work in offline mode
    - Administrators are required to take backups and also to replicate to multi-sites
* Distributed Architecture:
  + Whole copy of code is present on every node (client).
  + Backup of the Code is one more node
  + Other site is one more node
  + Git is a Distributed Version Control System.



**DECEMBER 11, 2019**

## DevOps Classroom Series – Git – 11/Dec/2019

### Installing Git

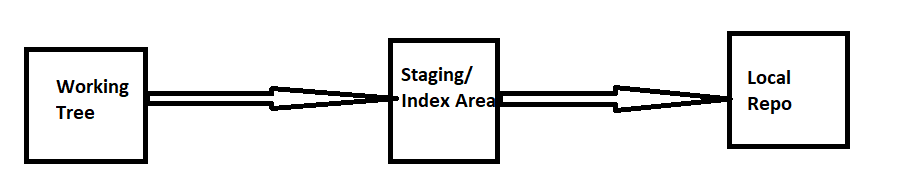
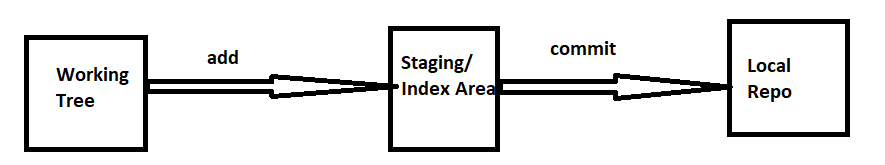
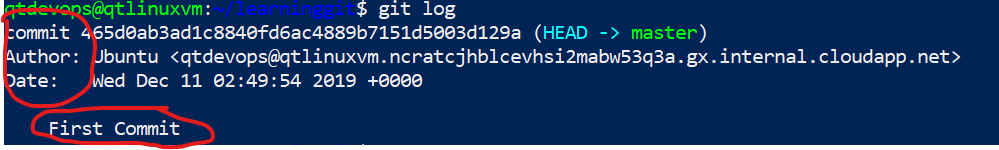
* Windows:
  + Download software from [here](https://gitforwindows.org/) (<https://gitforwindows.org/>)
  + Chocolatey:
    - Install from [here](https://chocolatey.org/install) (<https://chocolatey.org/install>)
    - Open the Powershell and execute choco install git
* Linux:
  + Ubuntu: sudo apt-get update && sudo apt-get install git -y
  + Redhat: sudo yum install git -y
* Mac:
  + Install homebrew and xcode

### Git Operations on Node (Local)

* Lets create a git Repository
* Create a folder and cd into it

mkdir learninggit

cd learninggit

* Mark this folder into repository by executing git init
* In Git we have 3 logical areas
  + Working Tree
  + Staging Area/Index Area
  + Local Repo / git database
* Local Repo Workflow
  + Make changes in Working Tree
  + Add the changes to Staging Area
  + Save the changes from Staging Area to Local Repo. 
* Lets try this WF
  + Create two file 1.txt and 2.txt
  + Execute the command git status
  + Now lets add the changes to staging area
* git add 1.txt
* git status
* git add 2.txt
* git status
  + Now lets try to commit the change from staging Area to Local Repo. To do this we can use the command git commit.
  + This command requires 3 details
    - Username
    - Email
    - Message
  + Username and Email can be configured once at the system level, but message has to passed for every commit.
  + To configure email and username
* git config --global user.name "<yourusername>"
* git config --global user.email "<youremailid>"
  + To commit the changes now execute git commit -m "First Commit" 
  + To check the history, execute the following command git log
  + For every change you get unique commit id as shown below 
  + Create a new folder called as docs
  + Create one more folder called as test
  + In the test folder create a new file called as 3.txt 
  + Now Execute git status and you will observed docs folder information is not shown in git status. Git never identifies empty directories.
  + Now add the change to staging area git add test/
  + Now execute the following commands and observe the ouptut info
* git commit -m "Second Commit"
* git status
* git log

### Popular Cheatsheets

* [Git Hub](https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf) (<https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf>)
* [Atlassian](https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet) (<https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet>)

**DECEMBER 12, 2019**

## DevOps Classroom Series – Git – 12/Dec/2019

### Git Operations

* Note: [Refer](https://directdevops.blog/2019/12/11/devops-classroom-series-git-11-dec-2019/) before you start this article.

### Untracked file and modified file

* Execute the following commands. Open Git-Bash or Linux/Mac Terminal

touch 4.txt

echo "Hello" >> 1.txt

* The above commands create a new file which is not in local repository and modifies a file which is already in local repository
* Now Execute git status and the following output will be shown

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: 1.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

4.txt

no changes added to commit (use "git add" and/or "git commit -a")

* Untracked file is a new file which has no history in local repo. Modified file is existing file which has history in local repo.
* Add only modified file to staging area

git add --help

git add -u

git status

* Now add the untracked file to staging area and commit the changes to local repo.

git add 4.txt

git commit -m "Third Commit"

git log

git status

### Handling multiple Changes

* Execute the following commands

touch docs/1.txt

touch docs/2.txt

touch test/5.txt

touch test/4.txt

touch 5.txt

touch 6.txt

echo "hello" >> 2.txt

echo "hello" >> test/3.txt

* If you want add all the changes at once to staging Area

git status

git add -A

# or if you are in top folder on git repo

git add .

git status

git commit -m "Fourth Commit"

### ignoring some directories and files

* Execute the following commands

mkdir bin

touch bin\10.txt

touch 7.txt

echo "hello" >> 6.txt

* We want everything in bin folder to be ignored. Now execute git status and the following o/p will be shown

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: 6.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

7.txt

bin/

no changes added to commit (use "git add" and/or "git commit -a")

* Git has a way to ignore certain files/foders. For that create a file called as .gitignore in the repository root.

touch .gitignore

* Now add bin/\* to the .gitignore file and execute the following

git status # no bin/ information should be shown

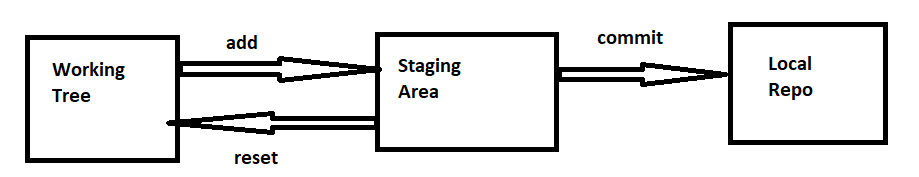
git add .

git commit -m "Fifth Commit"

git status

git log

### Moving Changes from Staging Area to Working Tree

* Reset command can help in doing this 
* Execute the following command

touch 8.txt

touch 9.txt

touch 10.txt

echo "hello" >> 7.txt

git status

git add .

git status

* Now move 10.txt from staging area to working tree

git reset 10.txt

git status

* Now if you want to remove all the changes in staging area and modified changes in working tree

git reset --hard

git status

* To remove all the untracked files from working tree

git clean -fd .

### Removing files from local repo.

* Execute the following command

rm 7.txt

* Execute the following commands

git status

git add .

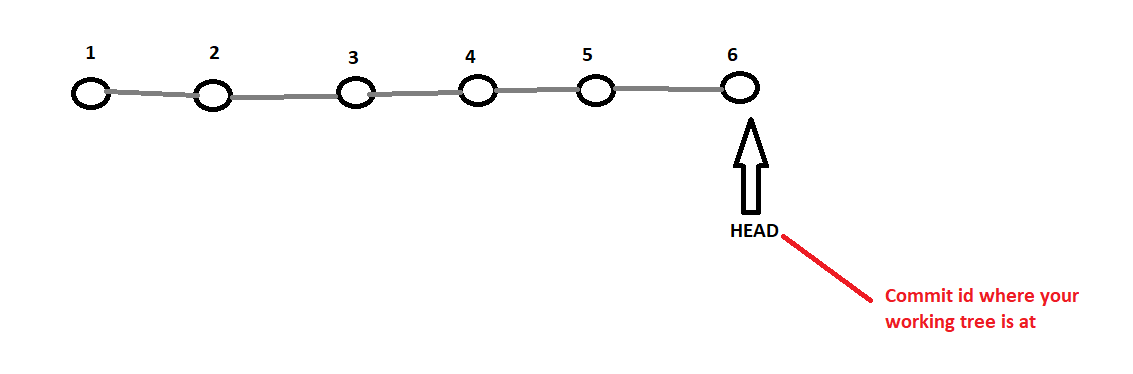
git commit -m "Sixth Commit"

* You can also use git rm. [Refer Here](https://www.atlassian.com/git/tutorials/undoing-changes/git-rm) (<https://www.atlassian.com/git/tutorials/undoing-changes/git-rm>)

**DECEMBER 13, 2019**

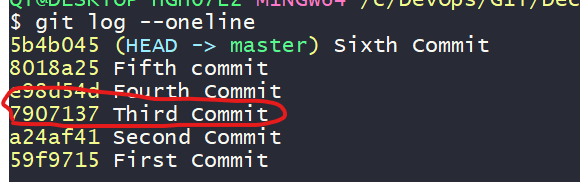
## DevOps Classroom Series – 13/Dec/2019

### Git History

* Every Change moved from staging Area to Local Repo is called as COMMIT.
* Every Commit has an ID
* using GIT you can move to any commit in history.
* Lets Define **HEAD**. HEAD is pointer which points to commit id working tree is looking at. 
* Lets try to move the HEAD to Third commit. For that you need commit id. Commit id which can be used are long and short

git log

git log --oneline



* Now Lets move the HEAD to Third Commit

git checkout 7907137

* Now to move the HEAD back to latest commit

git checkout master

### Revert the modified changes in the Working Tree

* Make changes to 6.txt and add some content.
* To revert the changes from the working tree on this file

git checkout -- 6.txt

**Note:** “Working Tree” to “Local Repo” is one-way ie- From Local repo we can not bring back the changes to working tree. Changing history is not possible as each user has it’s own history.

### Git Branches

* The default branch of git is **master**
* Now lets try to create some branches.

mkdir ecommerceapp

cd ecommerceapp

git init

* Now lets create basic folder structure

mkdir src

mkdir test

mkdir docs

touch Readme.md

touch src/main.py

touch test/main.py

touch docs/main.md

* Now commit these changes

git add -A

git commit -m "Initial Folder Structure Created"

* Now this ecommerce application development is agile, We need create the branches for sprints. For now we need to create a branch called as sprint-1

git branch sprint-1

git branch

git checkout sprint-1

* To create a branch called sprint-1 and move the head to sprint-1

git checkout -b sprint-1

* Sprint-1 Developement has made some commits

echo "Sprint 1 Started" >> src/main.py

git add .

git commit -m " Feature 1 completed"

echo "Sprint 1 Started Testing" >> test/main.py

git add .

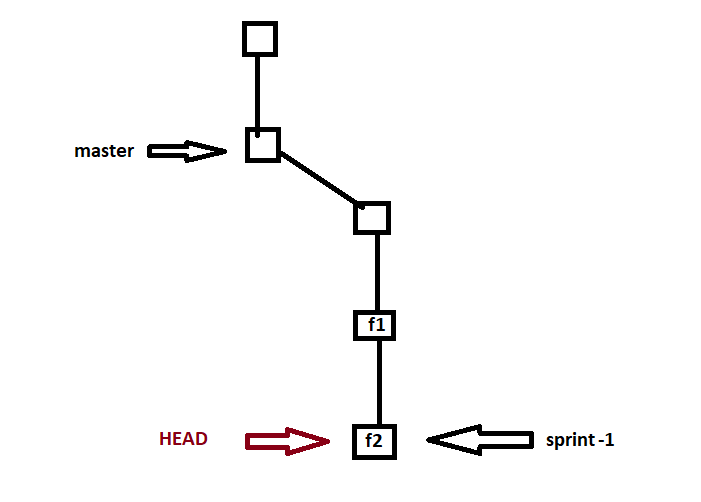
git commit -m "Feature 2 completed"

git log

* Now checkout to master branch and then back to sprint-1

git checkout master

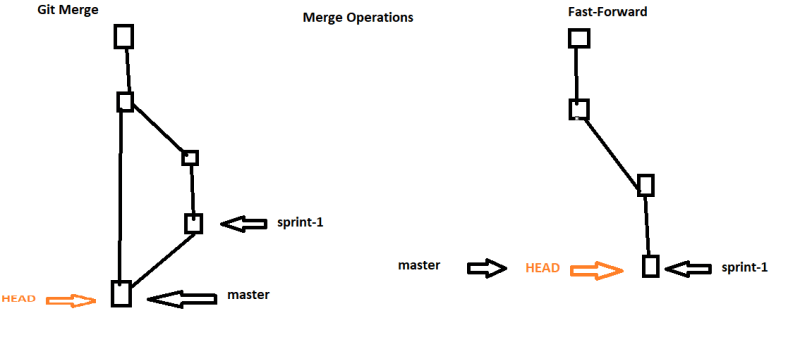
git checkout sprint-1



## DevOps Classroom Series – 15/Dec/2019 – Git Branching

### Merge

* Merge is done when changes from one branch has to be brought to other branches.
* Merges can be of two kinds
  + Fast-Forward:
    - No Extra commit gets created
    - When the Parent branch has no changes and if we try merging child to parent branch
    - Can be skipped
  + Merge:
    - Extra Commit with two parents will be created
    - In all the other cases apart from Fast Forward



* Lets continue from previous repository created from [here](https://directdevops.blog/2019/12/13/devops-classroom-series-13-dec-2019/)
* Merge the changes from sprint-1 to master.
* Note that there were no changes in master branch after sprint-1 branch was created, now merging will lead to fast forward
* Lets merge

# move to the destination branch (master)

git checkout master

git merge sprint-1

* Create a branch called as sprint-2 from master

git checkout -b sprint-2

* An Important fix from sprint-1 is done on master.

git checkout master

echo "Updated Docs of Sprint1" >> Readme.md

git status

git add .

git commit -m "added imp fix"

* Continue working on sprint-2 branch

git checkout sprint-2

echo "sprint-2" >> src/main.py

echo "sprint-2" >> test/main.py

git status

git add .

git commit -m "features in sprint 2"

* Now try to merge the changes from sprint-2 to master

git checkout master

git merge sprint-2

* New commit gets created and HEAD & Master will be pointing to this new commit.
* Create a new branch called as sprint-3.

git checkout master

git branch sprint-3

git checkout sprint-3

### Rebase

* Now add some changes for sprint-3 branch
* Now an important bug fix is done on master, which is required for sprint-3
* Now to rebase from master onto sprint-3, execute the following commands

git checkout sprint-3

git rebase master

* References:
  + [Rebase](https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase)
  + [Merge vs Rebase](https://www.atlassian.com/git/tutorials/merging-vs-rebasing)

### Merge-Conflict

* Merge Conflicts can happen during
  + merge
  + rebase
  + pull
* Lets create a child branch from master and call it as poc-qt

git checkout master

git checkout -b poc-qt

* Now add the following to src/main.py at the end

working for qt

* Now master branch has one commit which changes in src/main.py.
* In master branch main.py has the following contents

Sprint 1 Started

SPrint-2

testing is in progress

need to add many comments

* In qt-poc branch main.py has the following contents

Sprint 1 Started

SPrint-2

working for qt

* Conflicts has to be resolved manually.

### Viewing Differences

* To view differences, use git diff
* Try the following commands

# make local changes on Working tree

git diff

git diff <commit-id-A> <commit-id-B>

git diff <branch-A> <branch-B>

### How is git Working

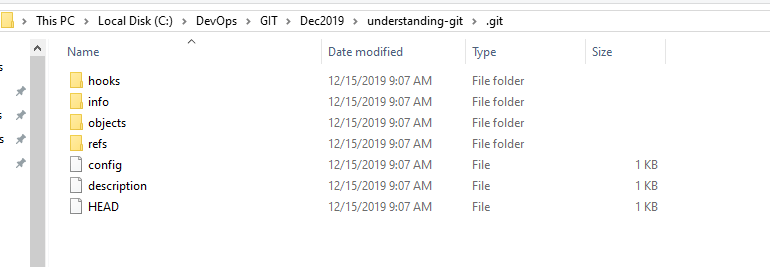
* Git is internally a very simple content tracker
* To understand this, lets create a new folder called as understanding-git

mkdir understanding-git

cd understanding-git

git init

* Now lets get into .git folder and see the contents



* lets make our first change

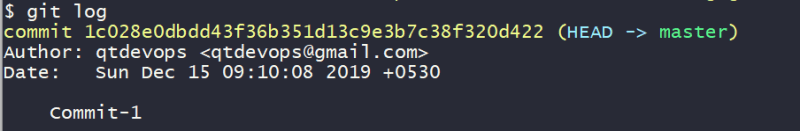
mkdir src

touch src/main.py

git add .

git commit -m "Commit-1"

git log



* Now lets make other change

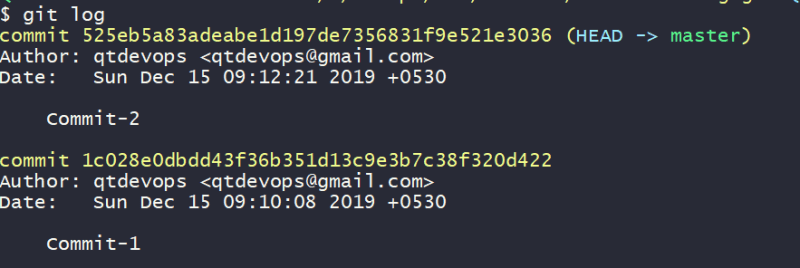
mkdir test

touch test/main.py

git add .

git commit -m "Commit-2"

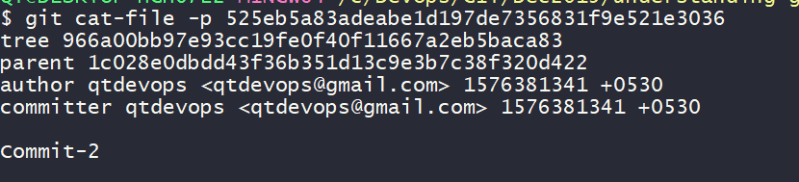
git log



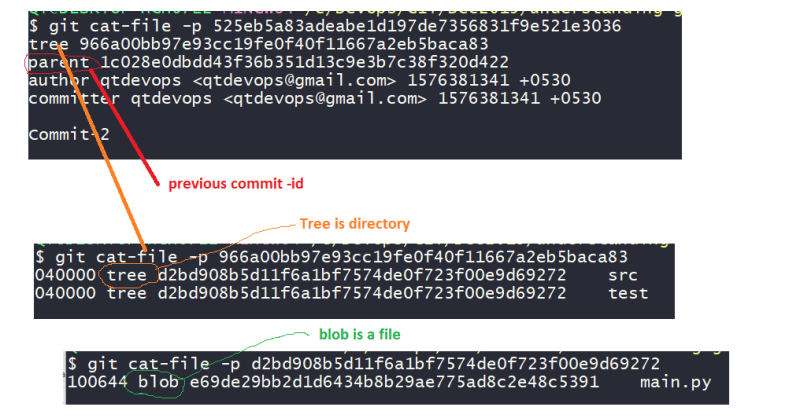
* To understand how git works , we need to understanding [Hashing SHA1](https://en.wikipedia.org/wiki/SHA-1)
* Git Uses hashing (SHA-1) to calculate commit ids. Commit id is hash of many things.
* To understand contents of commit

git cat-file -p <commit-id>

* Lets examine commit-2 contents

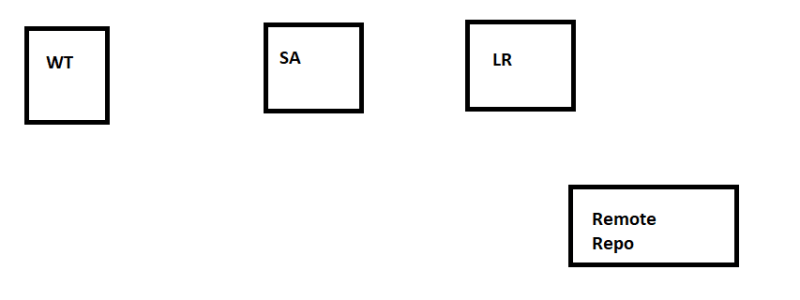


* Take a look at below image

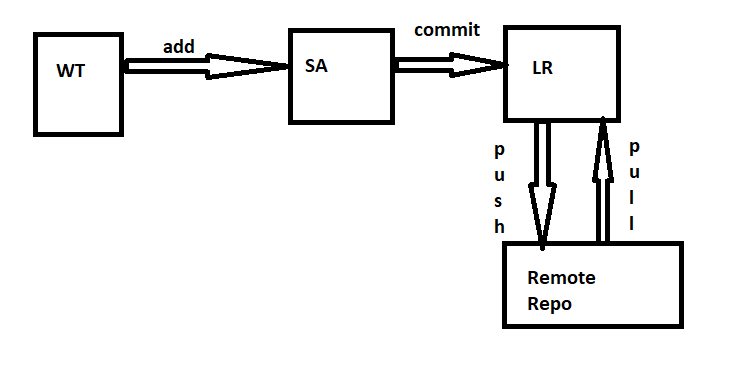


## DevOps Classroom Series 15/Dec/2019 – Git Remote

### Remote Repository

* Remote Repository is git on other node (generally) 
* Any node can host remote repositories. There are many softwares/platforms which can help us.
  + Softwares:
    - Gitolite
    - Git-Lab (Hosted)
  + Platforms
    - GitHub
    - BitBucket
    - Code Commit (AWS)
    - Azure Git Repos

### Remote Repository Scenarios

* Create a Local Repo first and push the changes to Remote Repository
* Clone the Local Repo using Remote Repository 

### Exercise on Remote Repository

* Create a Remote Repository for Local Repository (Ecommerce app)
  + I will be using GitHub
  + Created a new repo and got the url as <https://github.com/GitPracticeRepo/ecommerceapp.git>
  + Push the changes from Local Repo to Remote Repo.
  + Add this new url as your remote repository
* git remote add <name-of-remote> <url>
* git remote add origin https://github.com/GitPracticeRepo/ecommerceapp.git
  + Push the changes to Remote Repository
* git push <name-of-the-remote> <branch-name>
* git push origin master
  + Lets push other branches
* git push origin sprint-1
* git push origin sprint-2
* Clone a Local Repository from existing remote repository
  + I will be using a Repository present at <https://github.com/GitPracticeRepo/AnsibleSamples>
  + Clone a Local Repo from Remote Repo
* git clone <url>
* git clone https://github.com/GitPracticeRepo/AnsibleSamples.git

### How the Remote is Handled

* Whenever a Remote Repo is added, new branches gets created in Local Repository. <Remote-name>/<branch-name> will be the name for the remote branch.
* Sending the changes from local to Remote is Push
* Getting the latest changes from Remote to local is Pull (Fetch + Merge)

### Changing Hisory of commits

* Use Interactive Rebasing.

