DevOps Fundamentals

FORMAC INC |

Table of Contents

WHAT IS DEVOPS? 6
LINUX 6
IP TABLES 8
PING VS TELNET 9
PERFORMANCE TUNING OF UNIX 9
AUTOSSH 10
OPENING PORTS 10
PROVIDING ROOT PERMISSION TO USERS 10
USER CREATION AND DELETION IN AWS 11
SETTING TEMPORARY ROOT PASSWORD 13
GIT 13
ARCHITECTURE 13
GITHUB 13
GIT VS SVN: 13
GIT WORKFLOW 14
GIT COMMANDS: 14
MERGE CONFLICT 15
MERGE VS REBASE 15
BRANCHING STRATEGY: 15
SVN TO GIT MIGRATION 16
SVN TO GIT MIGRATION SCRIPT 17
MAVEN 18
LIFECYCLE 18
ANT VS MAVEN 19
PHASES VS GOALS 19
DEPENDENCY MANAGEMENT 19
MAVEN RELEASE PLUGIN 19
REPOSITORIES 20
SETTINGS.XML 20
POM.XML 20
INHERITANCE VS AGGREGATION 20
JENKINS 21
JENKINS_HOME 21

DASHBOARD 22
DIFFERENT TYPES OF JOB CREATED 22
JENKINS WEATHER 22
JENKINS BUILD STATUS 22
JENKINS PLUGINS 22
JENKINS PIPELINE METHODOLOGY 23
SAMPLE WINDOWS PIPELINE 23
JENKINS PIPELINE BENEFITS OVER JENKINS UI 24
WHAT IS CI? 25
WHY CI IS REQUIRED? 25
CI LIFECYCLE 25
JENKINS BEST PRACTICES 25
COMMON JENKINS FAILURES 26
CI/CD PROCESS 26
CHEF 27
BASIC DEFINITIONS 27
BOOTSTRAP PROCESS 29
CHEF-DK COMPONENTS 29
CHEF SERVER COMPONENTS 30
FOODCRITIC 30
BERKSHELF 30
CHEFSPEC 31
TESTKITCHEN 31
CHEF_CLIENT RUN 32
COOKBOOK FOLDER STRUCTURE 32
ATTRIBUTES PRECEDENCE 33
TYPES OF COOKBOOKS 34
PREDEFINED RESOURCES 34
Linux Academy – Chef Development: 39
AWS 43
PROVISIONING AN EC2 INSTANCE 43
BASIC AWS SERVICES 44
AWS ROUTE 53 45
AWS MACHINE TYPES 46
AWS S3 47

AWS IAM 48
AWS CLOUDFORMATION 49
CFT BEST PRACTICES 49
AWS HELPER SCRIPTS 50
AWS VPC 52
NACL VS SG 52
VPC PEERING 53
AWS ELB 53
APPLICATION LOAD BALANCER (ALB) VS CLASSIC LOAD BALANCER (CLB) 53
AWS AUTOSCALING 54
AWS RDS 54
AWS RESOURCE LIMITS 55
CLOUDFRONT 56
AWS DYNAMODB 56
MIGRATION TO CLOUD 56
BLUE-GREEN DEPLOYMENT 58
DOCKER 59
VM VS DOCKER CONTAINERS 59
DOCKERFILE 60
DOCKER REGISTRY & DOCKER HUB 63
DOCKER ADD VS COPY 63
DOCKER STOP VS KILL 63
DOCKER DEBUGGING 63
DOCKERFILE BEST PRACTICES 63
KUBERNETES 64
CHALLENGES ADDRESSED BY KUBERNETES 64
MASTER OR CONTROL PLANE COMPONENTS 64
NODE SERVER COMPONENTS 65
KUBERNETES IN A NUTSHELL 65
CONFIGURING MASTER AND MINIONS: 66
RUNDECK 68
TERRAFORM 69
Database: Oracle 11g (MYSQL) 70
SHELL SCRIPTING 71
ANSIBLE 74

PLAYBOOKS 75
REST VS SOAP 78
HTTP METHODS 78
WHEN TO USE SOAP 79
WHEN TO USE REST 79
GENERAL TOPICS 79
WHY MULTIPLE JVMS 79
STATEFUL VS STATELESS SERVICES 79
APPLICATION VERSIONS 79
APACHE WEB SERVER 80
TOMCAT 80
Running multiple web applications in one Tomcat Server: 80
Benefits of Tomcat over other servers: 80
Application Server Tuning 81
Tomcat JDBC resource configuration: 81
Tomcat clustering and loadbalancing with HTTP: 82
SERVICENOW 84
SONARQUBE QUALITY GATES 84
JAR, WAR & EAR: 84
FORWARD VS REVERSE PROXY 85
Setting up the reverse-proxy: 85
ROUTING AND SWITCHING 86
RAID 86
WHY BINARY REPOSITORY MANAGER OR ARTIFACTORY? 87
SSL CREATION 87
JIRA INTEGRATION WITH GITHUB 88
JBOSS 88
HTTP RESPONSE CODES 90
REFERENCES 91

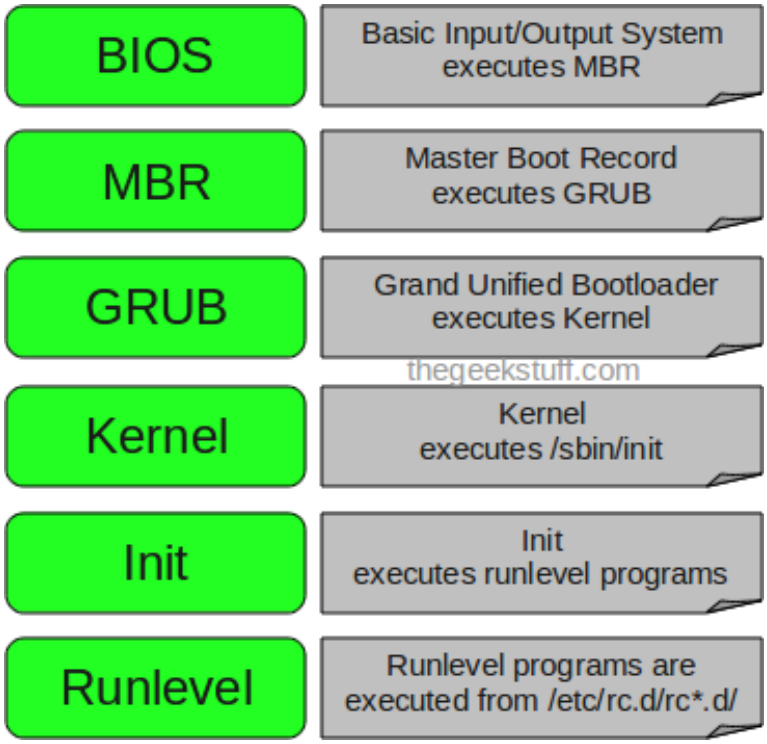
WHAT IS DEVOPS?