### Cherry-Pick

* To pick individual/sequence of commit(s) from one branch to other use cherry-pick
* [Refer Here](https://www.atlassian.com/git/tutorials/cherry-pick)

### Bare Repositories

* Git repositiries with only .git folders
* Generally used on servers and also used for taking backup

git clone --bare <url>

### One local Git Repo with multiple Remote Repos

git clone <url>

# a remote called as origin gets created

git remote add <other-repo-name> <other-repo-url>

### Topics to be covered

* Git Communication Protocols
* Git Stash
* Git Tags
* Git Branching Strategy
* Git Reflog
* Git config
* Git Hooks/WebHooks

### Exercises

* Demonstrate Fast-Forward
* Demonstrate Merge
* Demonstrate Rebase
* Remove 3 commits from histroy
* Combine 2 commits into one
* Get one commit from one branch to other
* Demonstrate Merge-Conflict

# JENKINS

**DECEMBER 16, 2019**

## DevOps Classroom Series – 16/Dec – Installing Jenkins

### Jenkins

* Is an Open source CI/CD Engine
* It is referred by many as **scheduler on steriods**
* Jenkins was designed for CI/CD
* Jenkins internally calls/executes commands.
* Generally when you install jenkins, a new user (jenkins) gets created. Whatever you do in jenkins will be run as jenkins user
* In Windows, Jenkins runs as installed user.
* Jenkins is a Web Application Developed using Java.
* Jenkins runs by default on port 8080

### Important considerations for Working with Jenkins

* Ensure user all the necessary permissions
* Ensure all the Environmental Variables like PATH, PYTHON\_HOME etc are defined correctly
* Before using any software (git, maven, docker, ansible etc) ensure they are installed and configured correctly

### Installing Jenkins

* Install Java
* Install Jenkins

#### Installing Jenkins on Ubuntu

* [Refer Here](https://jenkins.io/doc/book/installing/) (<https://jenkins.io/doc/book/installing/>)  for installation steps
* [Refer Here](https://directdevops.blog/2019/01/04/installing-specific-lts-version-of-jenkins-on-ubuntu/) (<https://directdevops.blog/2019/01/04/installing-specific-lts-version-of-jenkins-on-ubuntu/>)  for installing specific version of jenkins on ubuntu
* [Refer Here](https://pkg.jenkins.io/debian-stable/) (<https://pkg.jenkins.io/debian-stable/>) for debian/ubuntu based

**DECEMBER 17, 2019**

## DevOps Classroom Series 17/Dec/2019

### Jenkins Terms

* Project:
  + Set of Steps to be scheduled multiple times.
  + Internally every project stored as XML file.
  + This file gets stored in Jenkins-Home directory
* Build :
  + Project when executed is generally considered to be build.
  + Each build for a project will have unique running number.
* Master:
  + Jenkins Server is Master
* Node:
  + All the other server which are added to Jenkins Master are nodes.
* JENKINS HOME:
  + All the information about jenkins is stored in this directory
  + The default path is ~/.jenkins
  + In specific linux installations path can be different.
  + To know about Jenkins Home, Jenkins uses an environment Variable called JENKINS\_HOME

-----------------------------------------

Jenkins Home Location

root@JenkinsServer-Ubuntu:/home/osboxes# dpkg -L jenkins

root@JenkinsServer-Ubuntu:/home/osboxes#cat /etc/default/jenkins

# jenkins home location

JENKINS\_HOME=/var/lib/Jenkins

------------------------------------------

### Simulating any Jenkins Job Execution

* Login into the Jenkins master
* Become Root user

sudo -i

* Switch to jenkins user

su jenkins

* Execute commands

whoami

pwd

sudo apt-get update

### Exercise

* Create a simple Free Style Jenkins Project to display environmental variables, Username and list the contents of home directory

set

whoami

ls ~

**DECEMBER 18, 2019**

## DevOps Classroom Series – 18/Dec/2019 Jenkins Configuration

### Configuring Jenkins

#### Add Sudo access to Jenkins with NO Password

* Login and become root user and then execute the following commands

vi /etc/ssh/sshd\_config

# password Authentication to Yes

service sshd restart

visudo

# add the below line

# jenkins ALL=(ALL:ALL) NOPASSWD:ALL

#### Using git with jenkins

* ensure git is installed

sudo apt-get install git -y

* Login as jenkins user and verify the git using

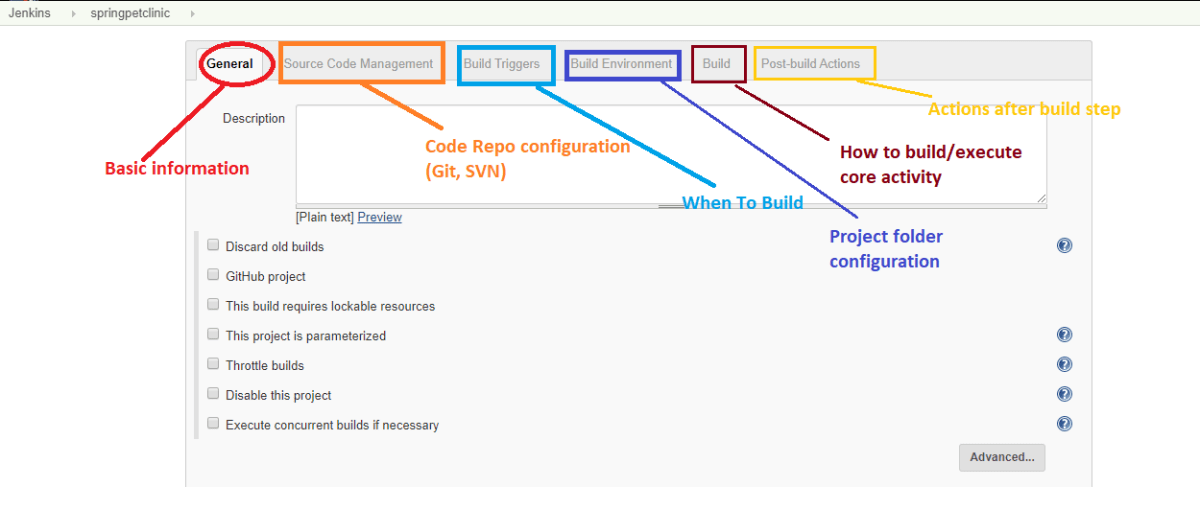
git --version

* Note: This applies to any softwares in use from jenkins

### Jenkins Projects

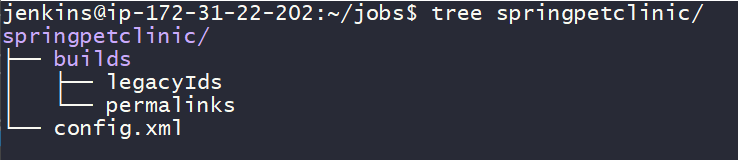
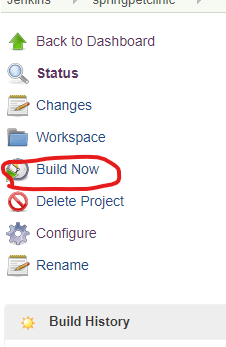
* Two kinds of Popular Projects are available in Jenkins
  + FreeStyle:
    - Project to create schedules of any activities
    - Heavily relies on UI Components of Jenkins.
    - If UI Components are not found, they can be added using **plugins**
  + Pipeline

#### Create a free style project to clone the code from git (GitHub)

* The project we will be cloning is spring pet clinic [from here](https://github.com/spring-projects/spring-petclinic) (<https://github.com/spring-projects/spring-petclinic>)
* Create a new Item and select the options as shown below 
* To understand different sections of Jenkins Freestyle project, Refer below 
* Now configure Source Code Management with Git url <https://github.com/spring-projects/spring-petclinic.git>
* In the build section => Execute Shell

pwd

ls

* Lets examine what has happened in Jenkins Home Directory
* Whenever a project is created , In the Jobs folder of Jenkins Home directory, a new folder with project name gets created. In this folder config.xml is project configuration, builds folder is available for symlinks of successful and pastbuilds 
* Click on Build Now as shown below 
* When your project/job is executed a folder gets created in Jenkins Home Directory in workspace folder

### Continuous Integration

* Purpose of CI is give feedback about the commit(s) from developer(s)

### Continuous Delivery

* Purpose of CD is give feedback about the days work done by the dev team
* Importance is for System Test/Performance Testing

**DECEMBER 19, 2019**

## DevOps Classroom notes – 19/Dec/2019

### Building Code and Unit Testing Using Jenkins

* It depends on Technology (EcoSystem)
* Every Technology/Application has different set of build tools and unit testing
* In Jenkins Approach is
  + Install necessary tools
  + configure tools for Jenkins user
  + In the build section execute these commands

### Example: Building Docker Image

* Even if you are not familiar with docker, we can still build this.
* First lets install Docker. Login as Jenkins user and do the following

curl -fsSL https://get.docker.com -o get-docker.sh

sh get-docker.sh

sudo usermod -aG docker jenkins

sudo service jenkins restart # when it is configured in jenkins

exit

# relogin

docker info

* Let’s look at commands

git clone https://github.com/DevProjectsForDevOps/StudentCoursesRestAPI.git

docker image build -t studentcourserestservice:1.0 .

* Create a free style Jenkins project, Select the Source Code Management Tab and give git details, and then to build section and Execute shell

docker image build -t studentcourserestservice:1.0 .

### Build/Compile/test Java Using Maven.

* Building Java Code
  + Compiling Each and every .java file
  + archive (zip) all the generated .class files into war/jar/ear
* To do this build activity, there are many build tools
  + ANT
  + Maven
  + Gradle
* In this series we restrict ourselves to Maven
* Maven Installation on ubuntu

sudo apt-get update

sudo apt-get install maven -y

mvn -v

# MAVEN

**DECEMBER 20, 2019**

## DevOps Classroom Series – Maven – 20/Dec/2019

### Maven

* Is a Project Management Tool.
* Can be used for
  + build
  + dependency Management
  + Releases
  + Documentation
  + Test Executions
* Maven prefers conventions over configurations
* Maven also works for Java Based Languages
  + groovy
  + Scala
* Maven uses a file called as **pom.xml** to define
  + dependencies
  + project information’s
  + plugins

### Installation

* Before installing maven ensure Java is installed
* Ubuntu:

sudo apt-get update

sudo apt-get install maven -y

### Maven folder conventions

* Code: <projectdirectory>/src/main/java/
* Location of pom: <projectdirectory>/pom.xml
* Test: <projectdirectory>/src/test/java/
* Target Folder: <projectdirectory>/target

### pom.xml

* POM (Project Object Model): is a xml which defines
  + Project info
  + Dependencies
  + Plugins
  + Profiles

### Goals

* compile : compile the code
* test: compile the code + test the code
* package: test the code + package the application
* install: pushes the pom file and jar/war to ~/.m2
* deploy: pushes the pom file and jar/war to Remote/Central Repo
* clean: remove the target folder

### Executing goals

* single goal execution

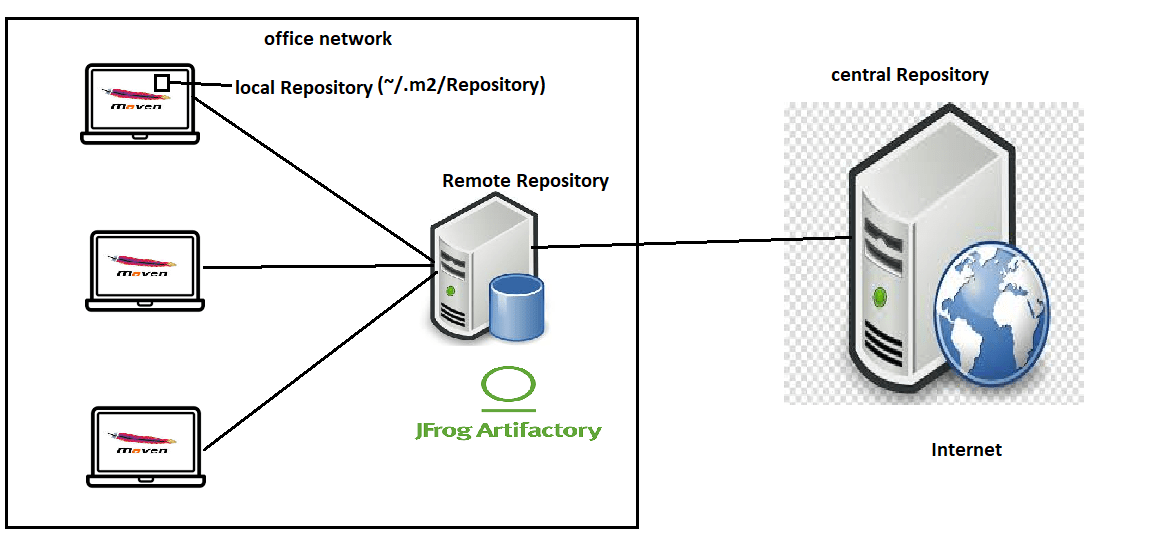
mvn compile

mvn test

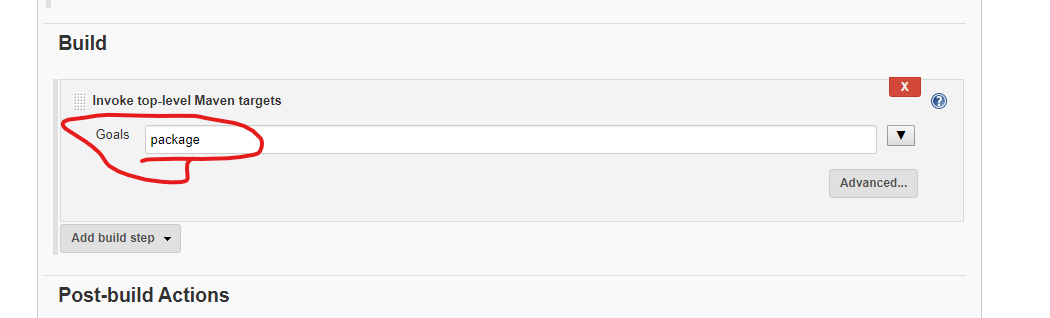
* multi goals

mvn clean package

### Maven Repository Architecture



### Using Maven with Jenkins

* Install maven and ensure Jenkins User has access to maven
* In the build section of Free style project
  + Execute Shell: directly execute the maven command
  + Invoke Top level Maven Targets: Specify goals from here 

**DECEMBER 22, 2019**

## DevOps Classroom Series – 22/Dec/2019 – Maven With Jenkins

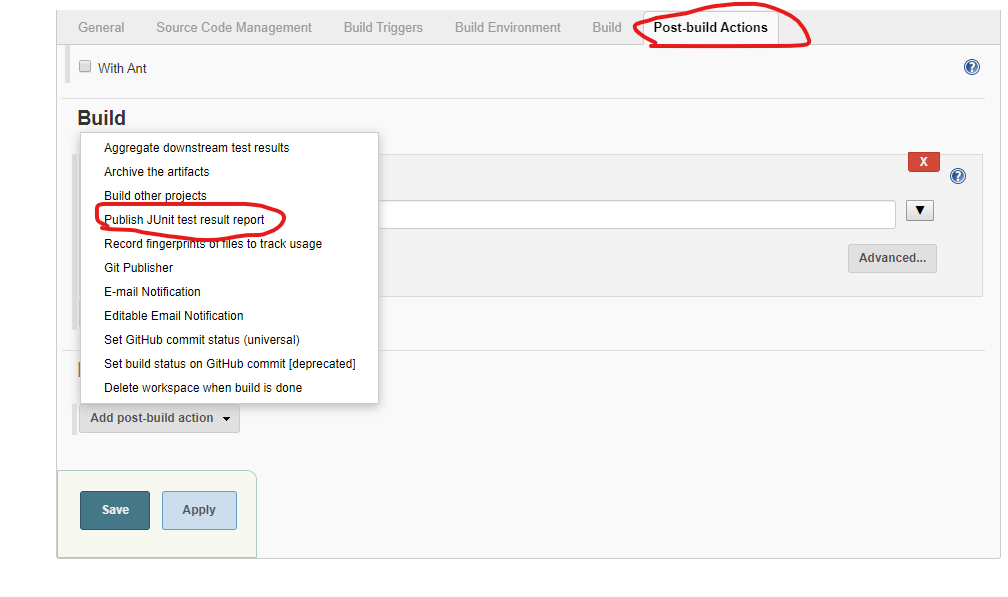
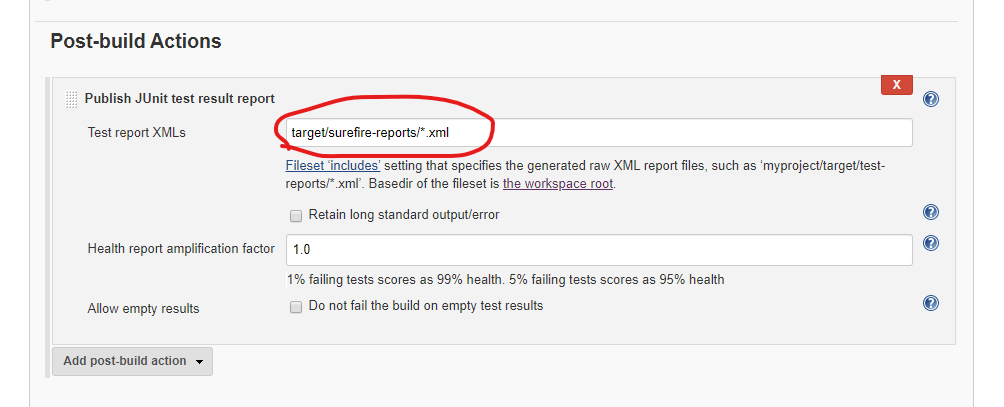
### Testing Types

* Unit Testing:
  + Testing smaller unit of developments
  + Test Harness tools:
    - JUNIT
    - MSTest
    - pytest
* Automated System Testing:
  + web Based Applications:
    - Selenium
    - Karate
    - QTP
  + API Based Applications
    - Soap UI
    - Postman
  + Mobile Apps:
    - Appium
  + Performance Testing
    - JMeter
    - Load Runner

### What is Measured From Tests

* Pass Rate
* Coverage

### Integrating Maven tests with Jenkins

* To configure publishing of junit test results to jenkins
* ensure your maven goal has test execution
* Navigate to post-build Actions and Select as shown below 
* Select xml files in surefire-reports folder in target 

### Archiving the Artefacts (Displaying the package built)

* In post build actions, select the section ‘archive the artefact’ and give the artefacts location (for e.g.: target/\*.jar)

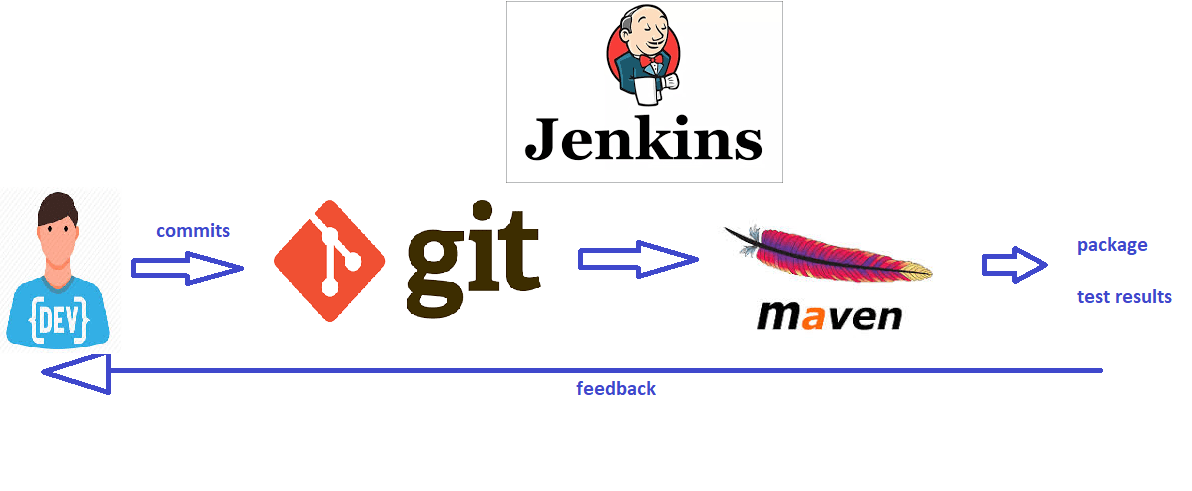
### Post Build Actions:

* Activities that are performed after build is completed
* Most commonly post builds are
  + Show Test Results
  + Show the Package
  + Call other Jenkins Project to start building
  + Send Emails to the team

### Jenkins Plugins

* Plugin is extra functionality into Jenkins
* Plugins can be installed into Jenkins. This installation can be from
  + Online
    - Plugins get downloaded from internet
  + Offline
    - Upload plugin to Jenkins
    - Plugin has two popular formats (hpi, jpi)

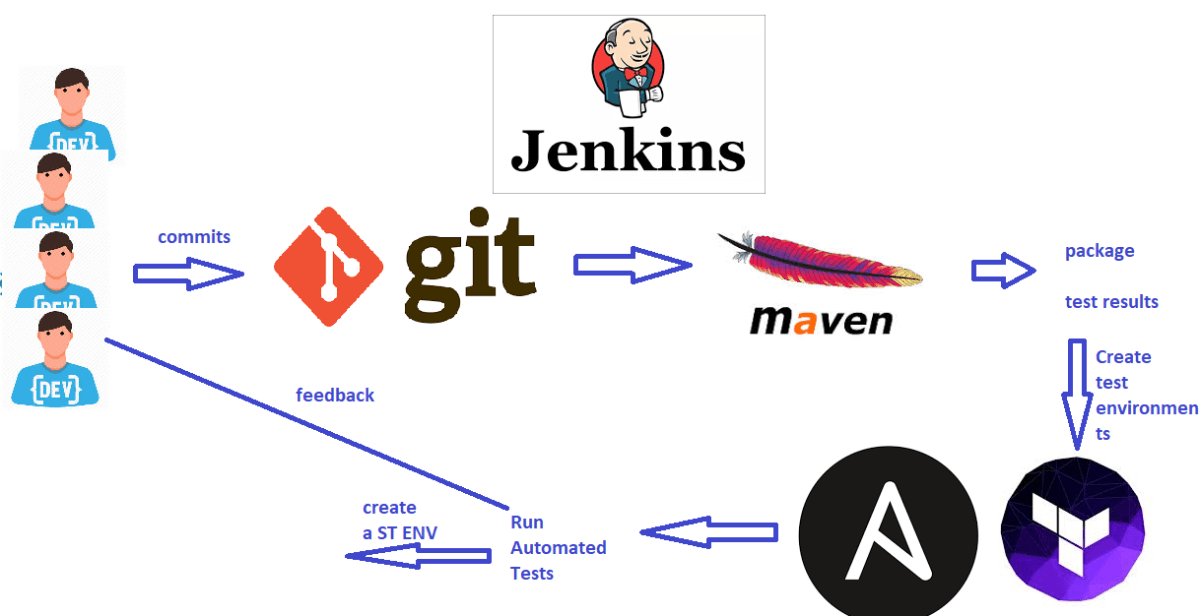
### Day Builds



* Day builds basic intention is
  + to give feedback of code quality of the commit(s) done by one (or more) developers for shorter period of time during the day.
  + Have to finish quickly

### Night Builds

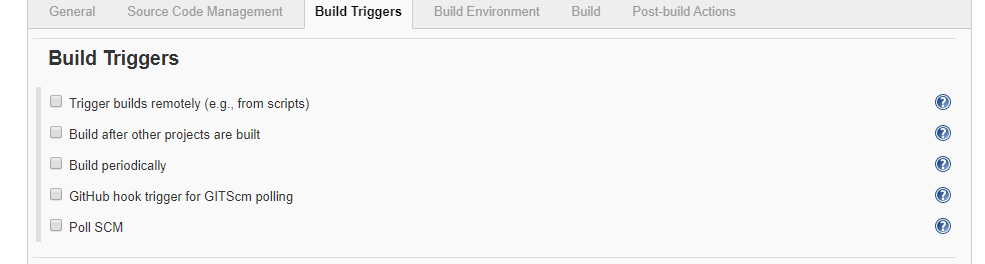
* Night Builds basic intention is
  + to give feedback of the product quality for the collective work done by developers during past day
  + Execute extensive tests (unit, System, Performance)
  + Time is no bar
  + Night Builds are used by system testers to execute the Functional test
  + Night Builds qualify for release



**DECEMBER 22, 2019 (Evening)**

## DevOps Classroom Series – 22/Dec/2019 – Jenkins Distributed Build

### Jenkins Build Triggers

* Will help in triggering the Jenkins Build based on
  + Schedule:
    - Periodic: e.g. Every 1 hour, Every 2 hours etc.
    - On Schedule: e.g. Every weekday at 3 AM
  + Git Commits: Any new commits to Git Repo. Two ways of doing this
    - Poll SCM: Jenkins will poll Git
    - Git/Web Hooks: Git will inform Jenkins whenever new commits happen
  + After Other Jobs are Built 

### Build Periodically

* Here you configure the schedules by using syntax which is much like cron jobs.

MINUTE HOUR DOM MONTH DOW

H/15 \* \* \* \*

0 4,15 \* \* \*

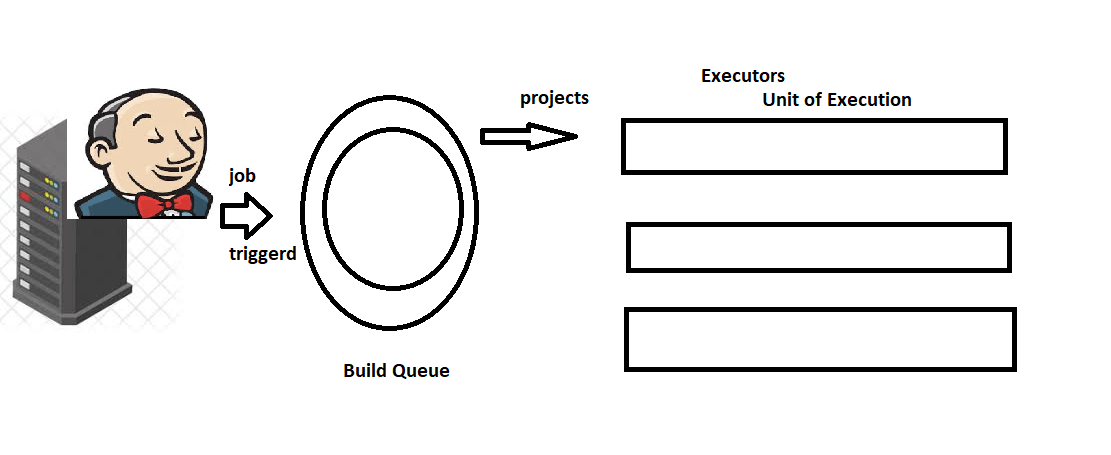
### Poll SCM

* Triggers a build whenever changes appear in git.
* Using Jenkins, user should describe how frequently poll should happen. For this we use the same syntax mentioned above

### Build Environment

* Some of the Prebuild Actions such as cleaning workspace
* Build Settings such as when build is taking more time, abort/fail etc. can be configured
* Additions to console logs

### Jenkins Logical Architecture



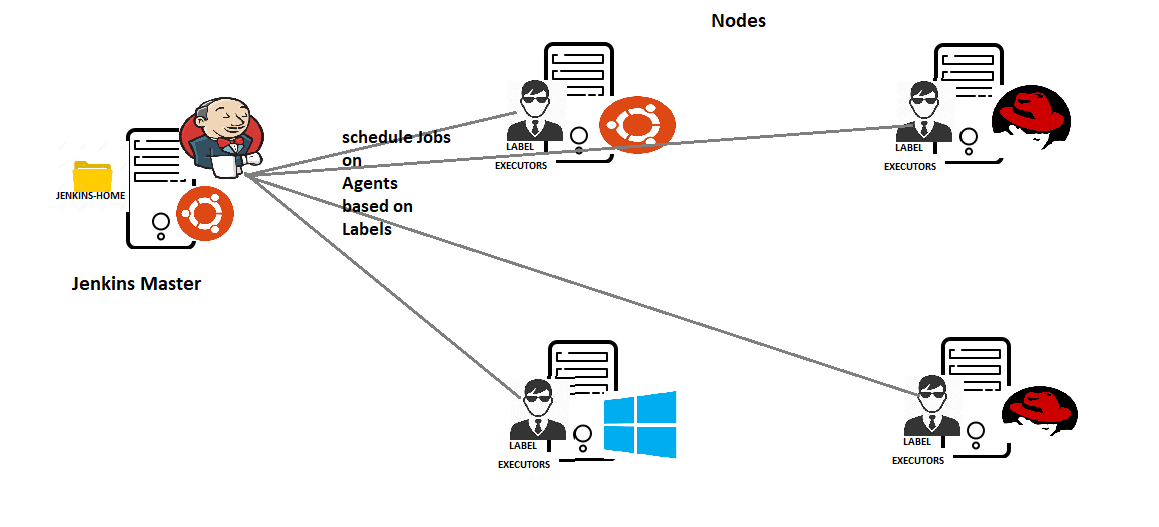
### Queue

* Any build from Jenkins will be sent to Queue.
* It stays in the Queue till it finds free executor

### Executor

* Is Execution unit of Jenkins (This is where build happens)
* Number of Executors signify number of parallel builds that can happen
* By Default, a project if it is under execution, will not be executed by other executors (Clicking Build now twice on same project will not lead to two executors)

### Jenkins Distributed Builds



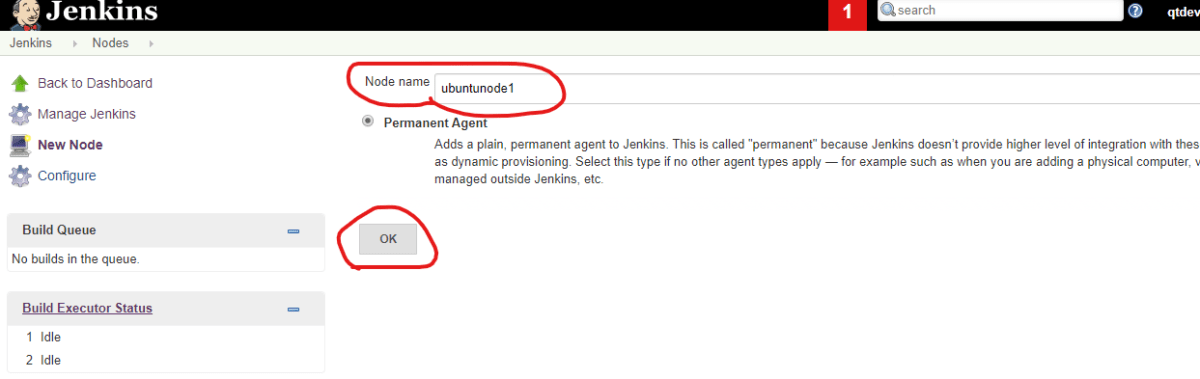
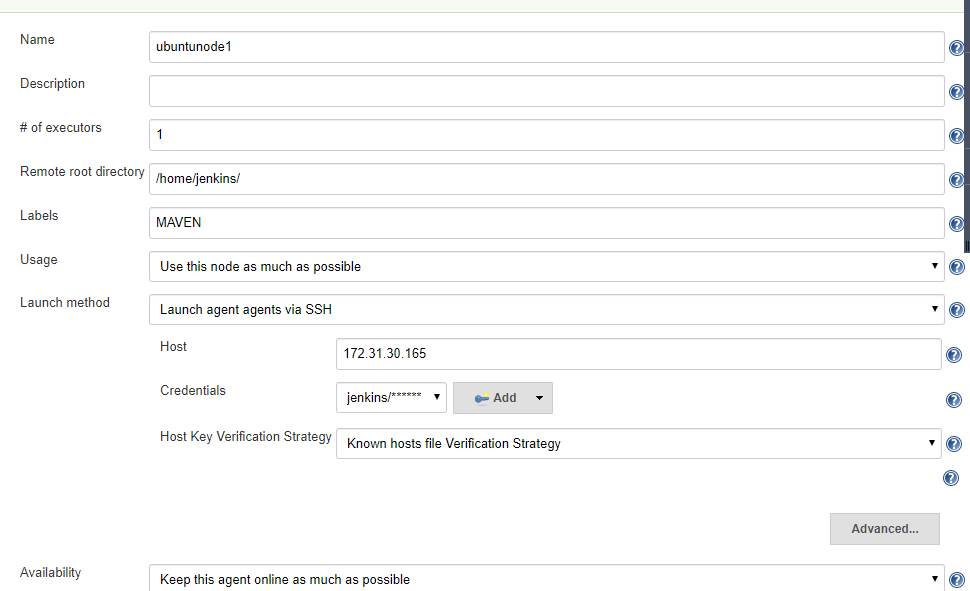
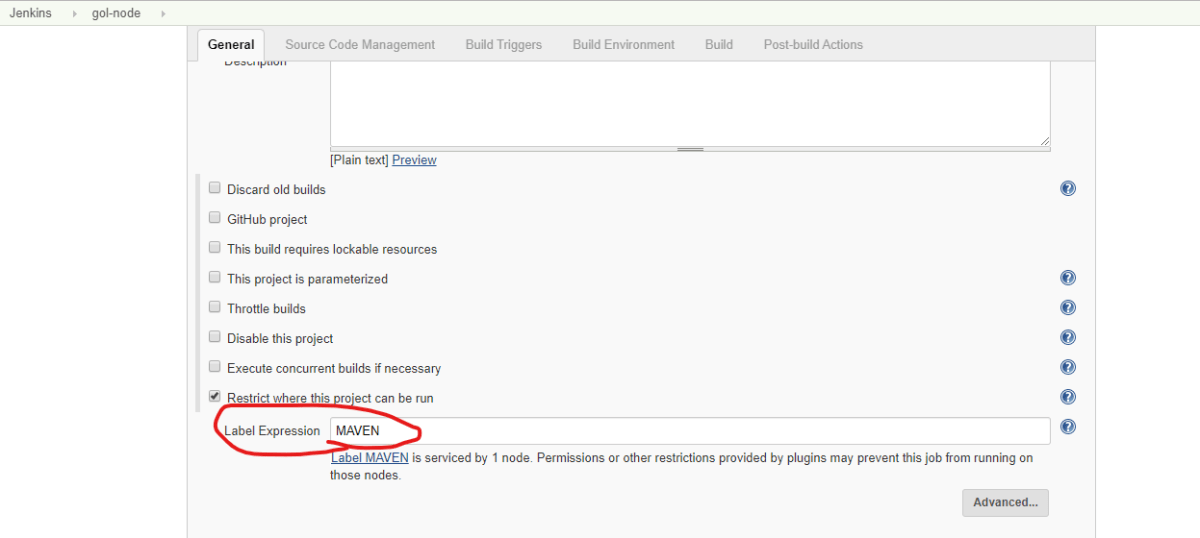
* Two kinds of Server Components are present in Jenkins
  + Master:
    - Jenkins Service will be running
    - Jenkins Home directory will be present
    - Communicates with Jenkins Agents using SSH/TCP (jnlp)
  + Node
    - Jenkins Agent will be running
    - Only the builds workspaces scheduled will be available
    - Every Agent should have some labels.

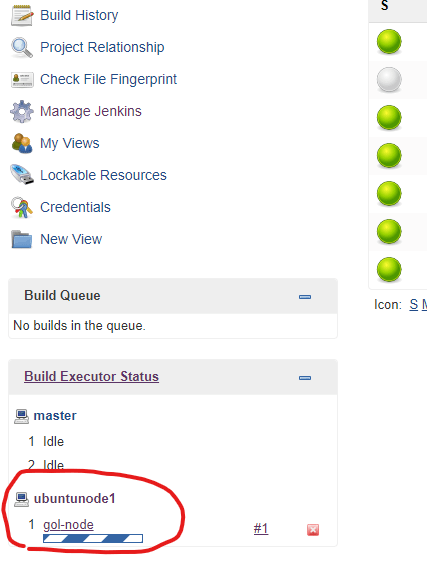
**DECEMBER 23, 2019**

## DevOps Classroom Series – Jenkins – 23/Dec/2019

### Configuring Linux Nodes to Jenkins Master

### Adding Ubuntu Node to Jenkins Master

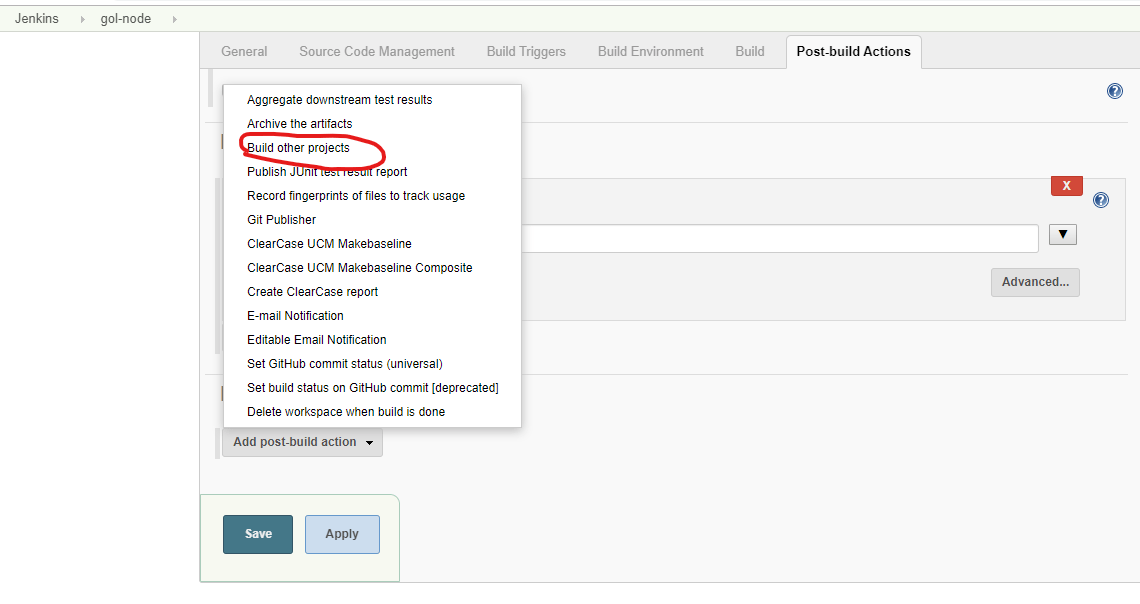
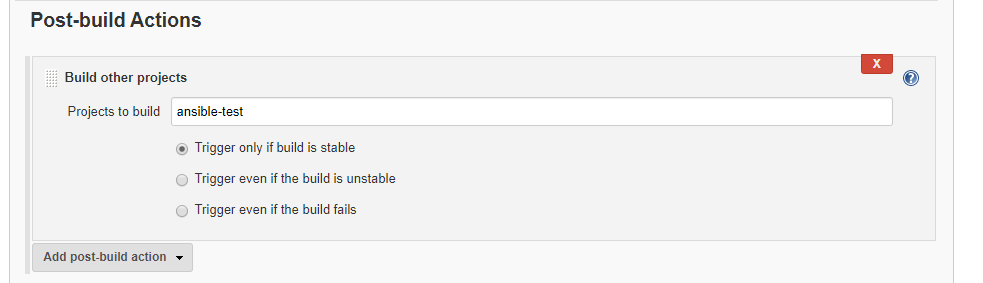
* Jenkins Master is up and running
* Create one more Ubuntu VM in the same network
* Login into the Ubuntu vm and create a user Jenkins, enable password authentication.
* Install JDK and maven
* Login into Jenkins master as Jenkins user and try to ssh into newly added node
* Add the node to the Jenkins master. Select Manage Jenkins => Manage Nodes => New Node 
* Configure the new node using known host strategy (bcoz we already have known hosts) and add a Label **MAVEN**.
* Add the credential username with password. 
* Now let’s try to build a java application on this Node
* Create new Jenkins Freestyle project and restrict where this project is built in General Section 
* Configure the other sections to have build steps and click on build now



### Exercise-1

* Add a Centos node to Jenkins master

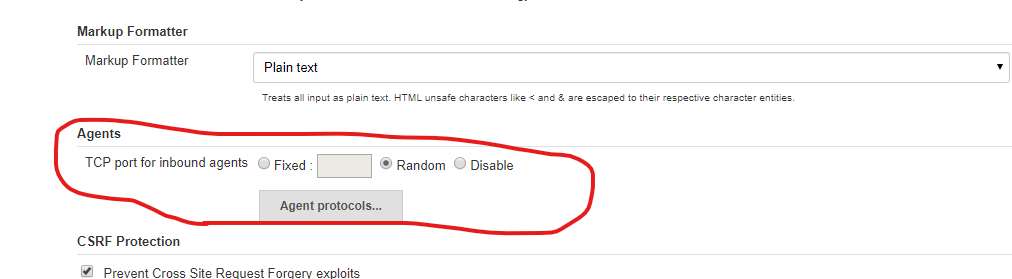
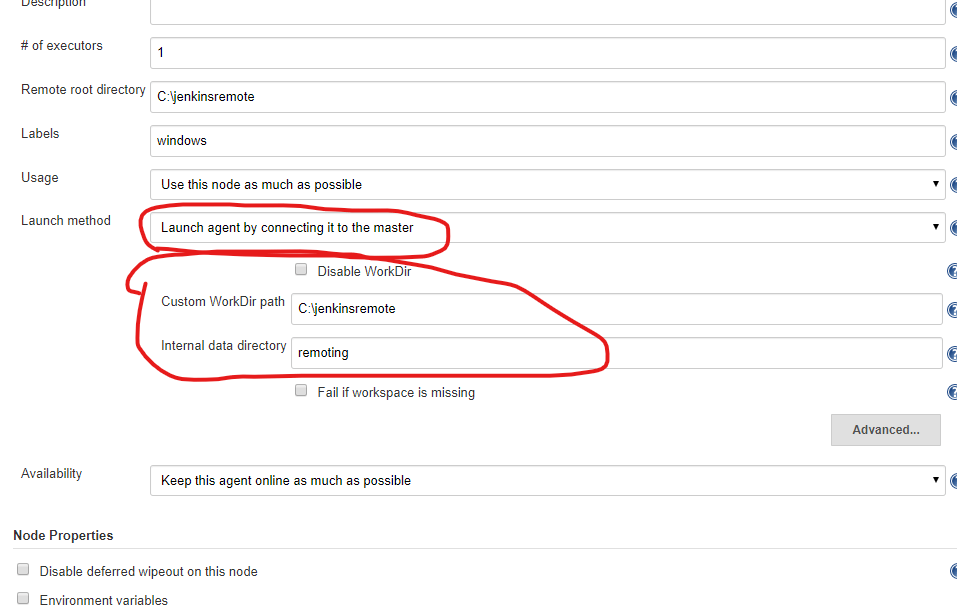
### Adding Downstream Jobs to Jenkins Job

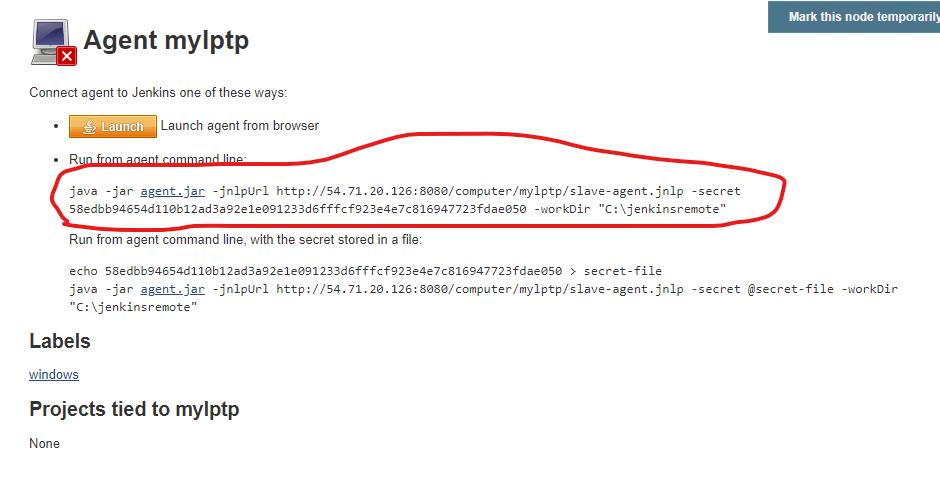
* Downstream job is job which gets executed after completion of other job (e.g.: job A completion triggers Job B, B is Downstream to A and A is upstream to B)
* Navigate to post build sections and select as shown below 
* Enter the project name that has to be executed 

**DECEMBER 24, 2019**

## DevOps Classroom Series – Jenkins Windows Node – 24/Dec/2019

### Adding Windows Node to Jenkins Master

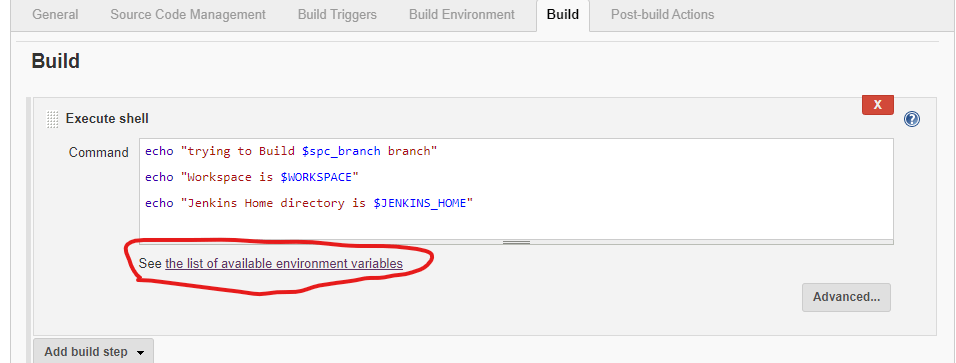
* Ensure your Jenkins Master is Running
* Create a Windows Instance with Java Installed. In this scenario we would use Windows 2016 Server
* Install Chocolatey and install Java 8
* To Connect Jenkins windows Node to Jenkins Master we use JNLP, To enable JNLP Navigate to Manage Jenkins => Global Security and Enable the Agents as shown below 
* Navigate to Manage Jenkins => New Node and add the necessary configuration 



* Now Create a Jenkins job to schedule the build on windows server

### Build Parameters

* Build Parameters are the options to user to pass parameters while building the jobs
* Parameter types available are
  + String
  + Boolean
  + Choice
  + File
  + Credentials
* To use these parameters on the Linux node, use the following syntax $Parameter-name
* Along with Build Parameters defined by the users Jenkins gives the list of Environmental Variables, To See complete list refer docs or do as shown below





**DECEMBER 25, 2019**

## DevOps Classroom Series – Jenkins Pipeline – 25/Dec/2019

### Need For Pipelines

* Consider a Repository (<https://github.com/GitPracticeRepo/spring-petclinic>) where there are multiple branches and User is expected to create multiple projects for building each branch.
* To achieve this we create multiple projects every time manually configuring the options
* For master branch, you are supposed to do clean build (mvn clean package) and sprint-1 branch incremental build (mvn package)

### Jenkins Pipeline

* Is a scripted way of defining jenkins job.
* Jenkins job can be defined in git repository in one file.
* Lets try this.
* Navigate to code repository and create a file **Jenkinsfile** with following content

node {

stage('SCM') {

// git clone

git 'https://github.com/GitPracticeRepo/spring-petclinic.git'

}

stage ('build the packages') {

// mvn package

sh 'mvn package'

}

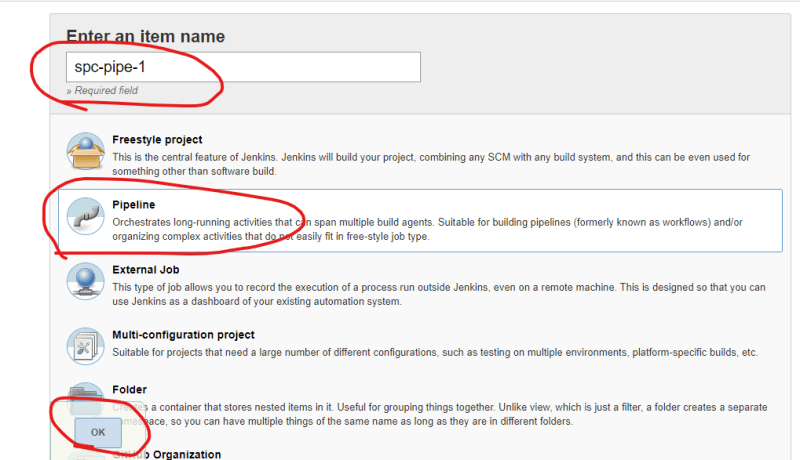
stage ('archival') {

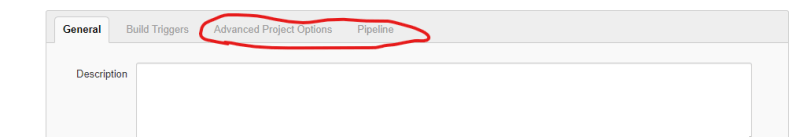
// archiving artifacts

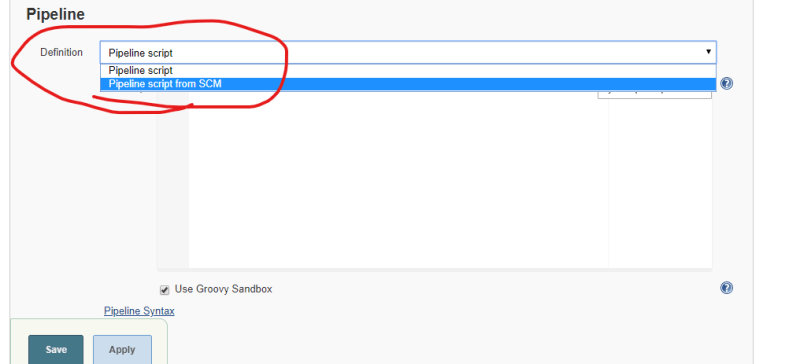
archive 'target/\*.jar'

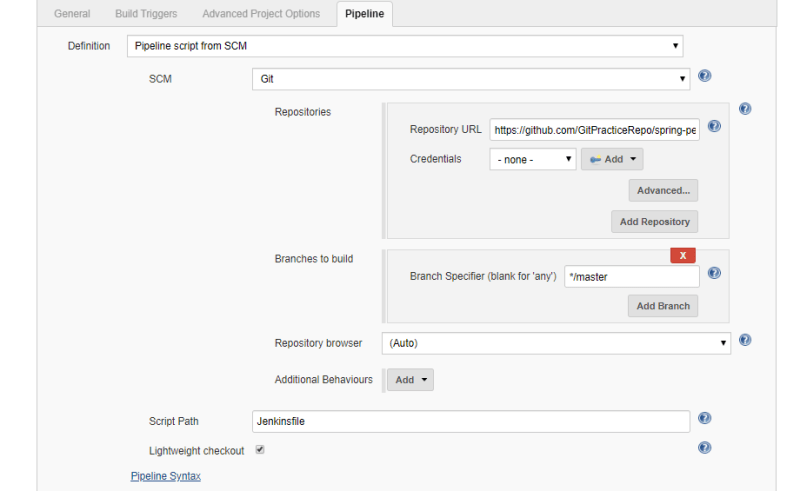
}

}

* Now open Jenkins and create a pipeline project as shown below 







* In Jenkins Pipeline the script is written by using groovy language (Java Based Language) with this we get the following benefits
  + Customization of builds become simpler
  + Creating reusable build library is possible
  + Even in the case of plugins what we generally call from Script is functions.