DevOps is a term used to refer to a set of practices which emphasize the communication and collaboration of the software developers and the rest IT professionals while automating the software integration and delivery process. While Agile has been in the industry for a while, DevOps is mostly a new concept which had turned into a movement in the technical arena. In simple, DevOps refers to Development+Operations where operations are a blanket term for all those apart from developers like system engineers, system administrators etc. It aims at enhancing the collaboration between these two fields and produce a high performing and reliable end-product. In order to be successful in DevOps arena, one should be thorough with various IT concepts:

- Operating System (**Linux** – one of the widely adopted open source OS)
- Version Control System (**GIT** – leading market share holder)
- Build tool – **Maven**
- CI & CD tool – **Jenkins**
- Configuration Management tool – **Chef**
- Cloud Services – **AWS**
- Containerization services – **Docker**

LINUX

The booting process of linux is represented in the below diagram:



The basic concept to be understood in linux is the **runtime levels** of a Linux OS. There are total 7 run levels where:

- 0 – Halt
- 1 – Single User
- 2 – Multiple user without NFS
- 3 – Full multiuser mode
- 4 – unused
- 5 – Full multiuser with networking on windows – basically gui
- 6 – reboot

Inode, also called as index number, identifies the file and its attributes – use **-I flag**

Few other important commands in Linux include:

To search for files and directories – **find . -name "string" -type f/d**

To search within a specified period - **-mtime -/+n**

To search within the contents of a file – **grep 'string' file_path**

To list all processes on the OS – **ps -ef**

To find and replace a string – **sed 's/find_term/replace_term/g' file_path**

To redirect the output into next command or file - >

To pass two or more commands at a time - **| (pipe)**

To display various network related information – **netstat**

- -a – all ports
- -t – tcp
- -u – udp
- -l – listening
- -s – statistics
- -p – PID and program name
- -r – kernel routing

To communicate with another host using Telnet protocol – **tel**

net [host or IP [port]] (if the telnet server is running, it will be listening on tcp 23 by default)

To kill a process – kill -3 PID

To login to a remote machine and execute command – **ssh remote_Host**

To securely copy files between remote hosts – **Scp source_file_path user@dest_host:dest_path**

To check the file systems disk space usage DISK FREE– **df -h (-T type, -m MB, -k KB)**

To check the DISK USAGE of the directories and sub directories and files – **du -h** (-s summarize)

To find top 10 files using most disk space – **du -a | sort -nr | head -n 10**

To monitor processes based on cpu/memory usage – **top** (M -memory, P – CPU usage, N – by PID, T – running time, i – idle processes).

To automate the routine tasks – **crontab -e, -r (delete), -l (list)**

```
* * * * * command to execute

 T T T T T
 | | | | |
 | | | | |
 | | | | └──── day of week (0 - 7) (0 to 6 are Sunday to Saturday, or use names; 7 is Sunday,
the same as 0)
 | | | └────────── month (1 - 12)
 | | └──────────────── day of month (1 - 31)
 | └────────────────────── hour (0 - 23)
 └──────────────────────────── min (0 - 59)
```

To preserve time stamp while copying – **cp -p**

To display firewall status – **iptables -L -n, -F(to clear),**

To get the CPU info **– cat /proc/cpuinfo**

To run a program in background: **nohup node server.js > /dev/null 2>&1 &**

nohup means: Do not terminate this process even when the stty is cut off.

> /dev/null means: stdout goes to /dev/null (which is a dummy device that does not record any output).

2>&1 means: stderr also goes to the stdout (which is already redirected to /dev/null). You may replace &1 with a file path to keep a log of errors, e.g.: 2>/tmp/myLog

& at the end means: run this command as a background task.

To check top 10 process is using most memory **- ps aux --sort=-%mem | awk 'NR<=10{print $0}'**

Disk partition commands: **fdisk /dev/sda (m-help, n-new, d-delete, w-to apply changes, p -print, s-size, x followed by f – to fix order of partition table)**

To transfer files remotely and locally – **rsync options source destination (v-verbose, r-recursive, a-archive (r+ symbolic links, file permissions, user & group ownerships and timestamps), z-compress, h-human, --exclude, --include, e-to specify protocol (ssh), --progress, --delete, --max-size, --remove-source-files, --dry-run)**

To analyze packets or linux packet sniffer tools – **tcpdump -i eth0 (c-to specify number of packets, D-display available interfaces, w-capture&save, r-read captured, n-capture ip address packets,**

Linux Proc contents: **APM, CPUINFO, DEVICES, DMA, FILESYSTEMS, IOMEM, LOADAVG, LOCKS, MEMINFO, MOUNTS, PARTITIONS, SWAPS, UPTIME.**

IP TABLES

iptables is a command-line firewall utility that uses policy chains to allow or block traffic. iptables uses three different chains: input – to control incoming connections, forward – to control incoming connections in which

the data is forwarded (ex router) and is used only when there is some sort of forwarding (checked using **iptables -L -V**), and output – to control outgoing connections.

iptables	**--policy**	**INPUT**	**ACCEPT/DROP**
iptables	**--policy**	**OUTPUT**	**ACCEPT/DROP**

iptables --policy FORWARD ACCEPT/DROP

iptables -A INPUT -s 10.10.10.10 -j DROP – Block all connections from 10.10.10.10.

iptables -A INPUT -s 10.10.10.0/24 -j DROP – Block all connections from a range of ip addresses.

iptables -A INPUT -p tcp --dport ssh -s 10.10.10.10 -j DROP – block ssh from

iptables -A INPUT -p tcp --dport ssh -j DROP – to block all ssh connections

/etc/init.d/iptables save – to apply the changes permanently

PING VS TELNET

Both ping and telnet could be used for diagnosing network problems. Ping simply gets a response and it is what it can do. While it is possible to monitor the output from a given port using telnet.

Telnet sends usernames and passwords over network in clear text format so it is open to eavesdropping and considered very unsecure. Main advantages of SSH over Telnet is all the communication is encrypted and sensitive user data travels encrypted over internet so it is not possible to extract these credentials easily.

PERFORMANCE TUNING OF UNIX

It is done by basically monitoring the following sub-systems: CPU, memory, I/O and Network.

Four critical performance metrics for CPU are: context switch – during multitasking, CPU stores the current state of the process before switching to the other and higher level of this causes performance issues. Run queue – refers to the number of active processes in queue for CPU and higher the number lower the performance, CPU utilization – through TOP command and higher % of utilization causes performance issues and finally Load Average – which refers to the average load on CPU which is displayed for the last 1, 5 and 15 minutes.

Network – things like number of packets received, sent and dropped etc needs to be monitored for network interfaces.

I/O – higher I/O wait time indicates disk subsystem problem. Reads per sec and writes per sec needs to be monitored in blocks referred to as bibo. TPS – RPS+WPS

Memory - Virtual memory = Swap space available on the disk + Physical memory.

Remember the 80/20 rule — 80% of the performance improvement comes from tuning the application, and the rest 20% comes from tuning the infrastructure components.