**DECEMBER 26, 2019**

## DevOps Classroom Series – 26/Dec/2019 – Jenkins Pipeline

### Jenkins Pipeline Syntax

* Pipelines are written in Groovy language.
* Jenkins has defined certain Functions and blocks
* Jenkins has Developed DSL (Domain Specific Language)

### Jenkins Pipeline – Important Definitions

* Node: Machine where Jobs get executed. Node is idenified by labels

node('LABEL') {

// your build config

}

* Stage: Complete build activity can be broken into multiple stages.

stage('<Stage name>') {

}

# Generally stages are part of nodes

node('MAVEN') {

stage('GIT') {

}

stage('BUILD'){

}

}

* shell: Executing shell in jenkins

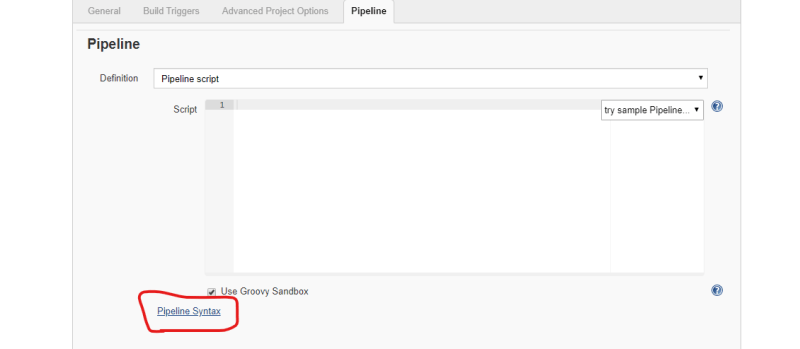
sh <command>

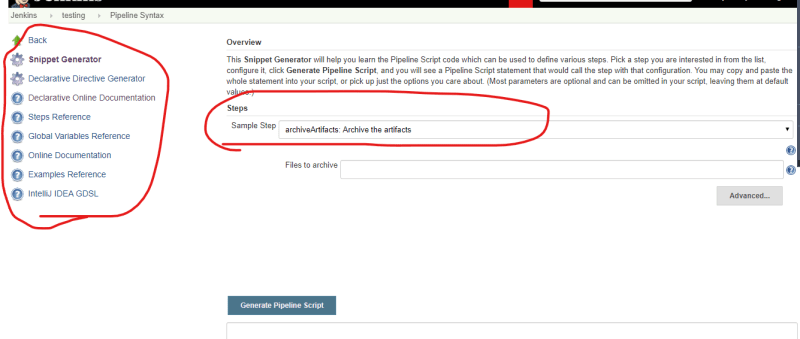
sh 'mvn clean'

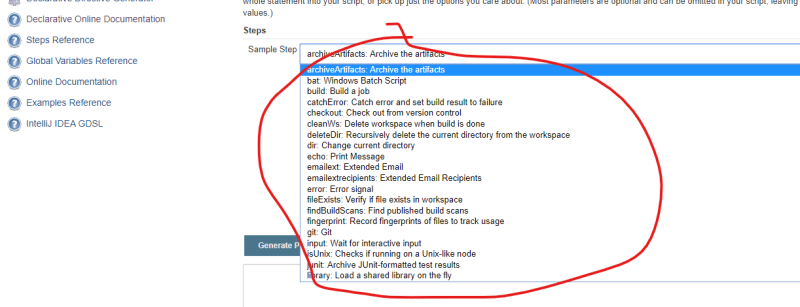
* git: Executes the git clone and pull operation. Refer [here](https://jenkins.io/doc/pipeline/steps/git/) (<https://jenkins.io/doc/pipeline/steps/git/> )

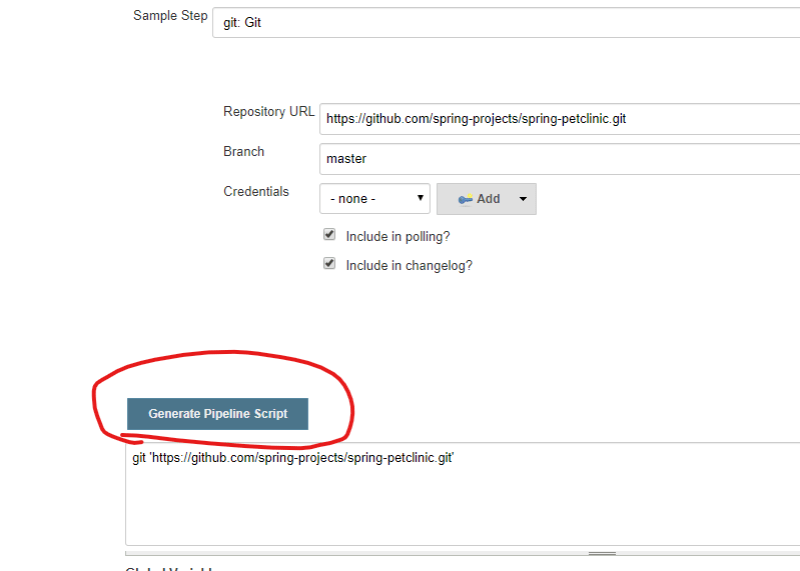
git '<url>'

### Quick Wins for generating pipelines.

* Create a jenkins pipeline project and navigate to pipeline section perform the below steps 







* Try to generate pipeline steps for individual activities from Freestyle Projects.
* Exercise-1: Convert every Free-Style Project to Jenkins Pipeline
* Exercise-2: Create a pipeline project with parameters

**DECEMBER 29, 2019**

## DevOps Classroom Series – 29/Dec/2019- Morning

### Scripted Pipeline

* Directly allows users to write Groovy
* Learning curve towards Groovy
* Main blocks/steps are
  + Node
  + Stage
  + git
  + sh
  + bat
* Example

node('mvn') {

stage('scm') {

git 'https://github.com/springpetclinic.git'

}

stage('build') {

if env.branch == 'master'

sh 'mvn package'

else

sh 'mvn clean package'

}

}

### Declarative Pipeline

* Created Declarative Syntax for Jenkins (DSL)
* Simpler learning Curve
* Declarative Steps are
  + pipeline
  + stages
  + stage
  + label
  + when
  + agent
  + sh
  + git
* Example

pipeline {

agent {

label 'MVN'

}

stages {

stage ('git') {

....

}

stage ('build') {

......

when branch == 'master'

....

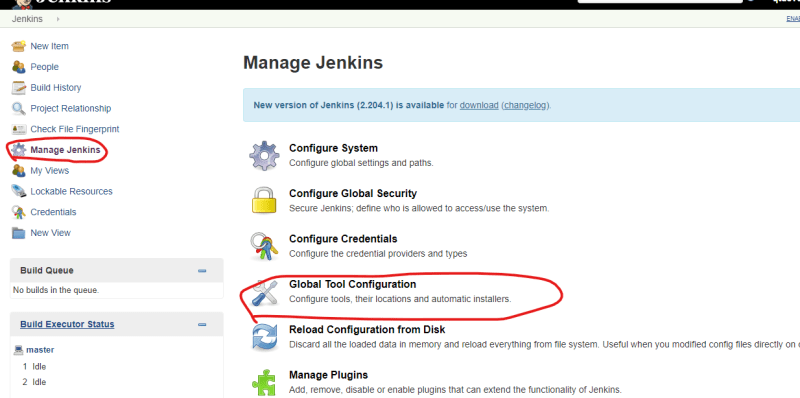
}

}

}

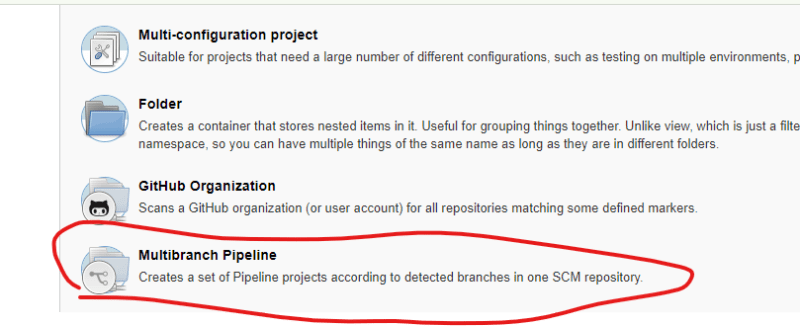
* [Refer Here](https://jenkins.io/doc/book/pipeline/syntax/) (<https://jenkins.io/doc/book/pipeline/syntax/>)  for specifics

### Configuring Tools to Jenkins



* Configure Tools like
  + git
  + Maven
  + Docker
  + Ant
  + Gradle
  + Artifactory
  + SonarQube

### Multi Branch Pipeline

* Ensure every branch has a Jenkinsfile and Jenkins automatically indexes all the branches of your project.
* Multi branches builds can be triggered at one shot.
* Navigate to New-Item => Multi Branch Pipeline 
* Best practice is to implement git hooks to scan multi branch pipeline.

### Writing Jenkins Declarative Pipelines

* [Refer Here](https://jenkins.io/doc/book/pipeline/syntax/#declarative-pipeline) (<https://jenkins.io/doc/book/pipeline/syntax/#declarative-pipeline>)  for official docs
* Basic Pipeline Sytnax

pipeline {

// pipeline stuff

}

* Agent: Where the job has to run

// run anywhere

agent any

// run on the agents with label MAVEN

agent {

label 'MAVEN'

}

* Stages: configure sequence of stages. Example – do git clone and then mvn package

pipeline {

agent any

stages {

stage('git') {

git 'https://github.com/asquarezone/game-of-life.git'

}

stage('mvn') {

sh 'mvn clean package'

}

}

}

* Parallel execution of stages

parallel {

stage('act1') {

steps {

echo 'act1'

}

}

stage('act2') {

steps {

echo 'act2'

}

}

}

* when: Conditional check to execute stage or not
* env: To access jenkins environmental variables use env.<variable name> in scripted pipeline
* parameters: To set parameters in declarative pipelines. [Refer Here](https://jenkins.io/doc/book/pipeline/syntax/#parameters) (<https://jenkins.io/doc/book/pipeline/syntax/#parameters>)

pipeline {

agent any

parameters {

string(name: 'learning' )

}

stages {

stage('test') {

steps {

echo '${params.learning} is the parameter'

}

}

}

}

* Build Triggers: [Refer Here](https://jenkins.io/doc/book/pipeline/syntax/#triggers) (<https://jenkins.io/doc/book/pipeline/syntax/#triggers>)
* Input: [Refer Here](https://jenkins.io/doc/book/pipeline/syntax/#input) (<https://jenkins.io/doc/book/pipeline/syntax/#input>)
* Also refer Globals Section from Pipeline Syntax Page
* Also refer Steps

pipeline {

agent any

parameters {

string(name: 'purpose',default: 'learning')

}

stages {

stage('git'){

git url: 'https://github.com/spc.git', branch: 'sprint-2'

}

stage('build the code'){

sh script: 'mvn package', label: 'build'

}

stage('archive the artifacts'){

archiveArtifacts archive: 'target/\*.jar', onlyIfSuccessful: true

}

}

}

**DECEMBER 29, 2019**

## DevOps Classroom Series – 29/Dec/2019- Eveining – Integrating Jenkins with Artifactory, Sonar Qube

### Integrating Jenkins with Sonar Qube

* Sonar Qube can help in Static Code Analysis
* Code Quality Reports
  + Coding Guidelines and Reports issues
    - Warning
    - Error
    - Information
  + Coverage
    - Line Coverage
    - Symbol Coverage
    - Branch Coverage
  + Quality Gates
* Integration: [Refer Here](https://directdevops.blog/2019/01/05/sonarqube/) (<https://directdevops.blog/2019/01/05/sonarqube/>)

### Integrating Jenkins with Artifactory

* In CI/CD Process we generate lot of builds.
* To make a customer release/test release we need to pick up specific build and then do the deployment
* To store this binaries, since there are versions we can use binary repository which is aware of build tools like maven, gradle, npm, pip, msbuild
* For Java based repositories, there are two popular binary repository servers
  + Nexus
  + Artifactory
* Artifactory has support not only for java but for almost all the other technologies.
* The use-case from Jenkins is two fold
  + Upload binaries built to artifactory (repository) post build
  + Download binaries from artifactory during deployment. For this artifactory rest api is used or you can use shell script/ansible modules or chef supermarket cookbooks to download.

### Integrating Jenkins with DevOps Tools like Ansible or Docker

* Ensure you add the node to jenkins server with necessary softwares and configurations
* From jenkins after you finish the build and upload the build call the downstream project on the above node added using labels. In the build step execute command

terraform apply .

ansible-playbook .

docker-compose up

kubectl apply -f <spec.yaml>

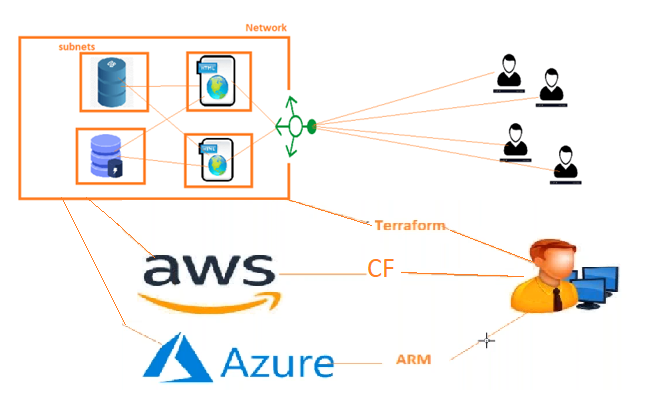
# PACKER

**DECEMBER 31, 2019**

## DevOps Classroom Series – Infra provisioning 31/Dec/2019

### Infra-Provisioning

* Creating Infrastructure for the application.
* Sample Application Architecture



* To realize architecture like the one above, a difficult manual work has to be performed.
* If there is a tool which can help in creating architectures like above, it would be better. To this tool you need to describe your architecture.
* The other challenge is where to create this architecture. Because each virtualization env has different tools
  + AWS => Cloud Formation
  + AZURE => ARM Templates
  + Google => Template (Deployment Manager)
  + VMWare => VMWare SDK
* Advantage of Packer/Terraform is they work on almost all the virtualization platforms

### Packer

* Every Virtualization Platform needs an Image to Create Virtual Machine
* Packer can help in the creation of VM Images in any Virtualization Platform
* Packer Workflow
  + Create a JSON file with your image information and the softwares required
  + Execute Packer to create image in any Virtualization Platform

### Terraform

* Terraform creates Infrastructures like VMs, network, disks
* Using terraform you can call shell script/ansible/chef/powershell
* Terraform creates infrastructure for the applications in any Virtualization Platform
* Terraform Workflow
  + Create a Terraform file (DSL) with your infra needs
  + Execute Terraform to create infra

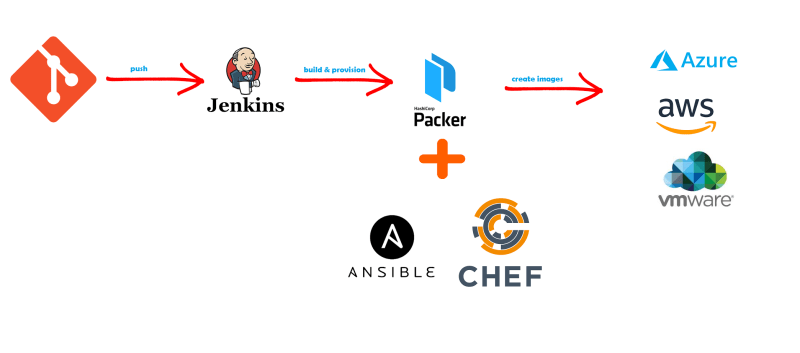
### Pre-requisites

* Azure Free Trail Account
* AWS Free Tier Account

**JANUARY 2, 2020**

## DevOps Classroom Series – 02/Jan/2020 – Packer

### Where Packer Fits in CI/CD



### Softwares Required

* Chocolatey: [Refer Here](https://chocolatey.org/docs/installation)
* Git:

choco install git -y

* Visual Studio Code

choco install vscode -y

### Create a tomcat Server AMI in AWS

* Create an EC2 machine with Ubuntu AMI
* Login into machine and execute the following commands

sudo apt-get update

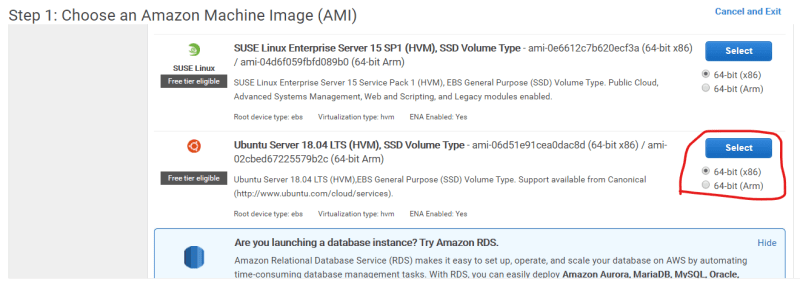
sudo apt-cache search jdk

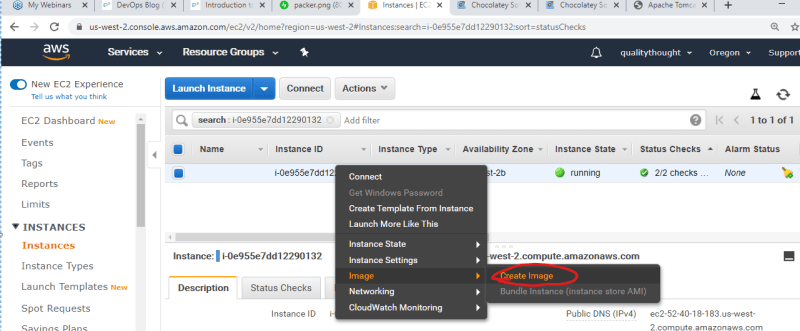
sudo apt-get install openjdk-8-jdk -y

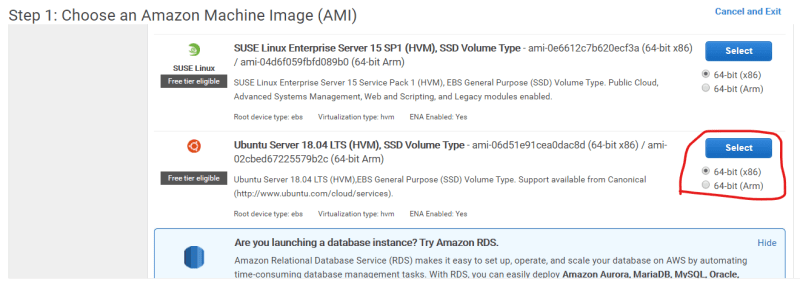
sudo apt-cache search tomcat

sudo apt-get install tomcat8 -y

sudo service tomcat8 status

* Create an AMI of the EC2 Machine
* Image References 





* Important info for creating ami in AWS
  + EC2 machines Image (Source AMI => Ubuntu)
  + EC2 machines size (t2.micro)
  + Region (Oregon)
  + Commands to be executed in ubuntu

### Install Packer

* Chocolatey in windows

choco install packer -y

# Close and reopen powershell

packer --version

packer --help

* [Refer Here](https://www.packer.io/downloads.html) (<https://www.packer.io/downloads.html>)  for direct downloads.

### Packer Json

* Packer Json has sections
  + Builder: Where to create Image
  + Provisioner: Commands to be executed to install software.
  + Post-Processor: any post build activities

### Builders

* Builder is a virtualization env where packer can build an Image. [Refer Here](https://www.packer.io/docs/builders/index.html)

### Provisioners

* Used for installing software / packages etc. [Refer Here](https://www.packer.io/docs/provisioners/index.html) (<https://www.packer.io/docs/provisioners/index.html>) for full list of Provisioners

**JANUARY 3, 2020**

## DevOps Classroom Series – 03/Jan/2020 – Packer

### Creating AWS Image From Packer

* To create AWS Image from Packer we need some info
  + AWS Account:
    - Create Secret Key and Access Key for Packer (IAM User)
  + Source Image AMI
  + Provisioning Script
  + Region
* Create a folder and a json file with the following content

{

"builders": []

}

* Lets create access key and secret key as mentioned over [here](https://serverless-stack.com/chapters/create-an-iam-user.html) (<https://serverless-stack.com/chapters/create-an-iam-user.html>)
* Fill all the required Fields from amazon-ebs builder as shown below

{

"builders": [

{

"type": "amazon-ebs",

"ami\_name": "tomcatfrompacker",

"ami\_description": "tomcat from packer",

"access\_key": "",

"secret\_key": "",

"region": "us-west-2",

"instance\_type": "t2.micro",

"source\_ami": "ami-06d51e91cea0dac8d"

}]

}

* Now execute the command packer validate aws.json and you should observe an error about ssh\_username add ssh\_username to the builder section

{

"builders": [

{

"type": "amazon-ebs",

"ami\_name": "tomcatfrompacker",

"ami\_description": "tomcat from packer",

"access\_key": "",

"secret\_key": "",

"region": "us-west-2",

"instance\_type": "t2.micro",

"source\_ami": "ami-06d51e91cea0dac8d",

"ssh\_username": "ubuntu"

}]

}

* Now lets build the image using debug build option

packer build -debug aws.json

* Now manually deregister ami after image creation is success.
* Now lets add provisioner. I have set of manual linux commands to be executed and i will be using a shell provisioner [Refer Here](https://www.packer.io/docs/provisioners/shell.html) (<https://www.packer.io/docs/provisioners/shell.html>)
* Lets add provisioners section

{

"builders": [

{

"type": "amazon-ebs",

"ami\_name": "tomcatfrompacker",

"ami\_description": "tomcat from packer",

"access\_key": "AKIA3TXJQGAJFDVFHD7C",

"secret\_key": "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",

"region": "us-west-2",

"instance\_type": "t2.micro",

"source\_ami": "ami-06d51e91cea0dac8d",

"ssh\_username": "ubuntu"

}],

"provisioners": [

{

"type": "shell",

"inline": [

"sudo apt-get update",

"sudo apt-get install openjdk-8-jdk -y",

"sudo apt-get install tomcat8 -y"

]

}

]

}

* Try building image using packer build

**JANUARY 4, 2020**

## DevOps Classroom Series – Packer – 04/Jan/2020

### Provisioning in Packer

* Shell Provisioning:
  + Executes Linux commands or shell script(s) once the VM is created.
* Ansible Provisioning:
* Chef Provisioning

### Applying Shell Provisioner with script

* Create a file called as installtomcat.sh in the same directory as aws.json
* Add the following content

#!/bin/bash

sudo apt-get update

sleep 10

sudo apt-get install openjdk-8-jdk -y

sleep 10

sudo apt-get install tomcat8 -y

sleep 10

* Change the shell provisioner from inline to script as shown below

{

"builders": [

{

"type": "amazon-ebs",

"ami\_name": "tomcatfrompacker",

"ami\_description": "tomcat from packer",

"access\_key": "AKIA3TXJQGAJFDVFHD7C",

"secret\_key": "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",

"region": "us-west-2",

"instance\_type": "t2.micro",

"source\_ami": "ami-06d51e91cea0dac8d",

"ssh\_username": "ubuntu"

}],

"provisioners": [

{

"type": "shell",

"script": "./installtomcat.sh"

}

]

}

* Note: [Refer Here](https://www.packer.io/docs/provisioners/shell.html) (<https://www.packer.io/docs/provisioners/shell.html>) for complete info on shell provisioner.

### Variables in Packer

* Variables in Packer give the flexibility of changing values used in builders/provisioners
* For variables a section called as **varaibles** is available in Packer.
* Variables defined in the Packer Template are called as user variables
* Basic Structure of packer template with variables is as shown below

{

"variables": {

},

"builders": [],

"provisioners": []

}

* To define a variables simple add a key pair/name pair to json

{

"variables": {

"myvar": "hello"

},

"builders": [],

"provisioners": []

}

* To use myvar variable in bulders or provisioners section use the following syntax

"{{user `myvar`}}"

* To the above packer template if we add variables for access, secret keys and region, source ami the template looks as follows

{

"variables": {

"aws\_access\_key": "",

"aws\_secret\_key": "",

"source\_image\_id": "ami-06d51e91cea0dac8d",

"image\_region": "us-west-2"

},

"builders": [

{

"type": "amazon-ebs",

"ami\_name": "tomcatfrompacker",

"ami\_description": "tomcat from packer",

"access\_key": "{{user `aws\_access\_key`}}",

"secret\_key": "{{user `aws\_secret\_key`}}",

"region": "{{user `image\_region`}}",

"instance\_type": "t2.micro",

"source\_ami": "{{user `source\_image\_id`}}",

"ssh\_username": "ubuntu"

}],

"provisioners": [

{

"type": "shell",

"script": "./installtomcat.sh"

}

]

}

* To validate the template use the -var argument

packer validate -var 'aws\_access\_key=<accesskey>' -var 'aws\_secret\_key=<secretkey>' .\aws.json

* To build the image from template

packer build -var 'aws\_access\_key=<accesskey>' -var 'aws\_secret\_key=<secretkey>' .\aws.json

* If you are using many variables -var-file option is available. Create a Json file with all variable names and values and use that file rather than individual -var parameters

packer validate -var-file myvariables.json ./aws.json

packer build -var-file myvariables.json ./aws.json

**JANUARY 5, 2020**

## DevOps Classroom Series – Packer – 05/Jan/2020

### Azure VM Image from Packer

### Manual Creation

* Create Resource Group
* Create a VM with some source image (ubuntu)
* SSH into the machine and execute

sudo apt-get update

sudo apt-get install openjdk-8-jdk -y

sudo apt-get install tomcat8 -y

### Using Packer

* [Refer Here](https://docs.microsoft.com/en-us/azure/virtual-machines/linux/build-image-with-packer) (<https://docs.microsoft.com/en-us/azure/virtual-machines/linux/build-image-with-packer>)

### Variables

* Environment Variables:
  + Define environment variable in your system and then to use it {{env `PATH`}}

### Useful Functions in Packer

* isotime
* clean\_resource\_name
* [Refer Here](https://www.packer.io/docs/templates/engine.html) (<https://www.packer.io/docs/templates/engine.html>)

# TERRAFORM

**JANUARY 7, 2020**

## DevOps Classroom Series – 06/Jan/2020 – Terraform

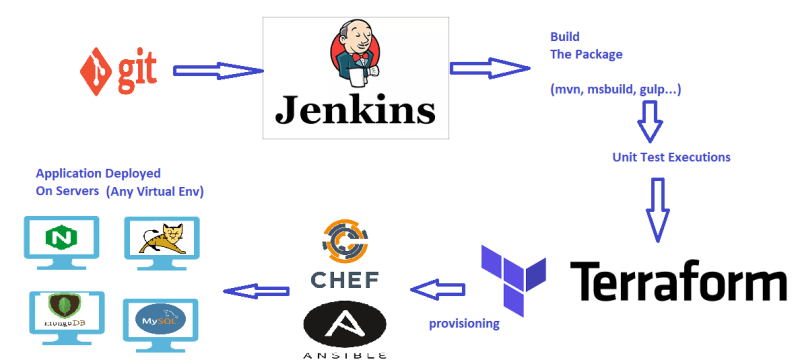
### Terraform

* Terraform helps in IaC (Infrastructure as Code)
* From Terraform, Infrastructure can be created in almost all the virtual environments (AWS, Azure, VmWare, VirtualBox etc)
* Terraform is developed in GO Language.
* Like Packer, Terraform is also a single executable.
* Terraform templates are written in Custom DSL. This DSL mostly looks like JSON.
* Terraform templates end with .tf extension

### Installation

* Windows:
  + choco install terraform
* Linux:
  + Download the terraform from [here](https://www.terraform.io/downloads.html) (<https://www.terraform.io/downloads.html>)
  + unzip the file to some folder. Add this folder to PATH environment variable (use export command and add this statement to /etc/environment or .bashrc)

### Terraform in CD Pipelines (DevOps)



### Concepts

* Terraform has list of providers supported. Each Provider provides resources and data sources.
* Provider: Provider is used to create infrastructure in particular virtual environments. [Refer Here](https://www.terraform.io/docs/providers/index.html)(<https://www.terraform.io/docs/providers/index.html>)  for the complete list. Some of the popular providers are
  + AWS
  + Azure
  + OpenStack
  + VmWare
* Resource: Resource is part of the infrastructure create on the provider. Every Provider provides resources.
* DataSource: <will be defining soon. not now>
* Variables: To give options for the user to enter different values to resources.
* Provisioners: Terraform supports various provisioners like shell, Chef, Powershell

**JANUARY 7, 2020**

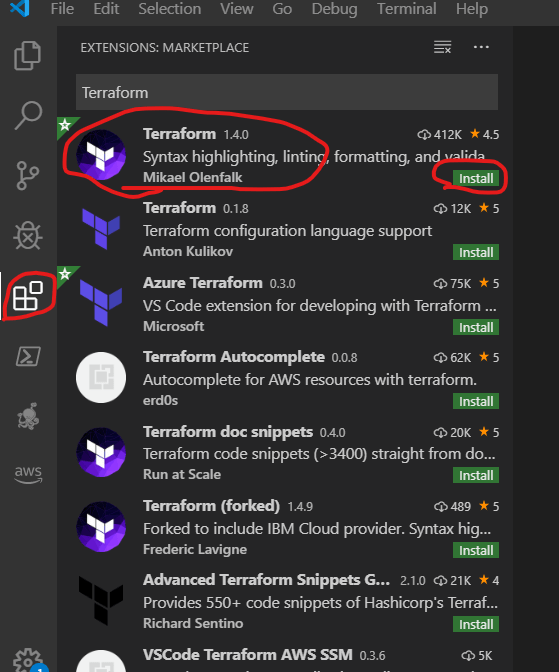
## DevOps Classroom Series – 07/Jan/2020 – Terraform

### Terraform Example for creating simple AWS Resource

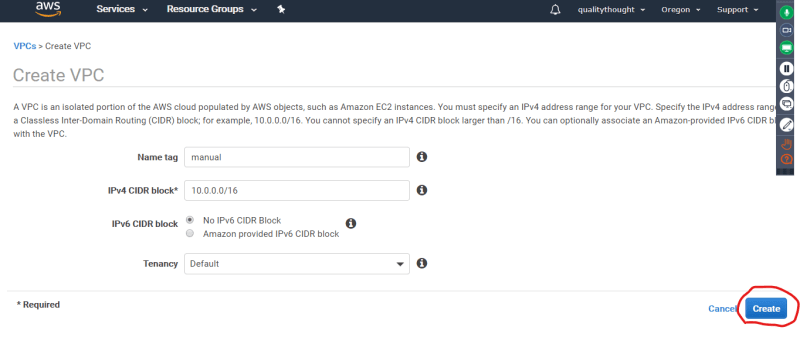
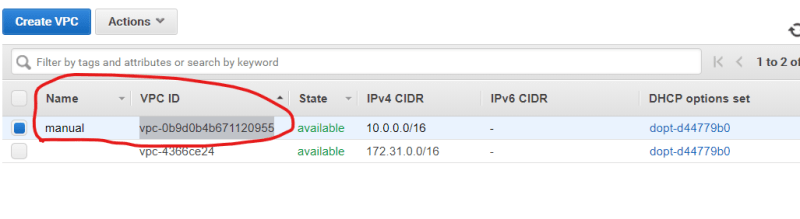
* Service: What cloud provider is offering you
* Resource: Anything created using Service

### Setup Terraform DEV Environment

* Install Terraform
* Install Visual Studio Code
* Install Terraform Extension into Visual Studio Code



### Manual: Create a VPC in oregon Region

### Terraform: Create a VPC in oregon Region

* Create a new directory **hello-tf** as folder is input to terraform
* Create a new file called as **main.tf**
* To create resources in aws, use aws provider. Provider syntax is

provider 'provider type' {

arg1 'value1'

...

...

argn 'valuen'

}

* Using The above syntax and the documentation over [here](https://www.terraform.io/docs/providers/aws/index.html), lets fill the aws provider

provider "aws" {

access\_key = "<ACCESS-KEY>"

secret\_key = "<SECRET-KEY>"

region = "us-west-2"

}

* Note: Inputs in Terraform are called as **arguments** and outputs are called as **attributes**.
* Now we need to create a VPC Resource. Search the Resources in Documentation page or just google ‘terraform resource <cloud> <resource>’
* Resource Syntax is

resource "type" "name" {

arg1 'value1'

...

...

argn 'valuen'

}

* Lets add the VPC resource details to existing main.tf file

provider "aws" {

access\_key = "<ACCESS-KEY>"

secret\_key = "<SECRET-KEY>"

region = "us-west-2"

}

resource "aws\_vpc" "myvpc" {

cidr\_block = "192.168.0.0/16"

tags = {

"Name" = "from-tf"

}

}

Now to execute this terraform template, Launch Powershell in the directory where terraform template is written

terraform init .

terraform validate .

terraform apply .

* Once you are done resource can be removed using

terraform destroy .

**JANUARY 8, 2020**

## DevOps Classroom Series – 08/Jan/2020

### Resource Dependencies

* When Resource A requires resource B to be present (or already existing) this is called as dependency
* In Terraform terms you have to create resource B before resource A
* To Demonstrate this lets add subnets to VPC. To create subnet resource vpc id is required (vpc has to be existing)
* To create resource dependencies use the following expression

"${<resource-type>.<resource-name>.<attribute-name>}"

"${aws\_vpc.myvpc.id}"

* If we apply the above expressions to create two subnets in the vpc

provider "aws" {

access\_key = "<accesskey>"

secret\_key = "<secret-key>"

region = "us-west-2"

}

resource "aws\_vpc" "myvpc" {

cidr\_block = "192.168.0.0/16"

tags = {

"Name" = "from-tf"

}

}

resource "aws\_subnet" "subnet1" {

cidr\_block = "192.168.0.0/24"

vpc\_id = "${aws\_vpc.myvpc.id}"

availability\_zone = "us-west-2a"

tags = {

"Name" = "subnet-1"

}

}

resource "aws\_subnet" "subnet2" {

cidr\_block = "192.168.1.0/24"

availability\_zone = "us-west-2b"

tags = {

"Name" = "subnet2"

}

vpc\_id = "${aws\_vpc.myvpc.id}"

}

* Now Execute terraform apply to create infra and also observe the folder where **terraform.tfstate** is created where the information about created resources are stored.
* Now Lets add one more subnet to existing template

provider "aws" {

access\_key = "<accesskey>"

secret\_key = "<secret-key>"

region = "us-west-2"

}

resource "aws\_vpc" "myvpc" {

cidr\_block = "192.168.0.0/16"

tags = {

"Name" = "from-tf"

}

}

resource "aws\_subnet" "subnet1" {

cidr\_block = "192.168.0.0/24"

vpc\_id = "${aws\_vpc.myvpc.id}"

availability\_zone = "us-west-2a"

tags = {

"Name" = "subnet-1"

}

}

resource "aws\_subnet" "subnet2" {

cidr\_block = "192.168.1.0/24"

availability\_zone = "us-west-2b"

tags = {

"Name" = "subnet2"

}

vpc\_id = "${aws\_vpc.myvpc.id}"

}

resource "aws\_subnet" "subnet3" {

cidr\_block = "192.168.2.0/24"

availability\_zone = "us-west-2c"

tags = {

"Name" = "subnet3"

}

vpc\_id = "${aws\_vpc.myvpc.id}"

}

* Now if you execute **terraform apply** command
  + Terraform will get the current status from provider (aws)
  + It compares the status of provider with tfstate file, if any differences are found they will be added for execution. (Deleting from aws and executing terraform apply)
  + during apply plan will be created and compared to current status, if plan is not matching the current state, then the changes will be added to terraforms execution.
* Experiment Terraform
  + By deleting one subnet from template
  + change the tag value for one subnet
  + change the avaliability zone for one subnet

**JANUARY 9, 2020**

## DevOps Classroom Series – 09/Jan/2020

### Terraform file organization

* Since writing every thing in one file looks complex and unreadable, terraform allows you to write resources, providers etc in multiple .tf files
* Lets organize the work into multiple files

touch provider.tf

touch network.tf

* In Provider.tf add the following

provider "aws" {

access\_key = "<ACCESS-KEY>"

secret\_key = "<SECRET-KEY>"

region = "us-west-2"

}

* In network.tf add the following

resource "aws\_vpc" "myvpc" {

cidr\_block = "192.168.0.0/16"

tags = {

"Name" = "from-tf"

}

}

resource "aws\_subnet" "subnet1" {

cidr\_block = "192.168.0.0/24"

vpc\_id = "${aws\_vpc.myvpc.id}"

availability\_zone = "us-west-2a"

tags = {

"Name" = "subnet-1"

}

}

resource "aws\_subnet" "subnet2" {

cidr\_block = "192.168.1.0/24"

availability\_zone = "us-west-2b"

tags = {

"Name" = "subnet2"

}

vpc\_id = "${aws\_vpc.myvpc.id}"

}

resource "aws\_subnet" "subnet3" {

cidr\_block = "192.168.2.0/24"

availability\_zone = "us-west-2c"

tags = {

"Name" = "subnet3"

}

vpc\_id = "${aws\_vpc.myvpc.id}"

}

* Now execute terraform apply .

### Adding some stuff to existing network

* Add internet gateway to vpc and a public route table. Create a new tf file and add the following content to existing folder

resource "aws\_internet\_gateway" "igw" {

vpc\_id = "${aws\_vpc.myvpc.id}"

tags = {

"Name" = "from tf"

}

}

resource "aws\_route\_table" "publicrt" {

vpc\_id = "${aws\_vpc.myvpc.id}"

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = "${aws\_internet\_gateway.igw.id}"

}

tags = {

"Name" = "public rt-tf"

}

}

* Lets create a plan by executing

terraform plan --out="aws.plan" .

* Now execute the plan by using terraform apply aws.plan

### Variable support to Terraform

* Create a new file input.tf and add variables as described [over here](https://www.terraform.io/docs/configuration/variables.html)

variable "accesskey" {

type = "string"

}

variable "secretkey" {

type = "string"

}

variable "region" {

type = "string"

default = "us-west-2"

}

* Now lets use these varaibles in the provider section

provider "aws" {

access\_key = var.accesskey

secret\_key = var.secretkey

region = var.region

}

* Now Create a plan and extecute using

terraform plan -var 'accesskey=<your-access-key>' -var 'secretkey=<your-secret-key>' -out='aws.plan' .

terraform apply aws.plan

* Add variables for vpc cidr, subnet cidr

variable "subnet1cidr" {

type = "string"

default = "192.168.0.0/24"

}

variable "subnet2cidr" {

type = "string"

default = "192.168.1.0/24"

}

variable "subnet3cidr" {

type = "string"

default = "192.168.2.0/24"

}

* Resources will be looking like

resource "aws\_subnet" "subnet1" {

cidr\_block = var.subnet1cidr

vpc\_id = "${aws\_vpc.myvpc.id}"

availability\_zone = "us-west-2a"

tags = {

"Name" = "subnet-1"

}

}

resource "aws\_subnet" "subnet2" {

cidr\_block = var.subnet2cidr

availability\_zone = "us-west-2b"

tags = {

"Name" = "subnet2"

}

vpc\_id = "${aws\_vpc.myvpc.id}"

}

resource "aws\_subnet" "subnet3" {

cidr\_block = var.subnet3cidr

availability\_zone = "us-west-2a"

tags = {

"Name" = "subnet3"

}

vpc\_id = "${aws\_vpc.myvpc.id}"

}

**JANUARY 10, 2020**

## DevOps Classroom Series – 10/Jan/2020 – Terraform

### terraform cli

* refresh: is used to update the local state (\*.tfstate) with resources created.

terraform refresh -var 'accesskey=<youraccesskey>' -var 'secretkey=<your-secret-key>' .

* taint: is used to mark resources for recreation during next apply

terraform taint <resource-type>.<resource-name>

terraform apply .

# example

terraform taint aws\_subnet.subnet3

# subnet 3 will be marked as tainted

terraform apply -var 'accesskey=<youraccesskey>' -var 'secretkey=<your-secret-key>' .

# since subnet3 is marked as tainted, it will be recreated.

* untaint: if you have tainted any resource by mistake/for any other reason and if you want it to be removed from taint then use untaint.
* graph: Generate a graph from your template. Ensure you install dot from graphviz. [Refer Here](https://www.graphviz.org/download/)

terraform graph | dot -Tsvg > graph.svg

### Terraform outputs

* Output is result of the infra provisioning which can be shared
* To Create outputs, create a new file called as outputs.tf with following content

output "vpc-id" {

value = "${aws\_vpc.myvpc.id}"

}

output "subnet1-id" {

value = "${aws\_subnet.subnet1.id}"

}

* Now execute terraform apply and observe the output

### Terraform modules

* Module is reusable terraform configuration.
* Terraform community shares many modules for resuse in [Terraform Registry](https://registry.terraform.io/)
* Terraform templates written by us can also be reused as modules.

### Using Terraform module from Local folder

* Create a new directory called as moduledemo
* in the module demo create one file main.tf with following content

module "<name>" {

source = "../hello-tf"

accesskey = ""

secretkey = ""

}

* Now execute

terraform init

terraform apply .

* In hello-tf all the variables will be arguments to module and all the outputs will be attributes of module.
* Any varaible without default is required argument and variable with default is optional argument.

### Next Sections

* Terraform Registry example
* Terraform remote-state
* terraform env and workspaces
* Terraform import
* Terraform example with Azure.

**JANUARY 20, 2020**

## DevOps Classroom Series – 20/Jan/2020

### Data-Sources

* For creating ec2 machine, we need to provide
  + subnet-id
  + security group
  + key value pair
* One approach create every thing and use the attributes.
* If we want to use existing subnet-id, security group and key-value pair, we need to know ids, for this terraform has data sources which can query the information from providers.
* Every provider gives various data sources much like resources.
* Lets create a simple data source to pull the information of default subnet and create a new subnet

data "aws\_vpc" "default" {

default = true

}

resource "aws\_subnet" "extra" {

cidr\_block = "172.31.48.0/20"

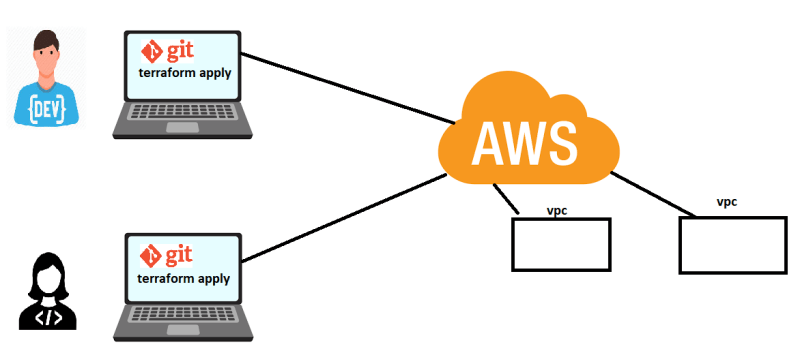
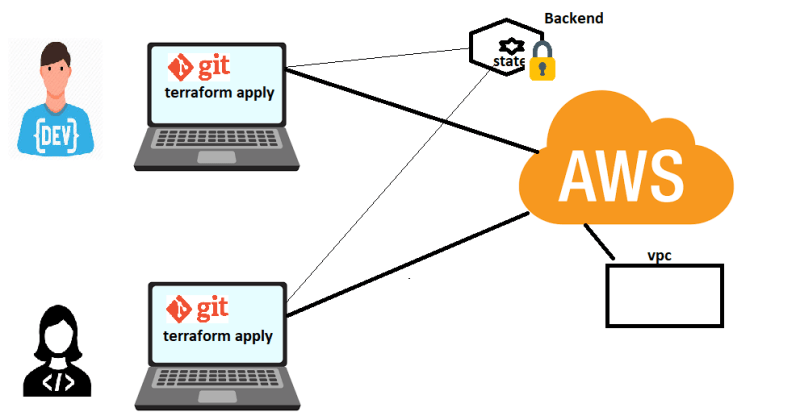
vpc\_id = "${data.aws\_vpc.default.id}"

}

### Terraform provisioning

* Execution of scripts/ansible/chef after creation of Virtual Machines is supported by terraform provisioners. [Refer](https://www.terraform.io/docs/provisioners/index.html) (<https://www.terraform.io/docs/provisioners/index.html>)

### Backends

* Two tf developers have same terraform script and they have applied terraform, it creates two different resources, as the state file is stored on individual developers laptop 
* Now, if we want to restrict these two developers in such a way, whenever they execute terraform it should not create two different but one resource.
* Terraform supports backends, to store state remotely and terraform also supports locking feature to avoid simultaneous access to terraform state. 
* [Refer](https://www.terraform.io/docs/backends/index.html) (<https://www.terraform.io/docs/backends/index.html>)

**JANUARY 21, 2020**

## DevOps Classroom Series – 21/Jan/2020

### Terraform Provisioning

* Provisioning in Terraform is majorly used to do Configuration Management
* Terraform supports many provisioners, some of them are
  + remote-exec
  + local-exec
  + Chef
* For Ansible based CM use remote-exec to install ansible and execute ansible-playbook with localhost in inventory
* To use provisioners, you should also use connection
* Provisioners are written in terraform resources

resource '<resource-type>' '<resource-name>' {

connection {

...