Performance tuning: The parameters that would be altered to perform a performance tuning of a linux machine would be the contents of /proc/sys folder which are: **shared memory** segment (an inter-process communication mechanism that enables the visibility of a memory segment to be visible to all processes in a single namespace),

Swappiness (how aggressively memory pages are swapped to disk), disabling the kernel's ability to respond to ICMP broadcast ping requests,

AUTOSSH

Server A to Server B

1. Generate the ssh keys on A:

ssh-keygen -t rsa

2. change the password authentication to yes in A:

sudo vi /etc/ssh/sshd_config

3. From .ssh folder:

ssh-copy-id user@host

4. check folder permissions in case of any issue. 700 for user home directory and .ssh folder and 600 for files in it.

5. Once done change the **passwod authentication** back to **no** in **sshd_config file**.

OPENING PORTS

To open a port: **sudo afw allow portno/port type** - example **sudo afw allow 22/tcp**

The service specific syntax is as follows to open http and https service ports:

sudo ufw allow http

sudo ufw allow https

OR

sudo ufw allow 80/tcp

sudo ufw allow 443/tcp

Advanced examples

To allow IP address 192.168.1.10 access to port 22 for all protocols

sudo ufw allow from 192.168.1.10 to any port 22

Open port 74.86.26.69:443 (SSL 443 nginx/apache/lighttpd server) for all, enter:

sudo ufw allow from any to 74.86.26.69 port 443 proto tcp

To allows subnet 192.168.1.0/24 to Sabma services, enter:

ufw allow from 192.168.1.0/24 to any app Samba

You can find service info as follows:

sudo ufw app list

PROVIDING ROOT PERMISSION TO USERS

By default, the sshd daemon is configured to refuse direct connections by the root user, so you won't be able to log in over SSH as a root user with this password. For security reasons, avoid enabling direct SSH access for the

root user. Instead, connect by using the user ID associated with your operating system (for example, "ec2-user" for many Linux distributions) and a key pair.

If you need to add a root user password temporarily:

1. Connect to your EC2 instance running Linux by using SSH.
2. Assume root user permissions by running the following command:

sudo su

passwd root - to create temporary password

passwd -l root - to delete the password

USER CREATION AND DELETION IN AWS

Sudo adduser user_name

Sudo su – user_name

Mkdir .ssh

Chmod 700 .ssh

Touch .ssh/authorized_keys

Chmod 600 .ssh/authorized_keys

Exit

Ssh-keygen -t rsa

Copy id_rsa.pub

Sudo su – user_name

Paste in .ssh/authorized_keys

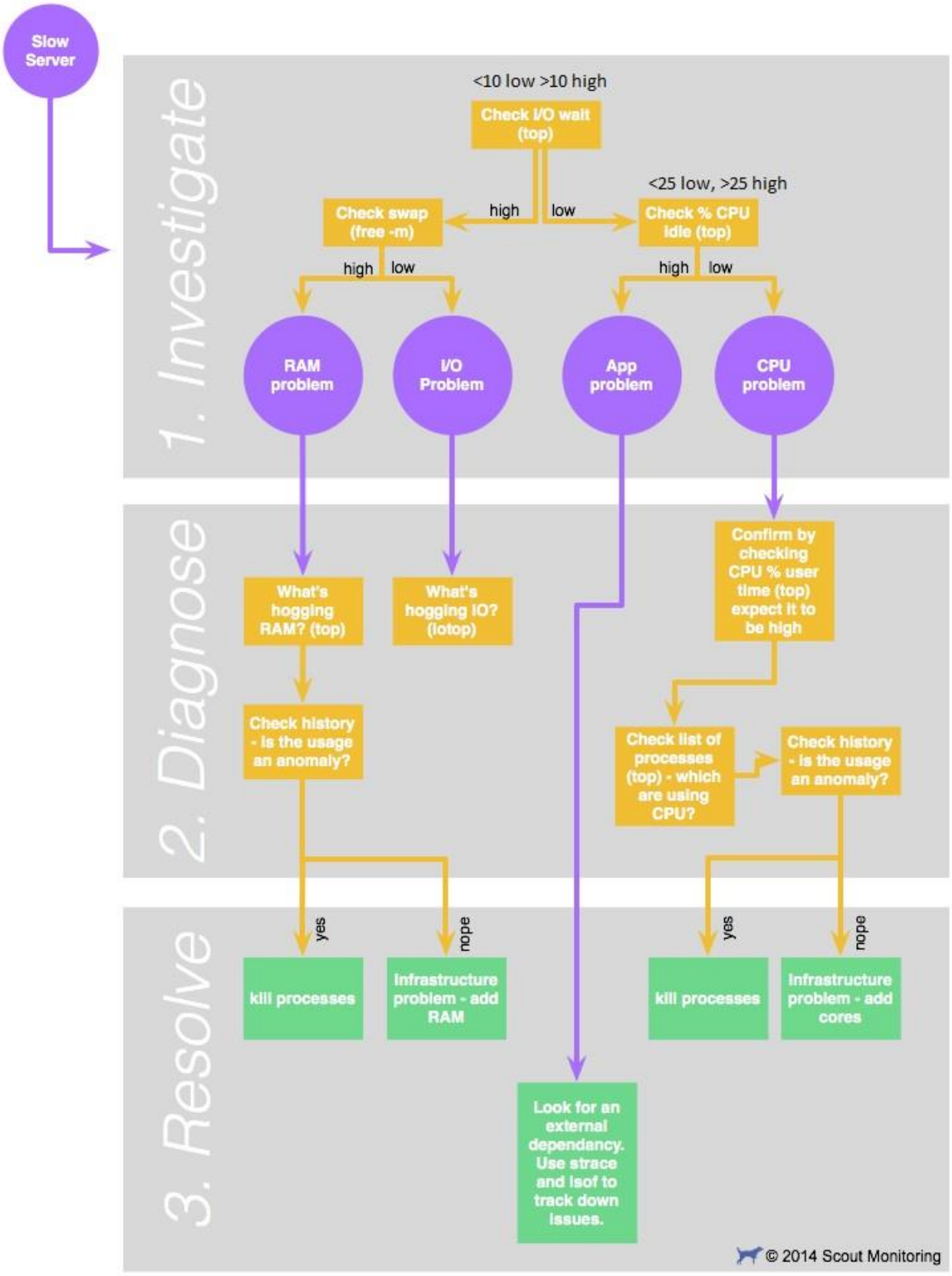
Sudo userdel -r user_name

SERVER MAINTENANCE BEST PRACTICES

- Verify whether your backups are working or not
- Check disk usage
- Monitor RAID alarms
- Keep on top of your OS updates
- Check your application updates
- Check for hardware errors (through logfiles)
- Check server utilization (systat)
- Review user accounts
- Change passwords
- Periodical audit of server security

To display all users: **getent passwd**

SLOW SERVER DEBUGGING:



SETTING TEMPORARY ROOT PASSWORD

sudo su

passwd root

enter password:

re-enteer password:

passwd -l root

GIT

GIT is the most widely used distributed version control system which is used for storing source code. Unlike other centralized VCS, in GIT every developer's working copy of the code is also a repository that can contain the full history of all changes.

ARCHITECTURE

A typical VCS have two trees viz., working copy and Repository. While working copy stores the copy of the code on which you are currently working, a repository stores the versions of the code. Checking-out is the process of getting files from repository to working copy and committing is the process of placing the work from working copy to repository. Git has a third tree referred to as staging which comes between the repository and working copy which is the place where all the changes that are to be committed will be prepared. Thus, you will have a chance to review your changes once again before committing them.