}

provisioner 'p-type' {

}

}

* Lets create a ec2 machine and install git & apache
* First create ec2 machine

resource "aws\_instance" "apache" {

ami = "ami-04590e7389a6e577c"

instance\_type = "t2.micro"

key\_name = "packer"

security\_groups = ["sshonly"]

}

* Now lets add the connection section for terraform to connect to ec2 machine

resource "aws\_instance" "apache" {

ami = "ami-04590e7389a6e577c"

instance\_type = "t2.micro"

key\_name = "packer"

security\_groups = ["sshonly"]

connection {

type = "ssh"

user = "ec2-user"

host = "${aws\_instance.apache.public\_ip}"

private\_key = "${file("./packer.pem")}"

}

}

* Now lets provision using remote-exec to install git & apache

resource "aws\_instance" "apache" {

ami = "ami-04590e7389a6e577c"

instance\_type = "t2.micro"

key\_name = "packer"

security\_groups = ["sshonly"]

connection {

type = "ssh"

user = "ec2-user"

host = "${aws\_instance.apache.public\_ip}"

private\_key = "${file("./packer.pem")}"

}

provisioner "remote-exec" {

inline = ["sudo yum install git -y", "sudo yum install httpd -y"]

}

}

### Backends

* To use Azurerm Backend [refer here](https://docs.microsoft.com/en-us/azure/terraform/terraform-backend) (<https://docs.microsoft.com/en-us/azure/terraform/terraform-backend>)
* To use Aws Backend [Refer Here](https://directdevops.blog/2019/01/27/terraform-backends/) (<https://directdevops.blog/2019/01/27/terraform-backends/>)

### Exercise

* Write a terraform template to create two vms (ec2 or azure vms) and install tomcat in one vm and mongo db (any database) in other vm

# AZURE DEVOPS

**JANUARY 26, 2020**

## DevOps Classroom Series – 25/Jan/2020 (Azure DevOps)

### Azure DevOps (Visual Studio Team Services)

* Is collection of Services/Tools to perform whole Project Management (majorly Agile)
* [Refer Here](https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops) (<https://docs.microsoft.com/en-us/azure/devops/?view=azure-devops>) for azure devops official documentation
* Azure DevOps can be used in two modes
  + Azure Cloud (Zero Installation/Hosted)
  + Azure DevOps Server (Self Hosted)
* Azure DevOps Services
  + [Refer Here](https://docs.microsoft.com/en-us/azure/devops/user-guide/services?view=azure-devops) (<https://docs.microsoft.com/en-us/azure/devops/user-guide/services?view=azure-devops>)

### VSTS Terms Equivalent in Jenkins

* Master => Azure DevOPs Server
* Node => Agent
* Free Style Project => Classic UI
* Jenkins File (Groovy) => Pipeline Spec (YAML)

# DOCKER

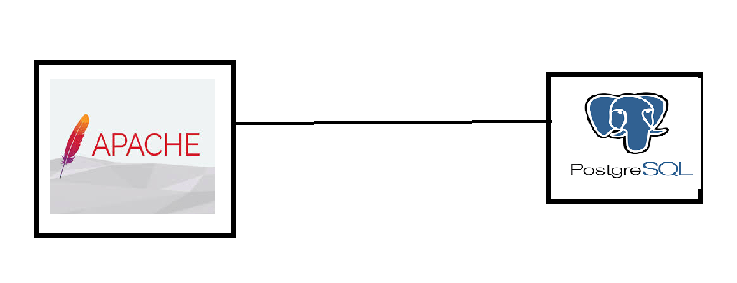
**JANUARY 23, 2020**

## DevOps Classroom Series – 23/Jan/2020

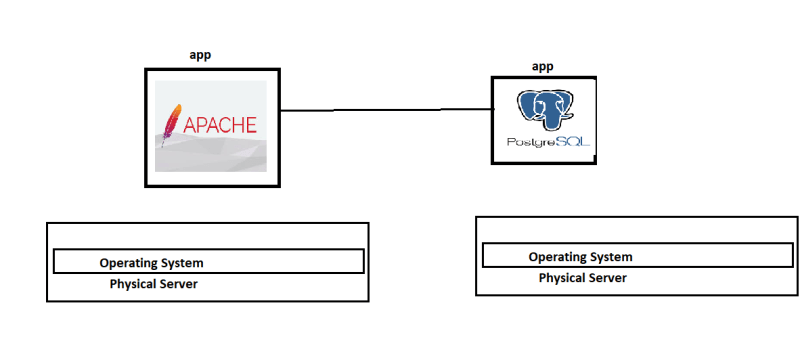
### Docker

### Evolution

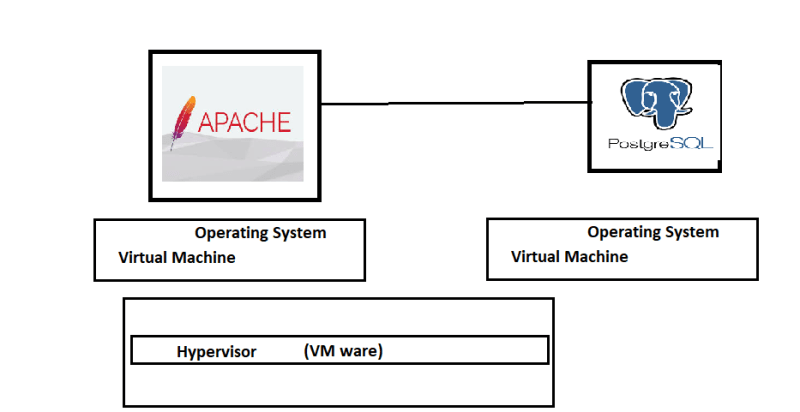
* Scenario: Running a simple web application



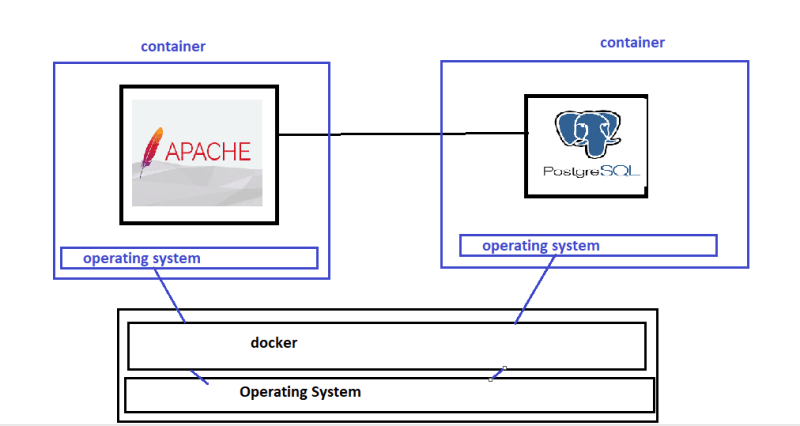
#### Solution-1: Physical Servers

* Procure 2 Physical Servers with decent configuration (higher than needed) 
* Cons:
  + Cost
  + Under usage of Server Resources

#### Solution 2: Operating System Virtualization

* Install Hypervisors like VMWare on the physical servers and create virtual machines
* Better usage of Server Resources 
* Cons:
  + Still OS overhead is involved

#### Solution 3: Container

* Install Container tech like docker/lxc/lxd and run the applications inside containers.
* Container is an isolated area, with all the necessary resources available to run the application
* With containers scaling and portability becomes extremely simple. 

**JANUARY 24, 2020**

## DevOps Classroom Series – 24/Jan/2020

### Microservices Deployments

* Running each microservice on Container is possibly the best option to deploy microservice based applications
* Advantages:
  + Individual Microservice can be scaled

### Container

* Todays Definition: Container is an isolated area which has every thing to run the application

### Role in Docker as a DevOps Engineer

1. First expectation is to do something to make your application run inside container (Contenarization).

### First Steps towards Containerization

* Installing Docker
  + At this moment lets not install docker but use [Docker Playground](https://labs.play-with-docker.com/) (<https://labs.play-with-docker.com/>)
  + Create an account in [Docker Hub](https://hub.docker.com/) (<https://hub.docker.com/>)
  + Run the below commands

docker info

docker container ls

docker image ls

docker run hello-world

* Understanding Docker Image and Docker Container

**JANUARY 25, 2020**

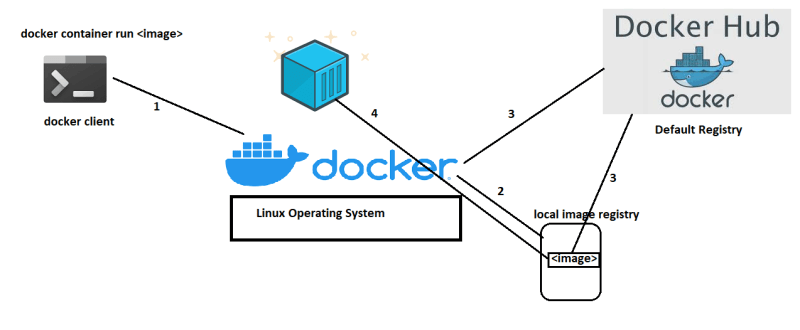
## DevOps Classroom Series – 25/Jan/2020

### Docker Images and Registry

### Components of Docker

* Docker will have two major components
  + Client
  + Daemon (Docker Engine)

### What happens when we execute docker container run <image>



1. docker client will request the daemon
2. daemon will check for the image in the local registry
3. If not found in the local registry, then the image is downloaded from default registry to local registry (pull)
4. Using the image in the local registry, a container is created.

### How to create Docker Image?

* Converting existing containers into images
* Exporting VMS to Docker Images
* Creating Docker Image using Dockerfile (Instruction based Approach)

### Instruction Based Approach to be followed.

* In this we will make a note of deployment steps for the application to be run inside container.
* Learn mapping these steps to Dockerfile Instructions

**JANUARY 27, 2020**

## DevOps Classroom Series – 27/Jan/2020

### Manual Creation of Apache Server

* Scenario: Creating an Apache Server
* Sol:
  + Create an ubuntu server
  + ssh into ubuntu server
  + execute the following commands

sudo apt-get update

sudo apt-get install apache2 -y

Browse to http://<publicip&gt;

### Creating a Docker Image for Apache Server

* Approach:
  + Search for ubuntu in Docker hub to choose **base image**
  + Execute some commands while building apache image (to convert base image into apache server image)

sudo apt-get update

sudo apt-get install apache2 -y

* Solution:
  + Create a file with name **Dockerfile** and add following contents

FROM ubuntu

RUN "apt-get update"

RUN "apt-get install apache2 -y"

* + If the above doesn’t work then try below

FROM ubuntu

ARG DEBIAN\_FRONTEND=noninteractive

RUN apt-get update

RUN apt-get install apache2 -y

* + Using this Dockerfile build a docker image using

docker image build -t myapache .

### Installing Docker on Linux Platforms

* Simple Approach:
  + Always installs the latest version

curl -fsSL https://get.docker.com -o get-docker.sh

sh get-docker.sh

* + add user to the Docker Group

sudo usermod -aG docker <username>

* + Logout and login into the linux machine

docker info

### Manual Creation of Spring-Petclinic (Spring Boot)

* Install JAVA
* Download Spring pet clinic [from here](https://war-jar-files.s3-us-west-2.amazonaws.com/spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar)
* Run the command java -jar spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar

### Docker image creation for Spring-Petclinic

* Find a base-image with java already installed. In this example i would be using openjdk:8
* Dockerfile

FROM openjdk:8

RUN wget https://war-jar-files.s3-us-west-2.amazonaws.com/spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar

EXPOSE 8080

CMD ["java", "-jar", "spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar"]

* Build the image

docker image build -t spc .

docker image ls

* Create the container from image

docker container run -d -p 8081:8080 spc

docker container ls

* Navigate petclinic using http://<vmip&gt;:8081

### Dockerfile Instructions

* Instructions are phrases in the Dockerfile with some Behaviors
* For complete list of instructions [refer here](https://docs.docker.com/engine/reference/builder/) (<https://docs.docker.com/engine/reference/builder/>)
* **FROM** => to choose the base image
* **RUN** => execute the command while building the image
* **EXPOSE** => to expose ports used by container
* **CMD** => command to be executed while container creation

**JANUARY 28, 2020**

## DevOps Classroom Series – 28/Jan/2020

### Importance of Tags in Docker

* Format is <image-name>:<tag>
* Ex: openjdk:8, jenkins:1.609.2
* In docker if the tag is not specified, **LATEST** is applied

FROM tomcat === FROM tomcat:LATEST

* note: in FROM instruction always specify tag and also while building images

### Dockerfile instructions

#### Entrypoint

* Create a Dockerfile with some cmd

FROM openjdk:8

RUN wget https://war-jar-files.s3-us-west-2.amazonaws.com/spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar

EXPOSE 8080

CMD ["java", "-jar", "spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar"]

* create a docker image (for tag use ddmmyy)

docker image build -t spc:280120 .

docker image build -t spc:1.0-280120

* list the docker images

docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

spc 1.0-280120 4c0dd4eb112b 59 seconds ago 536MB

spc 280120 4c0dd4eb112b 59 seconds ago 536MB

openjdk 8 8c6851b1fc09 5 days ago 488MB

# for spc:280120 and 1.0-280120 we have same image id, same image id means same image to docker

* To create a container

docker container run -d -p 8080:8080 spc:280120

# creates a new container and runs CMD from Dockerfile

docker container run -d -p 8081:8080 spc:280120 echo helloworld

# Creates a new container and CMD in Dockerfile will be overriden with echo helloworld

* Entrypoint instruction also executes when creating a new container, the differnce with CMD is ENTRYPOINT cannot be overridden.
* When you have written both ENTRYPOINT and CMD, ENTRYPOINT will be executable and CMD will be arguments

FROM openjdk:8

ENTRYPOINT ["echo"]

CMD ["hello"]

* Now if we run a container after creating the image (demo)

docker container run image

# hello

docker container run image how are you

how are you

* Containers will be alive as long as the command in ENTRYPOINT and CMD instruction is executing

#### Other instructions

* LABEL: This instruction adds metadata

FROM openjdk:8

LABEL author=khaja

LABEL org=qt

* ADD and COPY:
  + These instructions are used to copy the external files (file from host or from http(s)) into the docker image
  + ADD and COPY both takes two arguments source and destination
  + ADD can copy the contents from file system and URLs
  + COPY can copy the contents only from file system

FROM openjdk:8

ADD https://war-jar-files.s3-us-west-2.amazonaws.com/spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar /spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar

EXPOSE 8080

CMD ["java", "-jar", "spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar"]

**JANUARY 29, 2020**

## DevOps Classroom Series – 29/Jan/2020

### Docker references

* [Refer here](https://directdevops.blog/devops-blog-imported-from-qt-blog/) (<https://directdevops.blog/devops-blog-imported-from-qt-blog/>)

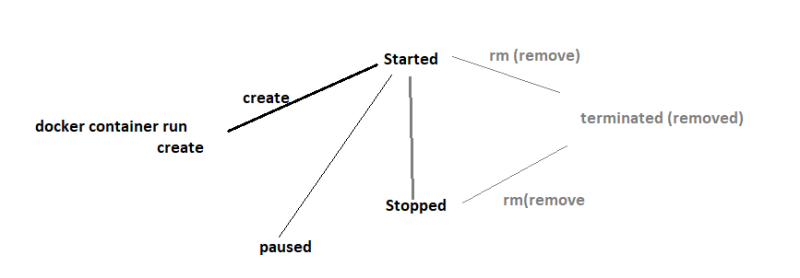
### Docker Internals

* [Refer Here](https://directdevops.blog/2019/01/31/docker-internals/) (<https://directdevops.blog/2019/01/31/docker-internals/>)

**JANUARY 30, 2020**

## DevOps Classroom series – 30/Jan2020

### Container States

* Running (Started)
* Exited (Stopped)
* Paused 

### Docker Container Creation modes

* Attached mode:
  + Our terminal will recieve outputs from containter stdout, stderr

docker container run jenkins

# press ctrl+c to exit the container and also the container outputs to your terminal

* Detached Mode (background)
  + Container runs in the background and you will recieve container id

docker container run -d jenkins

* Interactive Terminal
  + Create a container and login into interactive shell

docker container run -it jenkins /bin/bash

docker container run -it alpine /bin/sh

### Listing Containers

* docker container ls or docker ps will show all the containers in running state
* docker container ls -a or docker ps -a will show all the containers in running and stopped state

### Removing Containers

* Containers can be removed from stopped state, if they are in running state you have to force the delete
* To remove containers we need name/id

docker container rm <name-or-id>

docker container rm <name-or-id-container1> <name-or-id-container2> ..

* To remove all the containers

docker container rm -f $(docker container ls -a -q)

### Docker CLI

* Cheatsheet
* use –help

docker container run --help

* Use Docker Documentation [from here](https://docs.docker.com/engine/reference/run/) (<https://docs.docker.com/engine/reference/run/>)

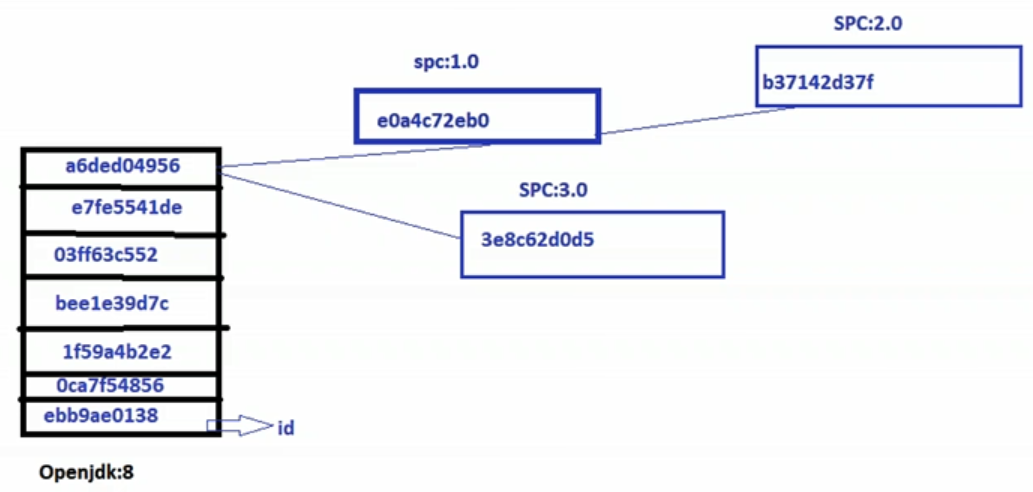
**JANUARY 31, 2020**

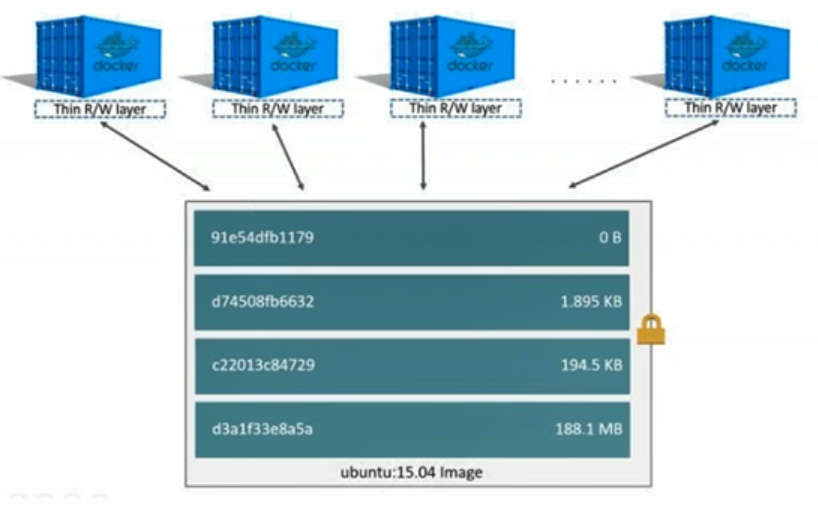
## DevOps Classroom Series – 31/Jan/2020

### Image Layers

* [Refer Here](https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/) (<https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/>)  for official docs.

Example:





### Commands used

docker inspect <image-id> or <image-name>

**FEBRUARY 2, 2020**

## DevOps Classroom Series – 02/Feb/2020 - Morning

### Docker Storage Driver

* [Refer](https://directdevops.blog/2019/09/27/impact-of-image-layers-on-docker-containers-storage-drivers/) (<https://directdevops.blog/2019/09/27/impact-of-image-layers-on-docker-containers-storage-drivers/>)  to storage drivers section in the article.

### Docker Volumes

* Volume adds persistence to the docker container
* Volumes add disk sharing across containers and also between container and host
* Volume life cylce is different from containers life cycle
* Example Scenario:
  + Create a docker container with mysql running:
    - For the containers which generate/store data it is important to preserve it, so we have to use volumes
* Application Categories:
  + Stateless Applications:
    - Application which donot store any state and rely on other applications to store/preserve the data
  + Stateful Application
    - Applications which store the data/state locally and for these applications the data has to preserved by us using volumes.
* For official Docs [Refer Here](https://docs.docker.com/storage/volumes/) (<https://docs.docker.com/storage/volumes/>)

### Docker Volume Command line

* Docker has created a sepearte command line for volume
* Volume command line

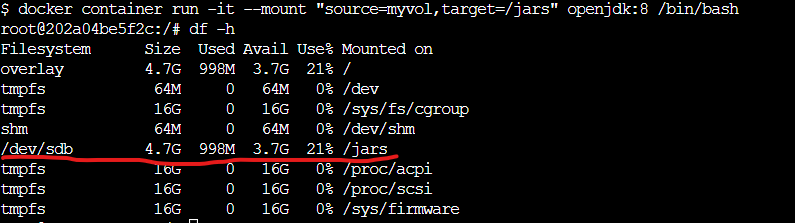
docker volume create myvol

docker volume ls

docker volume inspect myvol

* Creating the container and mounting the volume

docker container run -it --name cont1 --mount "source=myvol,target=/jars" openjdk:8 /bin/bash

* Check for the mounts available inside container using df -h 
* Now lets experiment with generating data inside the volume, remove the container and check in the volumes mount point if the data is still available

# Inside cont1

cd /jars

wget https://war-jar-files.s3-us-west-2.amazonaws.com/gameoflife.war

exit

# In Docker Host

docker container rm -f cont1

# Check for volumes

docker volume ls

docker volume inspect

# copy the mount point dir

ls <mountpointdir>

# two files should be available (spc.jar and gol.war)

* Now lets mount the myvol to two containers and run the application simultaneously

docker container run --name cont3 --mount "source=myvol,target=/app" -d -p 8080:8080 openjdk:8 java -jar /app/spring-petclinic.jar

docker container run --name cont4 --mount "source=myvol,target=/app" -d -p 8081:8080 openjdk:8 java -jar /app/spring-petclinic.jar

docker container ls

docker volume ls

# remove all the containers

docker container rm -f $(docker container ls -a -q)

* Data from Docker Volume will be removed only when the volume is removed

docker volume rm myvol

* For creating a readonly volume [refer here](https://docs.docker.com/storage/volumes/#use-a-read-only-volume) (<https://docs.docker.com/storage/volumes/#use-a-read-only-volume>)

### Bind mount volumes

* Now your docker host has a folder /app you want the same folder in the container

docker container run --name cont5 --mount "type=bind,source=/app,target=/jars" -d -p 8080:8080 openjdk:8 java -jar /jars/spring-petclinic.jar

* For tempfs [refer here](https://docs.docker.com/storage/tmpfs/) (<https://docs.docker.com/storage/tmpfs/>)

### Dockerfile

* Inside the Dockerfile we have a instruction called as VOLUME.
* When you use this for every container a new volume is created.
* Lets write the Dockerfile

FROM openjdk:8

RUN mkdir /app && cd /app && wget https://war-jar-files.s3-us-west-2.amazonaws.com/spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar

VOLUME /app

EXPOSE 8080

CMD ["java", "-jar", "/app/spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar"]

* Create two container from the image and verify the volumes

docker volume ls

# inspect the required volume and you should see spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar in the mountpoint dir

docker stats

**FEBRUARY 2, 2020**

## DevOps Classroom Series – 02/Feb/2020 - Evening

### Docker Essential Command line

* Executing the commands inside the running container

# Check the contents of root directory in tomcat container

docker container run --name tomcat1 -d tomcat:8

docker container exec tomcat1 ls /

# check the process list

docker container exec tomcat1 ps -eaf

# executing commands in the interactive mode

docker container exec -it tomcat1 /bin/bash

* Starting and stopping containers

docker container stop tomcat1

docker container ls

# tomcat1 shouldn't be in the list

docker container ls -a

# start the container

docker container start tomcat1

* Getting to know the instructions led to the creation of docker image

docker image history <imagename>:<tag>

### Dockerfile instructions

* **WORKDIR**: workdir changes the default directory of the container
  + Create a Dockerfile and build the image with tag demo:0.1

FROM openjdk:8

RUN mkdir /app && cd /app && wget https://war-jar-files.s3-us-west-2.amazonaws.com/spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar

WORKDIR /app

EXPOSE 8080

CMD ["java", "-jar", "spring-petclinic-2.2.0.BUILD-SNAPSHOT.jar"]

\* create a container in the interactive mode to check the work directory

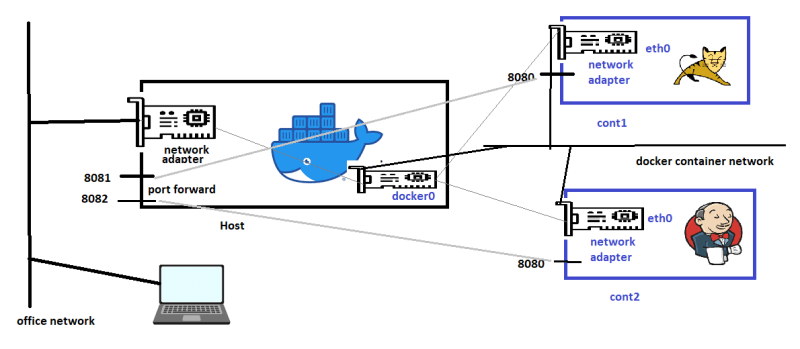
docker container run -it demo:0.1 /bin/bash

pwd

* **USER**: By default docker container runs as a root user, if you want to create a user and run as that user

USER qt

### Docker Single Host networking and Port forwarding



* To acces the application running inside the container we establish port forwarding between host port and container port
* Now lets examine one docker command

docker container run --name tomcatwn1 -d -p 8081:8080 tomcat:8

docker container run --name jenkinswn1 -d -p 8082:8080 jenkins

* Lets run this command

docker container run --name jenkinswn2 -d -p 8081:8080 jenkins

docker: Error response from daemon: driver failed programming external connectivity on endpoint jenkinswn2 (1b1e

dac2c349d064d7b5a50ff2e883ebb426d1da1b7afd61e6e2935bc5d9d919): Bind for 0.0.0.0:8081 failed: port is already allocated.

* To resolve the situation try use -P which will allocate any free port of the docker host to docker container

docker container run --name jenkinswn3 -d **-P** jenkins

* One scenario for container communications with in the container network
  + By default, docker containers (in linux) run on bridge network
  + Lets try to create two container c1 and c2.

docker container run --name c1 -d alpine sleep 1d

docker container run --name c2 -d alpine sleep 1d

# inspect the network

docker network inspect bridge

# make a note ip addresses of c1 and c2



\* Login into c1 container and ping c2 by name and ipaddress

docker container exec -it c1 /bin/sh

ping -c 4 172.17.0.3

# it works

# ping by name of the continer

ping -c 4 c2

# it fails

* Observeration: In the default bridge network, ip address is resolved but not the host name

**FEBRUARY 4, 2020**

## DevOps Classroom Series – 04/Feb/2020

### Docker Network Drivers

* Docker networking way back was part of docker daemon
* Docker networking is implemented as a different component **libnetwork**
* Networking in Docker is implemented as Network Drivers. Popular drivers are
  + Bridge
  + Host
  + MACVLAN
  + None
  + Overlay (multi host)
* [Refer](https://directdevops.blog/2019/10/05/docker-networking-series-i/) (<https://directdevops.blog/2019/10/05/docker-networking-series-i/>)  for Networking Series-1
* [Refer](https://directdevops.blog/2019/10/07/docker-networking-series-ii-overlay-networks/) (<https://directdevops.blog/2019/10/07/docker-networking-series-ii-overlay-networks/>) for Networking Series – 2 (Multihost)
* [Refer Here](https://docs.docker.com/network/) (<https://docs.docker.com/network/>) for official docker network docs

### Creating a new bridge network

* Create a new bridge network with range as 10.11.0.0/16 and name as mybridge

docker network create --driver bridge --subnet 10.11.0.0/16 mybridge

docker network ls

docker network inspect mybridge

* Create two container p1 and p2 on the mybridge

docker container run --name p1 --network mybridge -d alpine sleep 1d

docker container run --name p2 --network mybridge -d alpine sleep 1d

* Now test for the connectivity between p1 and p2 using ip and hostnames

docker network inspect mybridge

# make a note of ip addresses

docker exec p1 ping -c 3 10.11.0.3

docker exec p1 ping -c 3 p2

* In the above both containers can connect using names and ip addresss
* Try to add container c1 which is running in bridge network to mybridge network

docker network connect mybridge c1

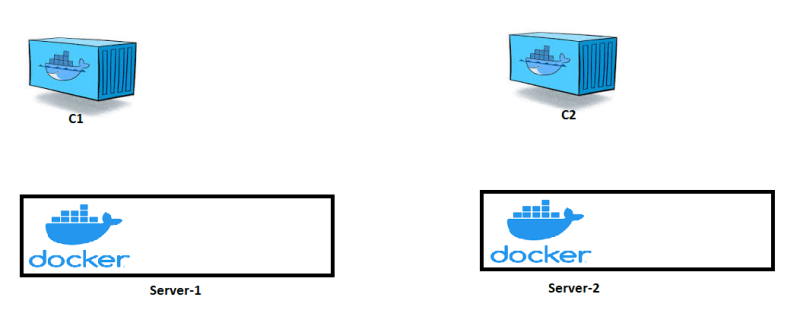
docker network inspect mybridge

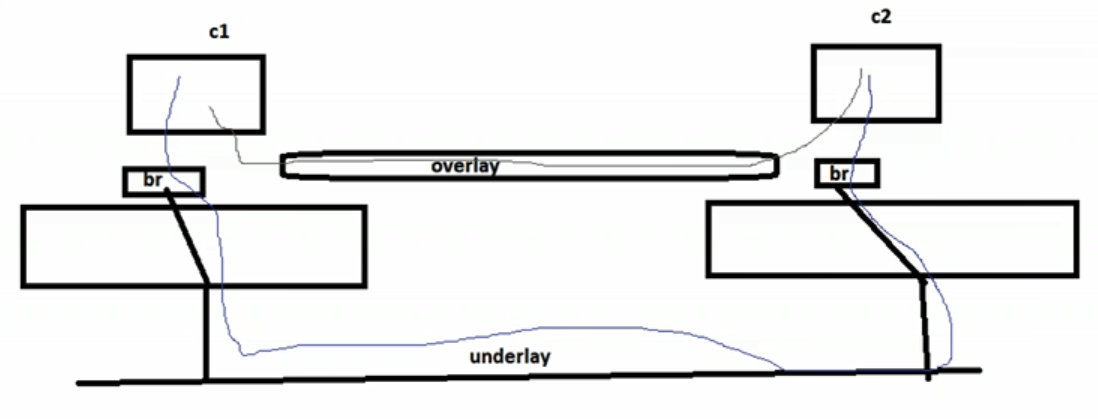
docker exec p1 ping -c 3 c1

**FEBRUARY 5, 2020**

## DevOps Classroom Series – 05/Feb/2020

### Multi-host Docker networking

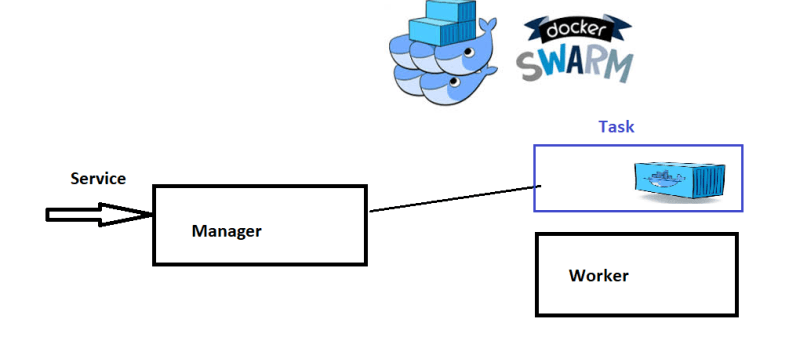
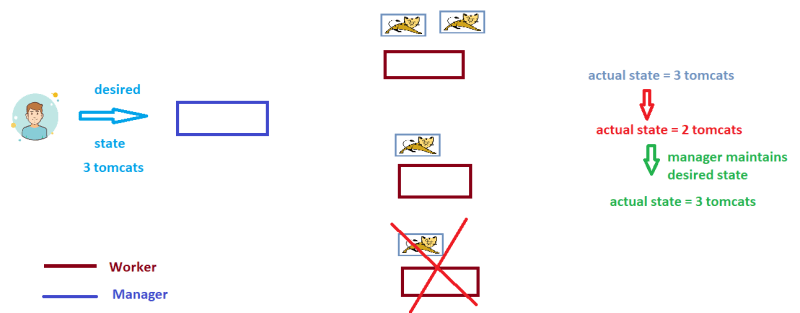
* Consider two docker hosts Server-1 and Server-2.
* On each host a docker container is launched (c1 and c2) 
* If c1 and c2 are in same host and same network we can use bridge network driver, which enables communication. Bridge can work with in one system , not across systems
* When we add multiple hosts, then we do it for
  + High availability
  + Fault Tolerance
* We need an orchestrator, to manage multi host docker containers, we have many options, but popular ones are
  + Docker Swarm
  + Kubernetes
* Coming back to networking, in docker we have a driver called as overlay, which can enable multi host networking (Swarm)
* [Refer Here](https://directdevops.blog/2019/10/07/docker-networking-series-ii-overlay-networks/) (<https://directdevops.blog/2019/10/07/docker-networking-series-ii-overlay-networks/>) for Overlay Networking
* [Refer Here](https://directdevops.blog/2019/10/07/docker-swarm-mode/) (<https://directdevops.blog/2019/10/07/docker-swarm-mode/>) for Docker Swarm



**FEBRUARY 6, 2020**

## DevOps Classroom Series – 06/Feb/2020

### Docker Swarm

* Nodes:
  + Machines with docker installed and are reachable to each other
  + Node Types
    - Manager
    - Worker
* Services
  + Using Docker service the user describes desired state
* Tasks
  + Atomic unit of Scheduling in Docker Swarm mode
  + Your containers will be running (logically) inside Task 
* Manager always tries to maintain state (desired == actual) 
* For official Swarm documentation [refer here](https://docs.docker.com/engine/swarm/) (<https://docs.docker.com/engine/swarm/>)
* [Refer Here](https://directdevops.blog/2019/10/07/docker-swarm-mode/) (<https://directdevops.blog/2019/10/07/docker-swarm-mode/>) for Swarm Quick start

**FEBRUARY 9, 2020**

## DevOps Classroom Series – 09/Feb/2020

### Dockerfile instructions (others)

* **ARG**:
  + is a variable that can be set during image build.
  + Scope: This ARG value will be available till the image is built
  + [Refer Here](https://docs.docker.com/engine/reference/builder/#arg) (<https://docs.docker.com/engine/reference/builder/#arg>)
* **ENV**:
  + this will be environment variable available in RUN container
  + These values can be set or changed while the container is created.
  + [Refer Here](https://docs.docker.com/engine/reference/builder/#env) (<https://docs.docker.com/engine/reference/builder/#env>)

Example:

FROM alpine:3.9.5

**ARG** CONT\_IMG\_VER

RUN echo $CONT\_IMG\_VER

**ENV** CONT\_IMG\_VER v1.0.0

RUN echo $CONT\_IMG\_VER

**CMD ["sh","-c","echo CONT\_IMG\_VER=${CONT\_IMG\_VER}"]**

FROM alpine:3.9.5

**ENV** DBIP 0.0.0.0

RUN echo $CONT\_IMG\_VER

CMD **echo $DBIP**

$docker container run –e **DBIP=1.0.0.0** test:0.5

1.0.0.0

CMD [“java” “–jar” “$LOCATION”] ---This will not work

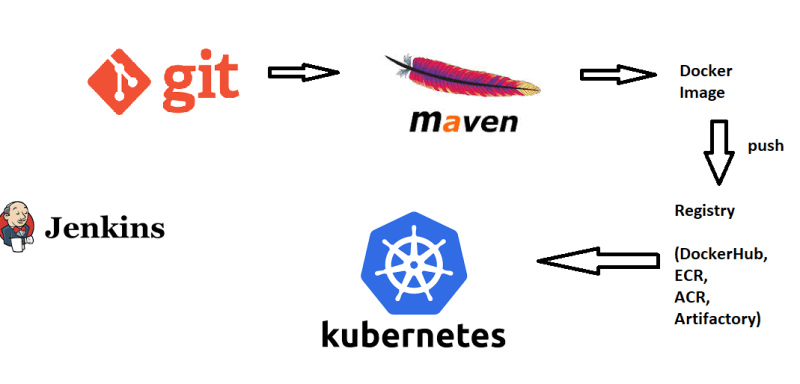
CMD java –jar $LOCATION ---This will work

ARG TEST=1.0

CMD java –jar $TEST ---This will not work as argument lifetime is till building the image.

* **ONBUILD**
* **STOPSIGNAL**
* **HEALTHCHECK**
* .**dockerignore**

### CI/CD Pipeline with Docker containers



* Create a jenkins job
* Get the latest code from git
* Build the package using build tools like maven
* Using the package build create the docker image
* tag the docker image and send it to the registry
* pull the docker image from registry and deploy using k8s/swarm etc

### Registry

* Collection of Docker Images
* DockerHub:
  + Before uploading images (push) to docker hub, login into docker hub from command line

### Sample Scenario: Push gameoflife to docker hub

* Clone the code

git clone https://github.com/wakaleo/game-of-life.git

* Build the code

mvn package

* Create a Docker file at the root folder of game-of-life

FROM tomcat:8

LABEL owner=none

EXPOSE 8080

ADD ./gameoflife-web/target/gameoflife.war /usr/local/tomcat/webapps/gameoflife.war

CMD ["catalina.sh", "run"]

* build the docker image

docker image build -t gol:1.0 .

* login to docker hub

docker login

username: shaikkhajaibrahim

password

* Now to push docker image to docker hub pattern is <username>/<imagename>:<tag>

docker image tag gol:1.0 shaikkhajaibrahim/gol:1.0

docker push shaikkhajaibrahim/gol:1.0

### multistage builds

* Multistage can be used to build applications and create docker images.
* [Refer Here](https://docs.docker.com/develop/develop-images/multistage-build/) (<https://docs.docker.com/develop/develop-images/multistage-build/>)
* Lets build the application game of life using maven and create a docker image from the single docker file

FROM maven:3-jdk-8 as builder

RUN git clone https://github.com/wakaleo/game-of-life.git

RUN cd /game-of-life

WORKDIR /game-of-life

RUN mvn package

FROM tomcat:8

LABEL owner=none

EXPOSE 8080

COPY --from=builder /game-of-life/gameoflife-web/target/gameoflife.war /usr/local/tomcat/webapps/gameoflife.war

CMD ["catalina.sh", "run"]

# KUBERNETES(K8S)

**FEBRUARY 7, 2020**

## DevOps Classroom Series – 07/Feb/2020

### Kubernetes

#### Where is it used?

* For any container deployments which require
  + automatic scaling
  + micro services
  + volumes mounted to EBS, Azure Disks or most of third party virtual disks
  + Zero down time deployments
  + Effective Load Balancer integrations

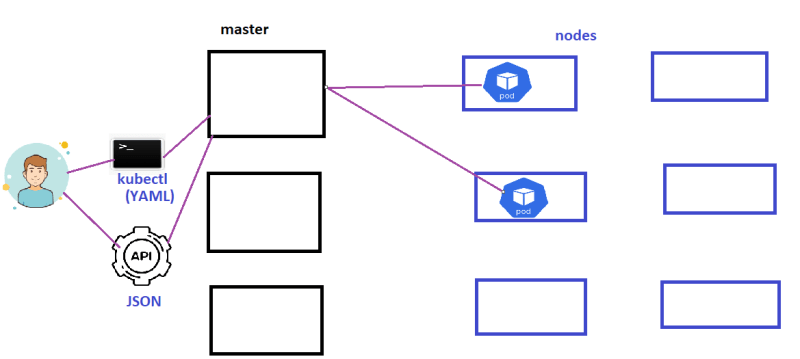
#### Why Kubernetes

* It was written by Google and then they made it OpenSource.
* Google has inhouse products
  + Borg
  + Omega
* Kubernetes is build based out of google’s experiences with Borg and Omega.
* Kubernetes runs anywhere (AWS, Azure, IOT , Linux, Windows, etc..)

#### Basic Working Style on Kubernetes

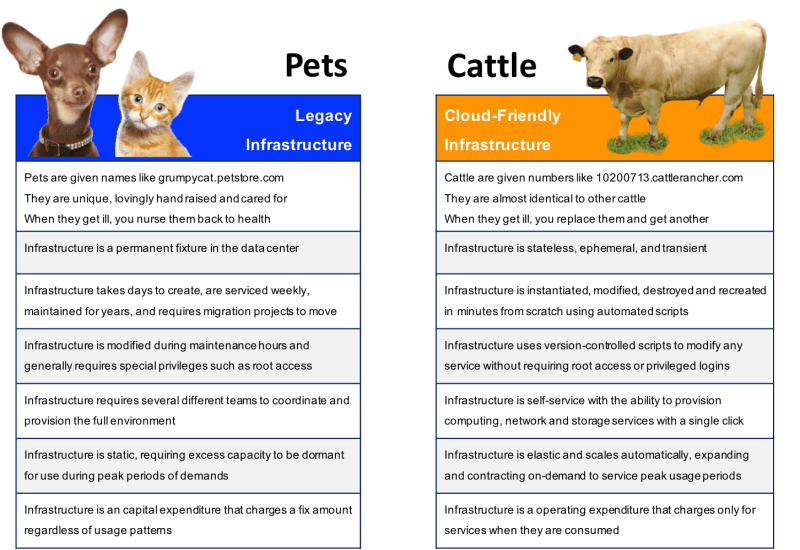
* Create manifests (Specifications) with minimal information which will be desired state.
* This manifest will be written in YAML.

#### Kubernetes Architecture (First View)



* User connects with Kubernetes Master and Master gets the work done (desired state == actual state) on nodes (minions).
* Kubernetes has two popular ways of communication to master from user
  + Command line:
    - kubectl is the command line tool
    - kubectl works in declarative and imperative way
      * for declarative way we use YAML
  + Rest API
    - JSON based Rest API is exposed

### Pet vs Cattle



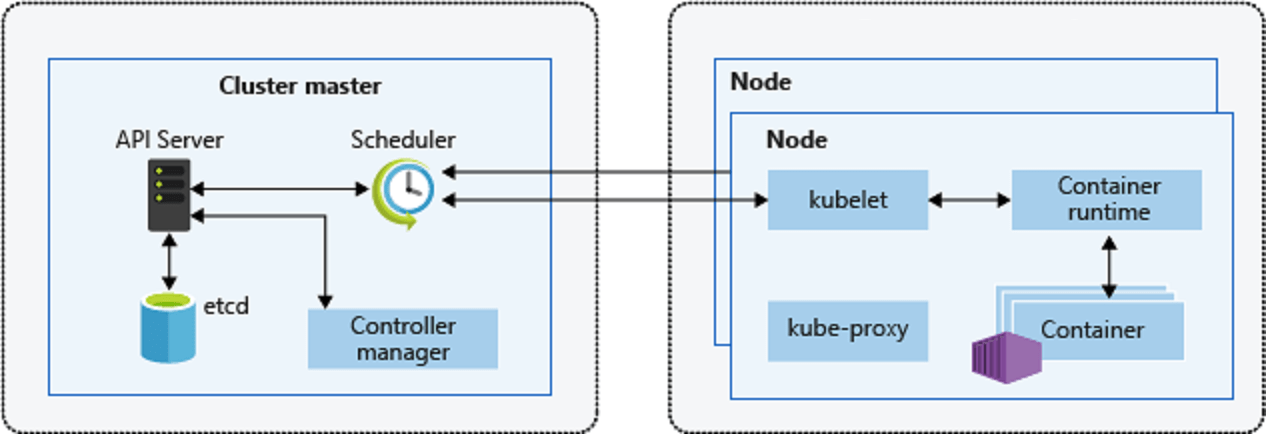
**FEBRUARY 8, 2020**

## DevOps Classroom Series – 08/Feb/2020

### Why Kubernetes

1. Self Healing Containers
2. Designed for Large Workloads
3. Specifications for the infrastructure in a simple YAML format
4. Kubernetes runs On-Premise, Cloud, Bare Metals
5. Rolling updates and rollbacks are supported.
6. [Refer Here](https://directdevops.blog/2019/10/09/kubernetes-introduction/) (<https://directdevops.blog/2019/10/09/kubernetes-introduction/>)  for introduction to Kubernetes

### Kubernetes Architecture



* **Cluster**: collection of nodes
* **Master**: Management of kubernetes cluster
* **Node**: Where applications run
* [Refer Here](https://directdevops.blog/2019/10/10/kubernetes-master-and-node-components/) (<https://directdevops.blog/2019/10/10/kubernetes-master-and-node-components/>)  for k8s components and architecture.