Server-client: Although GIT is a DVCS, it uses server-client concept in order to avoid the ambiguity among developers and to ensure the availability of the code even when the developer's system is offline. The source code will be stored centrally in a server from which each developer check-outs and commits.

GITHUB

GitHub, Bitbucket and GitLab are the repository hosting platforms which enables the management of the repositories. Out of these three, only GitLab is the open-source version and GitHub is the largest repository host with more than 38 million projects.

GIT VS SVN:

- GIT is a distributed VCS while SVN is a centralized VCS. (in SVN version history is on server side copy).
- In the case of large binary files, GIT is problematic in storing them while with SVN, the checkout times are faster with just the latest changes being checked out in SVN.
- Branching is considered to be lighter in GIT than SVN as in GIT a commit to a local repository is referred to as branch and it is easy to create and publish them at discretion.
- While GIT assumes all of the contributors have the same permission, SVN allows to specify read and write access controls.
- While SVN calls for network connection for every commit, GIT doesn't.

GIT WORKFLOW

The basic **Git workflow** goes something like this:

- You clone the files from your remote repository to your working tree
- You modify files in your working tree.
- You stage the files, adding snapshots of them to your staging area.
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

GIT COMMANDS:

To initialize a git repository – **git init .**

To get a local working copy of repository (for the first time) – **git clone path**

To create a branch – **git checkout -b**

To add a file to repository – **git add file_name**

To commit the changed files – **git commit -m "message"**

To get the changes on to the local repository (.git) – **git fetch**

To update the working copy with new commits on repository – **git pull = fetch + merge**

To merge a branch to master (on master) – **git merge <link_name> master**

To check the link between the local and remote repository – **git remote -v**

To add a link – **git remote add <link_name> path_to_repository**

To remove and don't track anymore – **git rm --cached**

To go back to a commit – **git revert <commit_hash>**

To go back to a commit and erase the history too – **git reset commit_hash** (soft-head, mixed head and staging, hard-head, staging and working directory)

To delete untracked files – **git clean -f**

To update commit message – **git commit --amend -m "message" commit_hash**

To choose a commit from a branch and apply it – **git cherry-pick commit_hash**

To record the current state of working directory and index and keep the working directory clean – **git stash save "message"**

To get the stashed changes back – **git stash apply commit_hash**

To see commits – **git log**

To compare modified files – **git diff** (--color-words to highlight changes)

To denote specific release versions of the code – **git tag -a name -m "message"**

To squash last N commits into a single commit – **git rebase -I HEAD={n}**

To list files changed in a particular commit – **git difff-tree -r <commit_id>**

MERGE CONFLICT

A merge conflict happens when two branches both modify the same region of a file and are subsequently merged. Git can't know which of the changes to keep, and thus needs human intervention to resolve the conflict.

Merge conflicts can be resolved either by using **git mergetool** which walks us through each conflict in gui. Or, it can be done using the **diff3 merge conflict style** by setting git config merge.conflictstyle diff3. It separates the file into three parts with 1st part being the destination of the merge, ,2nd part is common and the 3rd one is the source part.

MERGE VS REBASE

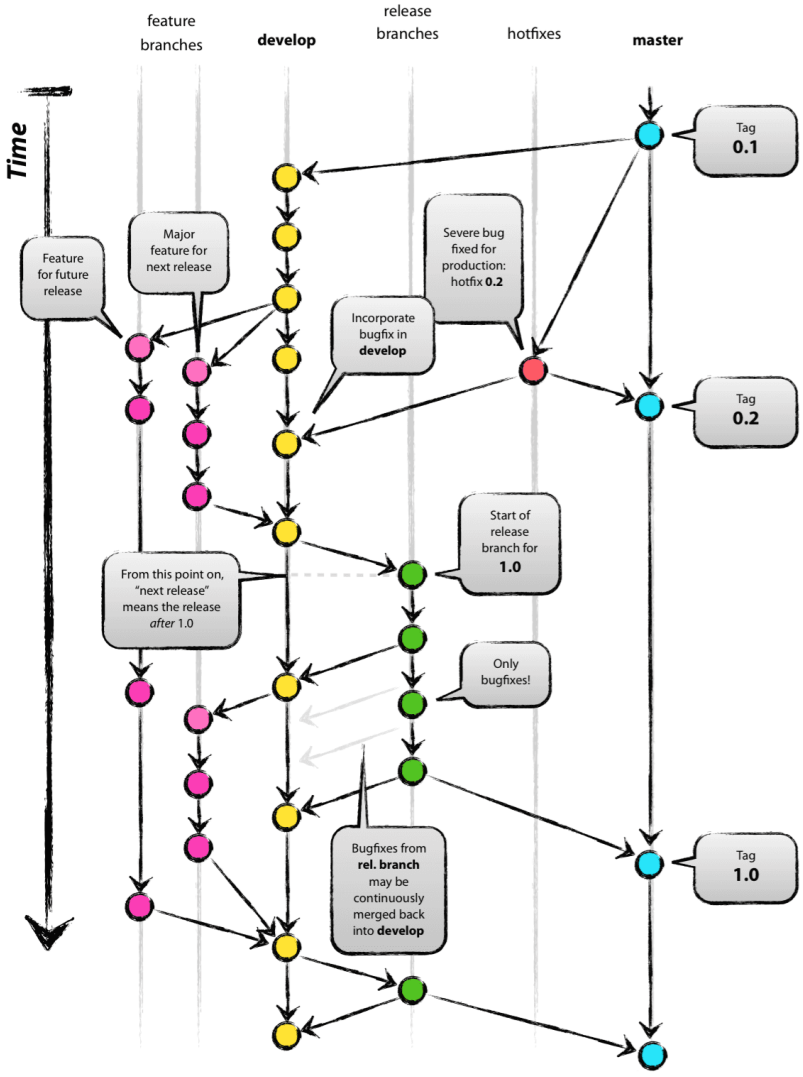
When we want the changes from master to working copy, it can be done in two ways: merging and rebasing. If we merge, then the changes on the master would be applied on to the working copy with a new commit. The second option is to rebase the entire feature branch on to the tip of master branch by fetching the changes, thus changing the actual history of commits. The benefits of rebasing are that it avoids the unnecessary merge commits required by git merge and will also yield linear project history.

BRANCHING STRATEGY:

Main branches: master and develop with unlimited lifetime. In master, the HEAD of sourcecode always reflects a production ready state while in develop it reflects a state with latest delivered development changes. Dev branch is also called as integration branch. When the sourcecode in develop branch reaches a stable point, it will be merged into master and tagged with a release number and an automatic deployment process will take place through git hooks.

Supporting branches: feature, release and bugfix branches are with limited lifetime and will be removed eventually. Feature branches are used to develop new features of the product and should origin and merge to develop branch only. These exist in developer repos (only) as long as that particular branch is under development. Release branches support the preparation of new production release and allow for minor bug fixes and meta-data preparation. It is created when the develop branch reflects the desired state of new release and thus releasing the develop branch to receive new features for next release. At this stage, version number for upcoming release is assigned. It originates from develop branch and merges to master and develop branches. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version. It originates from master and merges to develop and master and when there is an alive release version, then it has to be merged to release branch instead of develop.

.



SVN TO GIT MIGRATION

Git SVN– repository_path destination_path - this will migrate all to the current

SVN is cvs and GIT is dvcs and will have flexibility and is useful for all and allows parallel work on

Github licensing 10 users $2500 or per user on an average $250.

We can have Github in an organization in three modes – hosted, on premises and cloud.

On premises – For this mode, we will be provided with an ova file by github which is a software. Using Vm ware, (which will have vsphere client) we will deploy ova file in vsphere which will install and open the github on premises and we will integrate any of the following: active directory, ldap, saml and build-in authentication. Build-in authentication, we need to create an account evertime when I have , user name and domain name.

Github uses the Debian distro.

For on-premises Web management console we have monitoring tab, settings tab, upgrade tab, maintenance tab (We can enable schedule maintenance under maintenance tab while the server is under maintenance to intimate

the users of the same.). Under Monitoring tab, we have github enterprise backend process, iops, memory, code deployed/retrieved, statistics. Under Settings tab, we have ssh connection: only to the administrator workstation, authentication, ssl certificate (usually provided by third party), domain name setup for github access (we need to update dns records in vpshere or active directory for this sake), upgrade – details about version (current version – 2.92: as of 2017) and details about updates available (we always update only to n-1 version where n is the latest version). We can setup proxy server in settings tab.

We install ntp service in the main server and proxy server to ensure the compatibility.

Once the domain name is set, then we will login to the website using our credentials. Then we will create an organization (each for a team) and will create repositories in it and assign users and collaborators to it.

Post migration, we will hold the repositories in svn/git until two successful releases in git.

One month and will do a poc by migrating a test repo. We migrate repositories in phase-wise. We can migrate two or three repositories.

SVN TO GIT MIGRATION SCRIPT

1. Retrieve a list of all Subversion committers. ...

   From the root of your local Subversion checkout, run this command:

   svn log -q | awk -F '|' '/^r/ {sub("^ ", "", $2); sub(" $", "", $2); print $2" = "$2" <"$2">"}' | sort -u > authors-transform.txt

   This will create a authors-transform.txt file with author names which we need to edit and add the email addresses of authors.

2. Clone the Subversion repository using git-svn. ...

   git svn clone [SVN repo URL] --no-metadata -A authors-transform.txt --stdlayout ~/temp

3. Convert svn:ignore properties to .gitignore. ...

   If your svn repo was using svn:ignore properties, you can easily convert this to a .gitignore file using:

   cd ~/temp

   git svn show-ignore > .gitignore

   git add .gitignore

   git commit -m 'Convert svn:ignore properties to .gitignore.'

4. Push repository to a bare git repository. ...

   First, create a bare repository and make its default branch match svn's "trunk" branch name.

   git init --bare ~/new-bare.git

   cd ~/new-bare.git

   git symbolic-ref HEAD refs/heads/trunk

   Then push the temp repository to the new bare repository.

cd ~/temp

git remote add bare ~/new-bare.git

git config remote.bare.push 'refs/remotes/*:refs/heads/*'

git push bare

You can now safely delete the ~/temp repository.

5. Rename “trunk” branch to “master” ...

   Your main development branch will be named “trunk” which matches the name it was in Subversion. You’ll want to rename it to Git’s standard “master” branch using:

   cd ~/new-bare.git

   git branch -m trunk master

6. Clean up branches and tags. ...

   git-svn makes all of Subversions tags into very-short branches in Git of the form “tags/name”. You’ll want to convert all those branches into actual Git tags using:

   cd ~/new-bare.git

   git for-each-ref --format='%(refname)' refs/heads/tags |

   cut -d / -f 4 |

   while read ref

   do

   git tag "$ref" "refs/heads/tags/$ref";

   git branch -D "tags/$ref";

   done

This is just for one single repository, if you have multiple repositories to be migrated, then we need to have paths of all the subversion repositories in a single file and then get them invoked by a shell script which will does the migration work for you.

MAVEN

Apache Maven is a project management and build tool which converts the human readable code into machine readable format.

LIFECYCLE

There are three built-in build lifecycles: default, clean and site. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's site documentation. In the Default life cycle, maven has phases like validate, compile, test, package, verify, install and deploy.

ANT VS MAVEN

Ant	Maven
Ant **doesn't has formal conventions**, so we need to provide information of the project structure in build.xml file.	Maven **has a convention** to place source code, compiled code etc. So we don't need to provide information about the project structure in pom.xml file.
Ant is **procedural**, you need to provide information about what to do and when to do through code. You need to provide order.	Maven is **declarative**, everything you define in the pom.xml file.
There is **no life cycle** in Ant.	There is **life cycle** in Maven.
It is **a tool** box.	It is **a framework**.
It is **mainly a build tool**.	It is **mainly a project management tool**.
The ant scripts are **not reusable**.	The maven plugins are **reusable**.

PHASES VS GOALS

A goal represents a specific task which contributes to the building and managing of a project. A Build Phase is Made Up of Plugin Goals. Goals provided by plugins can be associated with different phases of the lifecycle.

DEPENDENCY MANAGEMENT

The dependency management section is a mechanism for centralizing dependency information. When you have a set of projects that inherits a common parent it's possible to put all information about the dependency in the common POM and have simpler references to the artifacts in the child POMs. A second, and very important use of the dependency management section is to control the versions of artifacts used in transitive dependencies.

MAVEN RELEASE PLUGIN

This plugin is used to release a project with Maven, saving a lot of repetitive, manual work. Releasing a project is made in two steps: prepare and perform.

- **release:clean** Clean up after a release preparation.
- **release:prepare** Prepare for a release in SCM.
- **release:prepare-with-pom** Prepare for a release in SCM, and generate release POMs that record the fully resolved projects used.
- release:rollback Rollback a previous release.
- release:perform Perform a release from SCM.
- release:stage Perform a release from SCM into a staging folder/repository.
- release:branch Create a branch of the current project with all versions updated.

- release:update-versions Update the versions in the POM(s)

The developerConnection contains the URL of the Source Control Management system pointing to the folder containing this pom.xml This URL is prefixed with scm:[scm-provider] so the plugin can pick the right implementation for committing and tagging. It is wise to do a dry run before the actual release.

REPOSITORIES

Maven repository are of three types: local, central and remote. Maven local repository is a folder location on your machine. It gets created when you run any maven command for the first time. Maven local repository keeps your project's all dependencies (library jars, plugin jars etc). It is referred to as .M2 repository.

Maven central repository is repository provided by Maven community. It contains a large number of commonly used libraries. When Maven does not find any dependency in local repository, it starts searching in central repository. Maven provides concept of Remote Repository which is developer's own custom repository containing required libraries or other project jars which can be used when Maven doesn't find the requisite dependency in both local and central repositories.

SETTINGS.XML

This file contains elements used to define values which configure Maven execution in various ways such as the local repository location, alternate remote repository servers, and authentication information, like the pom.xml, but should not be bundled to any specific project, or distributed to an audience.