### Kubernetes Single Master Cluster

* Installation will be done using **kube-adm**
* For installation [Refer Here](https://directdevops.blog/2019/10/12/kubernetes-classroom-series-12-oct-2019/) (<https://directdevops.blog/2019/10/12/kubernetes-classroom-series-12-oct-2019/>)

**FEBRUARY 10, 2020**

## DevOps Classroom Series – 10/Feb/2020

### Kubernetes Objects

* Are persistent entities of K8s system
* Every object has
  + Object Spec:
  + Object Status:

### Basic Workflow

* Write a Specification in YAML format and save in a file
* To Create/modify

kubectl apply -f <pathofyaml>

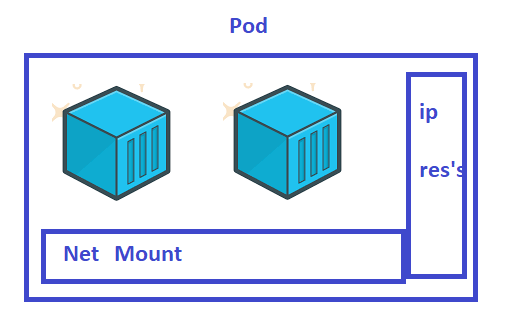
* To delete

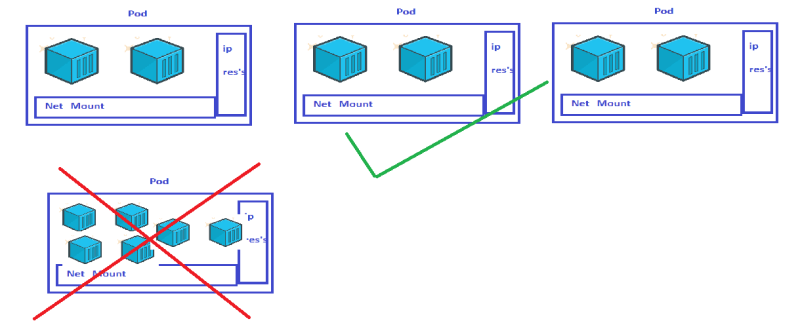
kubectl delete -f <pathofyml>

* To get status

kubectl get <object-kind>

### Pod



* Atomic unit of creation in k8s
* Pod contains container(s)
* Each pod gets an ip address
* When we have more than one container in the pod all the containers inside pod can be accessed over pod ip. For inter container communication with in pod the containers have to use localhost
* In k8s scaling is number of pods will be increased not containers 
* When you multi container pod
  + one container would be main car container
  + others would be side car containers
* Lets try to create Pod Specification. Refer to pod specification [from here](https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#pod-v1-core) (<https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/#pod-v1-core>)
* Refer to
  + apiVersion
  + kind
  + metadata
  + spec
* From the reference try to fill the values in a yaml file (learning)

---

apiVersion: v1

kind: Pod

metadata:

name: learning

spec:

containers:

- image: jenkins:1.609.2

name: jenkins

* Login into kubeadm master and execute kubectl apply -f learning.yml
* watch for status with changes

kubectl get pods -w

* Get full info of pods

kubectl get pods -o wide

* Delete the pods

kubectl delete -f learning.yml

**FEBRUARY 11, 2020**

## DevOps Classroom Series – 11/Feb/2020

### Important Concepts of K8s

* Namespace: In k8s we can create virtual clusters using namespace.

kubectl get namespace

* api-resources: All the different resources available in K8s

kubectl api-resources

### Workloads

### Pod

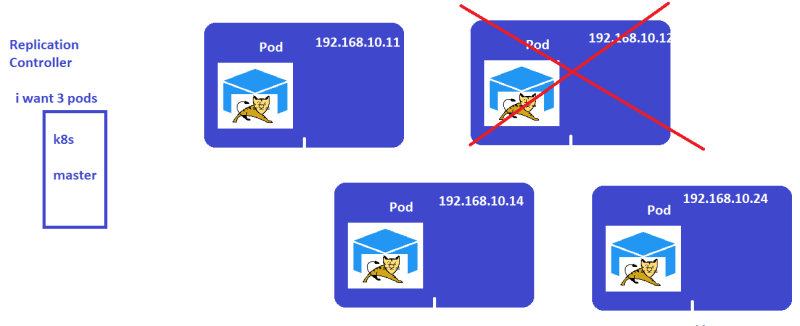
* [Refer Here](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/) (<https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/>)  for Pod Lifecycle
* init-containers: These are containers which will execute before the main containers. Init containers are supposed to executing only for a certain time.
  + Lets assume we have a Podspec in which
    - init-contiainer-1:
      * Here you have written ping -c 4 google.com
    - init-container-2
      * Here you have written ping -c 4 qt.artifactory.com
    - jenkins (main)

### Controllers

* In controllers, We control Workload objects

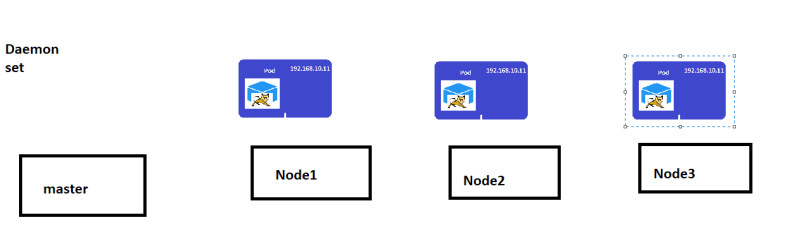
### Replication Controller

* In Replication Controller we can set number of pods as the desired state.



### Daemon Set

* Creates one pod on every node in K8s cluster



**FEBRUARY 12, 2020**

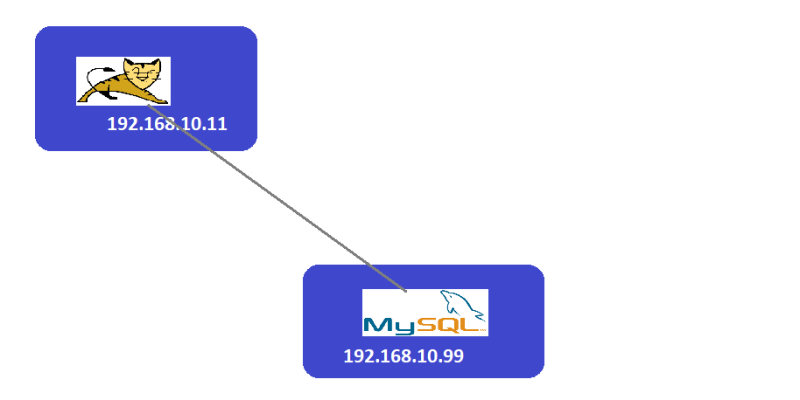
## DevOps Classroom Series – 12/Feb/2020

### Controllers

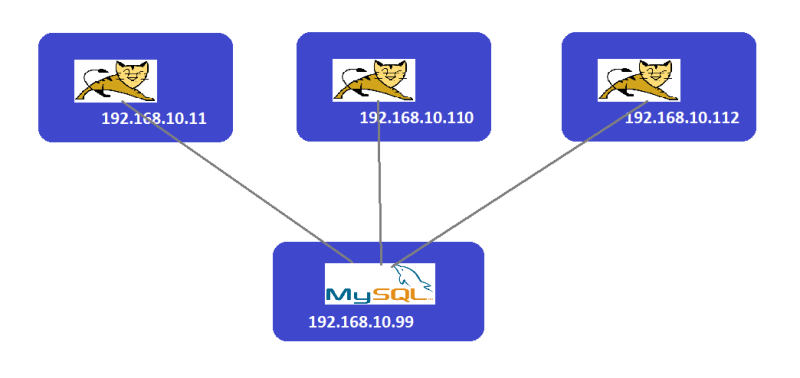
* Replica Set:
  + Much like Replication Controller
  + Replica Set can be preserved and revisited
* Stateful Sets: For easier understanding,
  + Stateful Sets are much like Replication Controllers with persistent storage
  + note: This definition is not 100% right, but its ok for now
* Jobs & Cron Jobs:
  + Job/Cron Job speaks about specification which will run some application/script/code which will execute and finish
* Note:
  + Like any other k8s object, Controllers also have
    - Specification
    - Status
  + Here the Specification is more about controlling the objects/workloads.
    - Can define workload
    - how to manage the workload

### Service

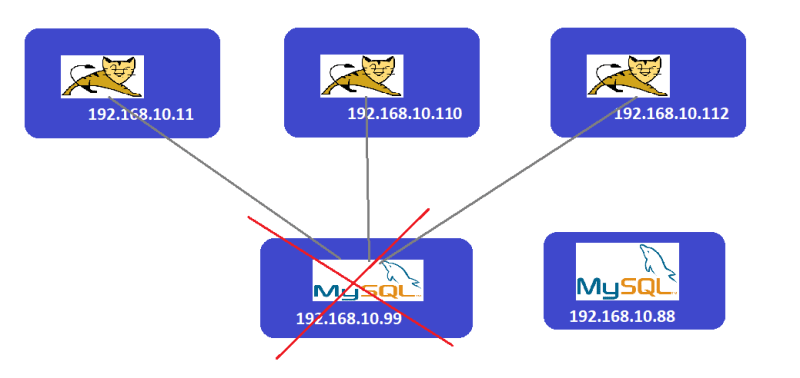
* Consider this Scenario, where we have app server and db server



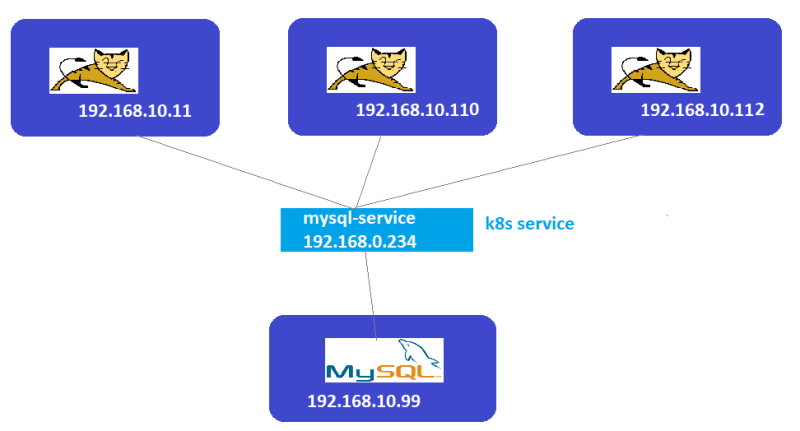
* In this app server will communicate with db server to store/retrieve the data
* Now due to application load you have scaled pods to 3



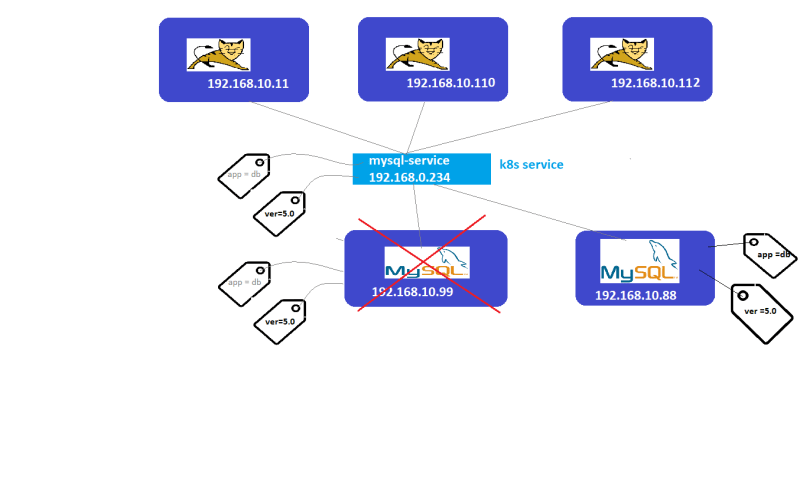
* Now lets assume the node on which mysql pod is running has failed, In that case we get a new sql pod which has new ip address



* In this case application stops working as db is not connected.
* To Resolve this , K8s uses Service which is used for communicating to the database pod.



* Now lets assume pod has failed and new mysql pod is created
* When a service is created, in the specification label information has to be provided.
* K8s service will forward traffic to the pods matching the labels given in the service spec



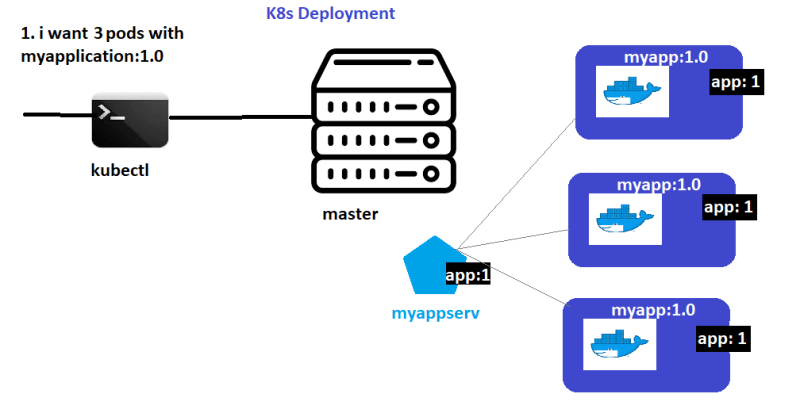
* Service is a Layer-4 load balancer
* In K8s layer-7 load balancing is supported by ingress.

**FEBRUARY 14, 2020**

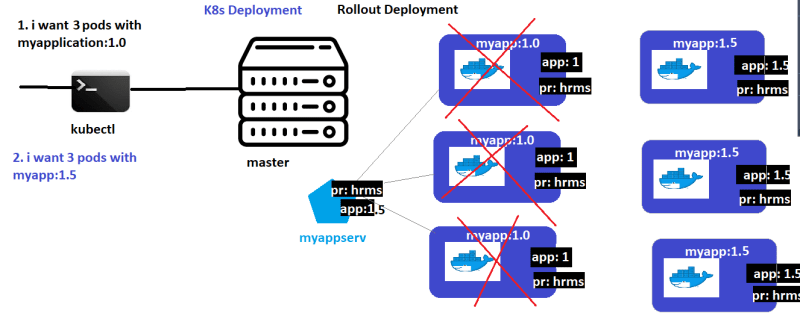
## DevOps Classroom Series – 14/Feb/2020

### Kubernetes Deployments

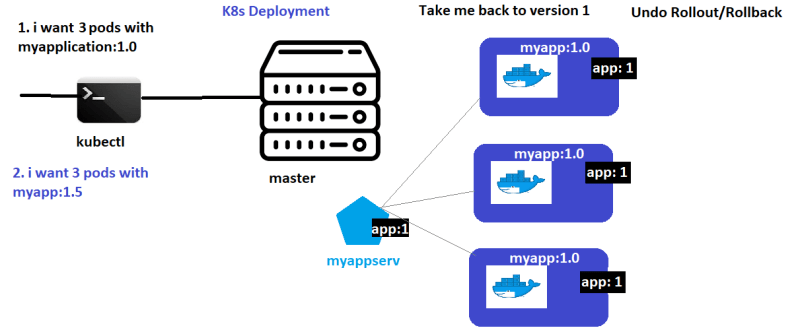
* In K8s Deployments specification we need to describe
  + How many Replications of Pod
  + Pod Spec
  + Strategy
* Deployments internally use Replica Sets.
* In this scenario we are deploying 3 pods with your application Docker image and Service



* Now new version of your application with tag 1.5 is available
* Then we make changes to spec and rollout deployment



* Now, for some reason your new version rolled out is defective, but k8s deployments support undo rollout or going back to previous deployment version
* If you rollback to version the scenario looks like below



**FEBRUARY 15, 2020**

## DevOps Classroom Series – 15/02/2020

### Storage in K8s

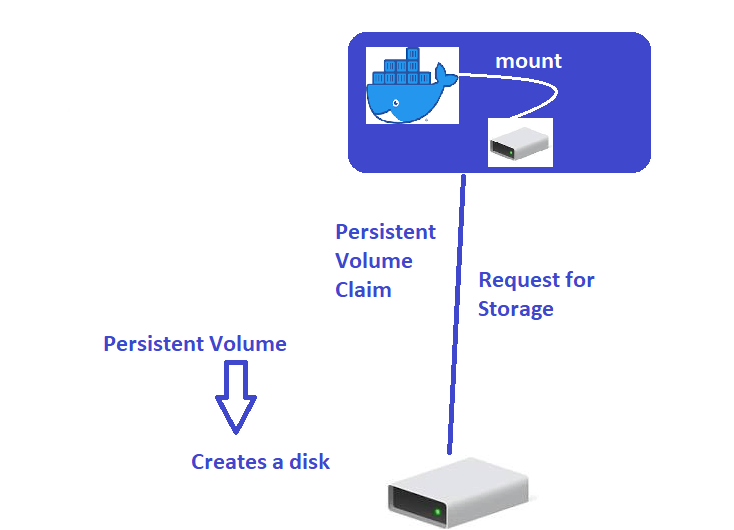
* Topics to understand in K8s Storage
  + Volume
  + Persistent Volumes
  + Storage Class
  + Persistent Volume Claims
* Areas of Concern
  + Docker Containers are ephemeral, to overcome we have Docker Volumes
  + Pods are ephemeral, As long as Pod is alive we have data, but once pod is deleted, data is also lost
  + K8s Works almost in many different platforms

### Kubernetes Storage

* Volumes lifetime is equal to lifetime of Pod
* Persistent Volumes lifetime is equal to lifetime of cluster

### Persistent Volumes Workflow

1. Create a Persistent Volume (PV) with requestable attributes
2. While Creating a Pod which needs PV, a claim/request called as PersitentVolumeClaim(PVC) is sent out to k8s
3. K8s searches for all the available PVs with the requested attributes in PVC.
4. If it finds the PV, then pod is created with attached PV which has to be mounted to docker container



### Persistent Volumes & Storage Classes

* PVs Can be created statically and also dynamically
* To get this dynamic provisioning storage classes help out.

### Networking in K8s

* Container Networking
* Pod to Pod Networking
* Pod to Service Networking
* Ingress
* Service Discovery
* [Refer Here](https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/) (<https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/>)  for deeper understanding of k8s networking
* [Refer Here](https://directdevops.blog/2019/10/17/kubernetes-classroom-series-17-oct-2019/) (<https://directdevops.blog/2019/10/17/kubernetes-classroom-series-17-oct-2019/>)

**FEBRUARY 16, 2020**

## DevOps Classroom 16/Feb/2020- K8s Start to End

### Journey of Openmrs-core from code to K8s

* Environments:
  + Azure Kubernetes Services (AKS)
  + Elastic Kubernetes Service (EKS)
* Steps:
  + Understanding how openmrs works
  + Creating Dockerfile(s) for Openmrs and its components
  + Creating K8s clusters
  + Creating K8s Specs for Deployment

### OpenMRS-Core Architecture



#### Dockerfile Creation

* Openmrs Core:

FROM maven:3-jdk-8 as mvn

LABEL author='Khaja'

RUN git clone https://github.com/openmrs/openmrs-core.git && cd openmrs-core && mvn clean package

FROM tomcat:8

LABEL author='Khaja'

COPY --from=mvn /openmrs-core/webapp/target/openmrs.war /usr/local/tomcat/webapps/openmrs.war

EXPOSE 8080

CMD ["catalina.sh", "run"]

* Now build the image and push the docker image to docker hub

docker image build -t shaikkhajaibrahim/openmrs-core:2.4.0 .

docker push shaikkhajaibrahim/openmrs-core:2.4.0

* How to run openmrscore on docker

docker container run --name openmrs -d -p 8080:8080 shaikkhajaibrahim/openmrs-core:2.4.0

# http:<ip>:8080/openmrs

* How to run mysql image on docker

docker container run --name mysql -e 'MYSQL\_ROOT\_PASSWORD=password' -e 'MYSQL\_DATABASE=openmrs' -e 'MYSQL\_USER=directdevops' -e 'MYSQL\_PASSWORD=directdevops' -d mysql:5.6

#### Setting up AKS k8s Cluster

* [Refer Here](https://docs.microsoft.com/bs-latn-ba/azure/aks/kubernetes-walkthrough) (<https://docs.microsoft.com/bs-latn-ba/azure/aks/kubernetes-walkthrough>)  for setting up AKS from azure cli

#### K8s Specifications

* Pod specification for Openmrscore

---

apiVersion: v1

kind: Pod

metadata:

name: openmrscore

labels:

app: openmrscore

ver: 2.4.0

spec:

containers:

- name: openmrscore

image: shaikkhajaibrahim/openmrs-core:2.4.0

ports:

- containerPort: 8080

protocol: TCP

* Pod specification for mysql

---

apiVersion: v1

kind: Pod

metadata:

name: mysql

labels:

app: db

ver: 5.6

spec:

containers:

- name: mysql

image: mysql:5.6

ports:

- containerPort: 3306

protocol: TCP

env:

- name: MYSQL\_ROOT\_PASSWORD

value: password

- name: MYSQL\_DATABASE

value: openmrs

- name: MYSQL\_USER

value: directdevops

- name: MYSQL\_PASSWORD

value: directdevops

* Replication Controller for openmrs will look like

---

apiVersion: v1

kind: ReplicationController

metadata:

name: openmrs-rc

labels:

app: openmrs

ver: 2.4.0

spec:

replicas: 3

selector:

app: openmrscore

ver: 2.4.0

template:

metadata:

labels:

app: openmrscore

ver: 2.4.0

spec:

containers:

- name: openmrscore

image: shaikkhajaibrahim/openmrs-core:2.4.0

ports:

- containerPort: 8080

protocol: TCP

* Deployment of the Openmrs will look like

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: openmrs-deploy

spec:

minReadySeconds: 10

replicas: 3

selector:

matchLabels:

app: openmrscore

ver: 2.4.0

strategy:

rollingUpdate:

maxSurge: 35%

maxUnavailable: 30%

type: RollingUpdate

template:

metadata:

labels:

app: openmrscore

ver: 2.4.0

spec:

containers:

- name: openmrscore

image: shaikkhajaibrahim/openmrs-core:2.4.0

ports:

- containerPort: 8080

protocol: TCP

* While Writing mysql, we need to persist data in the case of azure [refer here](https://docs.microsoft.com/en-Us/azure/aks/azure-disks-dynamic-pv) (<https://docs.microsoft.com/en-Us/azure/aks/azure-disks-dynamic-pv>)
* Deployment for mysql in azure

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: azure-managed-disk

spec:

accessModes:

- ReadWriteOnce

storageClassName: default

resources:

requests:

storage: 1Gi

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: mysql-deploy

spec:

minReadySeconds: 10

replicas: 1

selector:

matchLabels:

app: mysql

template:

metadata:

labels:

app: mysql

spec:

containers:

- name: mysql

image: mysql:5.6

ports:

- containerPort: 3306

protocol: TCP

volumeMounts:

- mountPath: /var/lib/mysql

name: mysqlvolume

env:

- name: MYSQL\_DATABASE

value: 'openmrs'

- name: MYSQL\_USER

value: 'directdevops'

- name: MYSQL\_PASSWORD

value: 'directdevops'

- name: MYSQL\_ROOT\_PASSWORD

value: 'password'

volumes:

- name: mysqlvolume

persistentVolumeClaim:

claimName: azure-managed-disk

### How to enable communications between pods

* [Refer](https://github.com/shaikkhajaibrahim/openmrs-core/tree/master/k8s)(<https://github.com/shaikkhajaibrahim/openmrs-core/tree/master/k8s>) for k8s spec yaml files.
* Create Service with loadbalancer for openmrs

---

apiVersion: v1

kind: Service

metadata:

name: openmrssvc

spec:

selector:

app: openmrscore

ver: 2.4.0

type: LoadBalancer

ports:

- port: 8080

* Create a Service with Cluster Ip for mysql

---

apiVersion: v1

kind: Service

metadata:

name: mysqlsvc

spec:

selector:

app: mysql

type: ClusterIP

ports:

- port: 3306

* Apply the services to the k8s cluster

kubectl apply -f openmrs-svc.yml

kubectl apply -f mysql-svc.yml

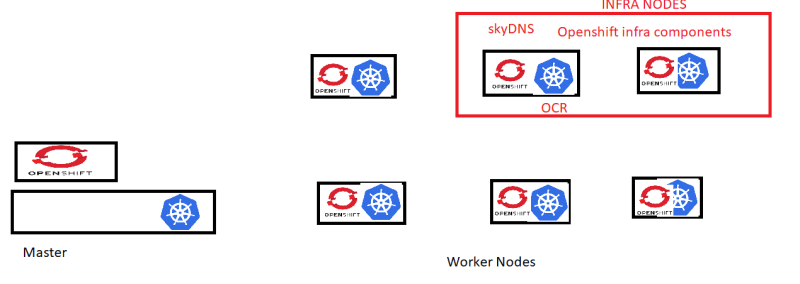
# OPENSHIFT

**FEBRUARY 23, 2020**

## DevOps Classroom Series – 22/Feb/2020

### Openshift

* Openshift helps in simplifying k8s deployments
* Openshift has a K8s cluster running.
* In K8s, we have two types of nodes
  + Master
  + Worker
* In Openshift
  + Master(k8s) => Master (openshift)
  + Worker (k8s)
    - Worker Node (openshift)
    - Infra Node
* Infra Nodes are required to maintain
  + DNS Components of open shift
  + OCR (a Private Docker Registry)
  + Routes of Openshift
  + Maintain
    - Build Specs
    - Image Streams
    - Catalogs



### Openshift Product Lines

* OKD (all the components on one server)
* Openshift Online
* Openshift Enterprise
* Openshift on Cloud Providers like Azure and AWS

### Openshift Exercise

* Create an Openshift online account [from here](https://www.openshift.com/) (<https://www.openshift.com/>)