Local repository, plugin registry and groups, servers, mirrors, proxies, profiles etc.

POM.XML

Project Object Model is an xml file which resides in the project base directory, contains the information about project and various configuration details used for building projects. It also contains the goals and plugins. The configuration details include project dependencies plugins, goals, build profiles, project version, developers and mailing list.

All POMs inherit from a parent (**default POM of Maven**). This base POM is known as the **Super POM**, and contains values inherited by default.

INHERITANCE VS AGGREGATION

The poms of the project would inherit the configuration specified in the super POM. This can be achieved by specifying the parent. While super POM is one example of project inheritance, own parent POMs can be introduced by specifying parent element in the POM.

Project Aggregation is similar to Project Inheritance. But instead of specifying the parent POM from the module, it specifies the modules from the parent POM. By doing so, the parent project now knows its modules, and if a Maven command is invoked against the parent project, that Maven command will then be executed to the parent's modules as well.

Pom,xml: [**Coordinates**: groupID, artifactID, version, packaging, classifier] [**POM relationships**: dependencies (groupID, artifactID, version, type, scope;compile/provided/ runtime/test/system, exclusions, Parent, dependency management] **Modules**, [**Build**: default goal, directory, final name] **Resources, Plugins, Plugin management, Reporting**, [**Organization**, developers, contributors], [**Environment settings**: issue management, CI management, Mailing list, SCM], Repositories, Plugin repositories, d Management, Profiles, Activation.

JENKINS

Jenkins is a java based CI and CD application that increases your productivity which is helpful in building and testing software projects continuously.

Jenkins Installation:

Install the Jenkins from the official Jenkins.io page. 2.46.2

Prerequisites: java and maven

Set Path: add JAVA_HOME: path to java executable file in system and user variables

Add path to java executable in path in system variables.

Jenkins Master Folder Structure

JENKINS_HOME has a fairly obvious directory structure that looks like the following:

JENKINS_HOME

```
+- config.xml     (Jenkins root configuration)
+- *.xml          (other site-wide configuration files)
+- userContent    (files in this directory will be served under your http://server/userContent/)
+- fingerprints   (stores fingerprint records)
+- plugins        (stores plugins)
+- workspace (working directory for the version control system)
   +- [JOBNAME] (sub directory for each job)
+- jobs
   +- [JOBNAME]      (sub directory for each job)
      +- config.xml     (job configuration file)
      +- latest         (symbolic link to the last successful build)
      +- builds
         +- [BUILD_ID]     (for each build)
            +- build.xml      (build result summary)
            +- log            (log file)
            +- changelog.xml  (change log)
```

DASHBOARD

New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, My Views, Credentials

DIFFERENT TYPES OF JOB CREATED

Commit jobs, Night build jobs, Deployment jobs and Release jobs.

JENKINS WEATHER

Jobs statistics

Job health	Description	Number of jobs with this health status
	No recent builds failed	3
	20-40% of recent builds failed	0
	40-60% of recent builds failed	0
	60-80% of recent builds failed	0
	All recent builds failed	0
	Unknown status	0
Total jobs	All jobs	3

JENKINS BUILD STATUS

Build statistics

Status of the build	Description	Number of builds with status
	Failed	18
	Unstable	1
	Success	92
	Pending	0
	Disabled	0
	Aborted	3
Total builds	All builds	114

JENKINS PLUGINS

Maven integration plugin, Text file operations plugin, Artifactory plugin, Thinbackup, Job Importer, Publish over SSH, Build Pipeline, Mask passwords plugin, Github plugin, Team view plugin, HTML publisher plugin, jdk parameter, green ball, color ball, docker, ec2, cft, chef, rundeck, jacoco, Cobertura, sonarqube, sonarqube quality gates, multiple scm, scm sync configuration, job configuration history, matrix authorization, build analyzer, build cross examine, pipeline,

JENKINS PIPELINE METHODOLOGY

Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline DSL.

The pipeline methodology is used for job chaining to automatically start other jobs which are dependent on a job and for rebuilding the jobs when there is a change in one of its dependencies. Let us assume that there are three jobs Project A, B and C in such a way that A is dependent on B which is dependent on C. In this illustration, While B is a downstream job to A, C is a downstream project to B. Inversely, A is an upstream job to B and B is upstream project to C.

SAMPLE WINDOWS PIPELINE

```
#!/usr/bin/env groovy
stage('compile') {
  node {
    checkout scm
    stash 'everything'
    dir('src/cafe') {
      bat 'dotnet restore'
      bat "dotnet build --version-suffix ${env.BUILD_NUMBER}"
    }
  }
}
stage('test') {
    parallel unitTests: {
      test('Test')
    }, integrationTests: {
      test('IntegrationTest')
    },
    failFast: false
}

def test(type) {
  node {
    unstash 'everything'
```

```
    dir("test/cafe.${type}") {
      bat 'dotnet restore'
      bat 'dotnet test'
    }
  }
}
stage('publish') {
  parallel windows: {
    publish('win10-x64')
  }, centos: {
    publish('centos.7-x64')
  }, ubuntu: {
    publish('ubuntu.16.04-x64')
  }
}

def publish(target) {
  node {
    unstash 'everything'
    dir('src/cafe') {
      bat "dotnet publish -r ${target}"
      archiveArtifacts "bin/Debug/netcoreapp1.1/${target}/publish/*.*"
    }
  }
}
```

Components in jenkins file - Agent, stage, steps, script

JENKINS PIPELINE BENEFITS OVER JENKINS UI

- You can't run parallel commands in the UI, just sequential.
- You can't commit it to version control and have an approval and promotion process in the UI.
- You can't know what changes were made in the Pipeline.
- On top of all these, you get more control and options than in UI
- Code review/iteration on the Pipeline

- Audit trail for the Pipeline
- Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

Before pipeline

- Many atomic jobs
- Hard to share variables/state between jobs
- Limited logic
- Job chaining
- Mix build triggers, parameterized build

WHAT IS CI?

Continuous Integration is an extreme Programming (XP) development practice where members of a team integrate their work frequently; usually each person commits their code changes at least once daily - leading to multiple integrations per day.

WHY CI IS REQUIRED?

CI allows for:

- Detect integration errors
- Identify early broken builds and fix them
- Reduced development times
- Reduction of time to fix errors

CI LIFECYCLE

All about compiling/testing/building/packaging your software on a continuous basis. With every check-in, a system triggers the compilation process, runs the unit test, and runs any static analysis tools you use and any other quality-related checks that you can automate.

JENKINS BEST PRACTICES

- Backup Jenkins periodically (Jenkins home directory)
- Use file fingerprinting to manage dependencies
- Build from source code to ensure consistent builds
- Integrate with issue management tool
- Take advantage of automated testing
- Never delete a job, if at all it is required make sure that you archive a copy of that
- Have one build for one environment
- Notify the developers with the build results
- Tag, merge or baseline your code after every successful build
- Keep your Jenkins and plugins up to date

- Don't build on master

COMMON JENKINS FAILURES

- **Maven dependency plugin:** we have two dependency plugins for Maven and at times Maven looks for old mojo which has no goal analyze. In such case, we need to redefine the goal and re-trigger the build.
- **Missing Artifact:** The reasons would be: a typo error, the referenced module would have never been built till now and that artifact is not deployed to the Artifactory and thus that module build has to be triggered and or the artifact would have be removed from the cache which happens once every few months in which case, we need to build that module again.
- **Invalid JDK version:** debug the jdk issue and add the required JDK.

CI/CD PROCESS

Build Part

Whenever Developer commits the code to mainline trunk or master in case of git and push the code to the github, Jenkins will check out the code using poll scm and it will kick off the maven scripts and maven will do the compile, test, package, install and deploy to the artifactory here we use nexus in my current project. Here I configured nexus with maven whenever we do mvn deploy, Artifacts are deployed into the nexus repository. There are again snapshot and release versions, For Continuous integration part we keep on using snapshot version, Whenever developer thinks that the development is done and says like we are good to go for the release. Then there is another build which will be kicked off called release build where it will checkout the latest code, builds the code and deploys the artifacts to the nexus release repository. Till here build part is over.

Of course, we will run code quality checks, unit test cases, unit tests, integration test and if everything is good we are going to publish into the nexus repository.

Deployment Part

Coming to deployment we need to create different environments like QA, UAT, PROD

For the deployment part also we will have a job called deploy job. Deploy job will have some parameters like environment, Component, Branch and version. Depending on the environment Jenkins will kick of the CFT templates from GIT repository and CFT will spin off the instances in AWS and

It is Integrated chef with CFT where chef will take care of provisioning of nodes where we kind of install and configure different packages.

Downloads Chef-Client package,

installs chef-client,

also download required keys like user.pem and validator.pem and configuration files for authenticating node to the chef server into /etc/chef/ directory

and bootstrap the node to register node into the chef server, (Then after we Assign the role consisting of the runlists which are having required cookbooks to configure the node for particular cookbook) and runs the chef-client on the node and deploys the artifact into the newly created environment. For example if I give the parameter as QA, QA environment is created and Deploy the artifacts into QA environment. Now we will give the QA env for the testing purposes, If the testing is done and everything is good we will promote the same code into different other environments like UAT, Stage and production.

We have deployment cookbook to pull the artifacts from artifactory and deploy to Weblogic

Continuous Delivery

Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time.

You achieve continuous delivery by continuously integrating the software done by the development team, building executables, and running automated tests on those executables to detect problems. Furthermore, you push the executables into increasingly production-like environments to ensure the software will work in production.

The principal benefits of continuous delivery are:

- **Reduced Deployment Risk:** since you are deploying smaller changes, there's less to go wrong and it's easier to fix should a problem appear.
- **Believable Progress:** many folks track progress by tracking work done. If "done" means "developers declare it to be done" that's much less believable than if it's deployed into a production (or production-like) environment.
- **User Feedback:** the biggest risk to any software effort is that you end up building something that isn't useful. The earlier and more frequently you get working software in front of real users, the quicker you get feedback to find out how valuable it really is (particularly if you use ObservedRequirements).

CHEF

Chef is a powerful automation platform that transforms infrastructure into code and automates how infrastructure is configured, deployed, and managed across your network, no matter its size. Chef helps to solve the infrastructure handling complexity.

BASIC DEFINITIONS

Organizations: Are the independent tenants in the enterprise chef and may represent companies/business units/ departments. Organizations do not share anything with others.

Environments: These model the life-stages of your application. These are the way that reflect the patterns of your organization. The stages application that goes through during its development process like development,

testing, staging, production etc. Chef generate environment creates environment folder and within which we can define each file for our environment which consists of env name, description, cookbooks and their version.

Policy: maps business and operational requirements, process, and workflow to settings and objects stored on the Chef server.

Roles: a way of identifying and classifying the different types of servers in the infrastructure like load balancer, application server, DB server etc. Roles may include configuration files (run list) and data attributes necessary for infrastructure configuration.

The **workstation** is the location from which users interact with Chef and author and test cookbooks using tools such as Test Kitchen and interact with the Chef server using the knife and chef command line tools.

Nodes are the machines—physical, virtual, cloud, and so on—that are under management by Chef. The chef-client is installed on each node which performs the automation on that machine.

The **Chef server** acts as a hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by the chef-client. Nodes use the chef-client to ask the Chef server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef server).

One (or more) workstations are configured to allow users to author, test, and maintain cookbooks. Cookbooks are uploaded to the Chef server from the workstation. Some cookbooks are custom to the organization and others are based on community cookbooks available from the Chef Supermarket.

Ruby is the programming language that is the authoring syntax for cookbooks.

The **Chef Development Kit** is a package from Chef that provides a recommended set of tooling, including Chef itself, the chef command line tool, Test Kitchen, ChefSpec, Berkshelf, and more.

A **chef-client** is an agent that runs locally on every node that is under management by Chef. When a chef-client is run, it will perform all the steps that are required to bring the node into the expected state

Runlist – A runlist, is an ordered list of roles and/or recipes that are run in the exact defined order, defines the information required for chef to configure the chef node into the desired state.

Chef Supermarket is the location in which community cookbooks are shared and managed. Cookbooks that are part of the Chef Supermarket may be used by any Chef user. It is not used to just store and access cookbooks but also things like tools, plugins, drivers, modules, DSC resource and compliance profile.

Chef management console is the user interface for the Chef server. It is used to manage data bags, attributes, run-lists, roles, environments, and cookbooks, and to configure role-based access for users and groups. Chef supermarket can resolve its dependencies which implies that it downloads all the dependent cookbooks when a cookbook is downloaded.

Cookbooks - A container used to describe the configuration data/policies about infrastructure. and contains everything that is required to support that configuration like recipes, attribute values, file distributions, templates etc. They allow code reuse and modularity.

Recipes - A recipe is a collection of resources which ensure that the system is in its desired state.

Resources – is a representation of a piece of system in infrastructure and its desired state. These are building blocks. Examples are installing packages, running Ruby code, or accessing directories and file systems.

Attributes – An attribute is a specific detail about a node which are used by the chef-client to understand the current state of the node, state of the node at the end of the previous chef-client run and desired state at the end of each chef-client run.

Definitions - A definition is used to create additional resources by stringing together one (or more) existing resources.

Files - A file distribution is a specific type of resource that tells a cookbook how to distribute files, including by node, by platform, or by file version.

Libraries - A library allows the use of arbitrary Ruby code in a cookbook, either to extend the chef-client language or to implement a new class.

Custom Resources - A custom resource is an abstract approach for defining a set of actions and (for each action) a set of properties and validation parameters.

Metadata - A metadata file is used to ensure that each cookbook is correctly deployed to each node.

Templates – An embedded ruby (erb) template which uses Ruby statements to manage configuration files.

Databags: Are generally used to hold global information pertinent to your infrastructure that are not properties of the nodes. These are used to maintain the secrets in chef. These are not specific any cookbook/recipe. They are used to store secure details like credentials which will be managed by chef-vault/HashiCorp Vault. Chef-Vault creates an extra layer of security by creating an additional encryption mechanism by providing separate set of keys to node/workstation to access the Databags.

BOOTSTRAP PROCESS

Through bootstrap process, in simple, we will be connecting the chef workstation, server and the node. When we bootstrap, the following process happens in the background:

1. We will be sshing into the node and then sending the files – Configuration file with details of chef server url, pem key path etc, and the pem key.
2. Install, configure and run chef-client on the node
3. Later, node gets registered with the chef server and its details are stored in the database server.

CHEF-DK COMPONENTS

- Foodcritic – Linting tool (syntax testing)

- Kitchen - CLI
- Chefspec – unit test (to test code – recipes)
- InSpec – Integration test (to test the functionality of the code – recipes) – used with test kitchen
- Recipes
- Cookbooks
- Knife & Chef CLI
- Ohai – tool used to detect attributes on a node
- Ruby
- Chef-Vault
- Berkshelf

CHEF SERVER COMPONENTS

- API – chef API
- Datastore – which stores the data pertaining to recipes, nodes, environment, roles etc.
- Search – Knife search <index> which has 5 indexes: -
    - API Client
    - Node
    - Role
    - Environment
    - Databags
- Cookbooks – Stored in bookshelf in server
- Supermarket
- Runlist
- Policy – which is a role-based access control

FOODCRITIC

Foodcritic is used to check cookbooks for common problems: Style, Correctness, Syntax, Best practices, Common mistakes and Deprecations. Foodcritic does not validate the intention of a recipe, rather it evaluates the structure of the code

Syntax: Foodcritic path_to_the_cookbook

BERKSHELF

Berkshelf is a dependency manager for Chef cookbooks. With it, you can easily depend on community cookbooks and have them safely included in your workflow. Berkshelf is included in the Chef Development Kit.

You add the dependencies of your cookbook to metadata.rb and then you run berks install to get those dependency cookbooks to be downloaded from supermarket to cache. A Berksfile describes the set of sources and dependencies needed to use a cookbook. It is used in conjunction with the berks command. By default, a Berksfile has a source for Chef's public supermarket.

CHEFSPEC

ChefSpec is a framework that tests resources and recipes as part of a simulated chef-client run. ChefSpec tests execute very quickly. ChefSpec tests are often the first indicator of problems that may exist within a cookbook. ChefSpec is packaged as part of the Chef development kit.

Chef exec rspec

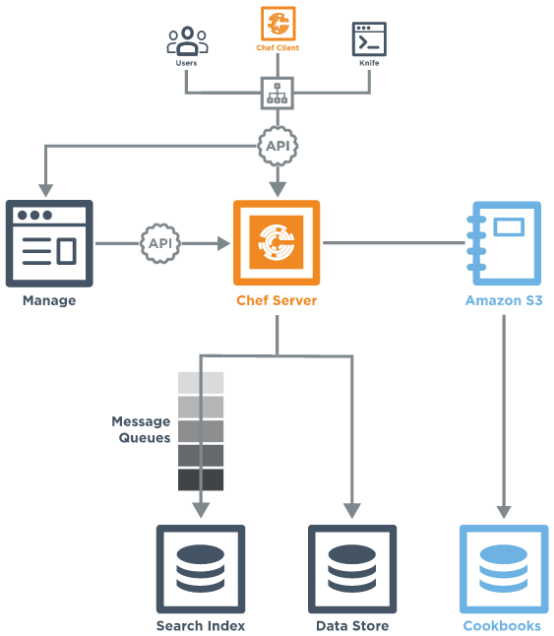
TESTKITCHEN

Use Test Kitchen to automatically test cookbook data across any combination of platforms and test suites:

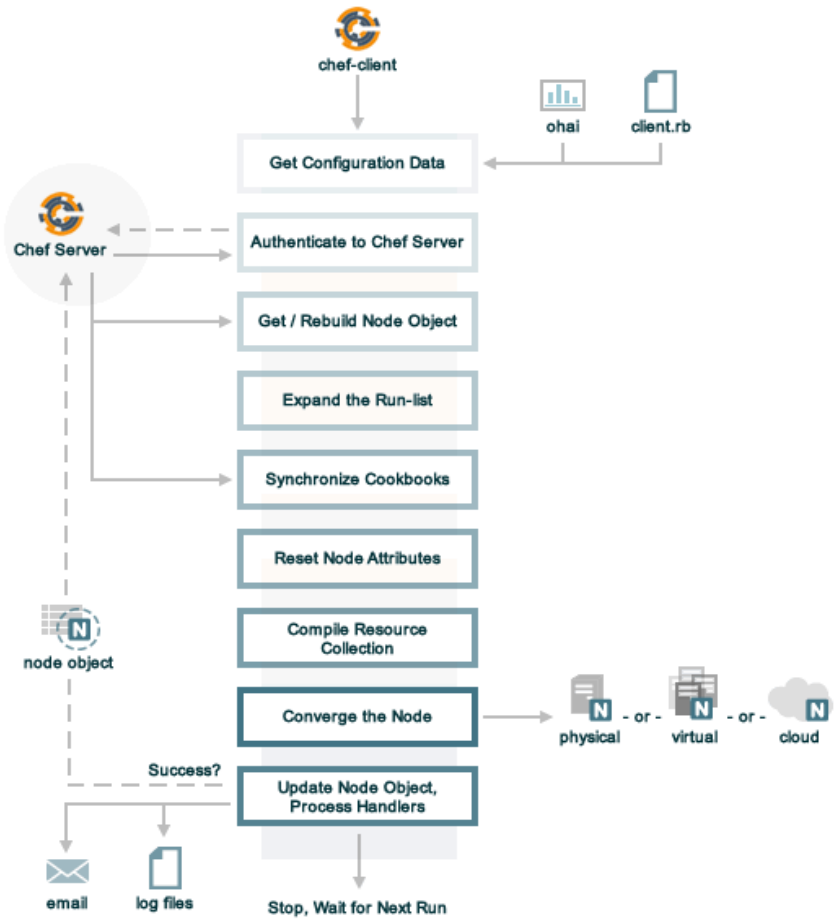
- Defined in a .kitchen.yml file
- Uses a driver plugin architecture
- Supports cookbook testing across many cloud providers and virtualization technologies
- Supports all common testing frameworks that are used by the Ruby community
- Uses a comprehensive set of base images provided by Bento

Use a kitchen.yml file to define what is required to run Kitchen, including drivers, provisioners, verifiers, platforms, and test suites. The stages in test kitchen lifecycle are: create, converge, login, verify, destroy and diagnose.

CHEF ECOSYSTEM



CHEF_CLIENT RUN



COOKBOOK FOLDER STRUCTURE

- Knife cookbook create cookbook_name – this is deprecated and we should use Chef generate command

```
drwxr-xr-x.  3 root root   20 Jun  1 09:28 templates
drwxr-xr-x.  2 root root    6 Jun  1 09:28 resources
drwxr-xr-x.  2 root root    6 Jun  1 09:28 providers
drwxr-xr-x.  2 root root    6 Jun  1 09:28 libraries
drwxr-xr-x.  3 root root   20 Jun  1 09:28 files
drwxr-xr-x.  2 root root    6 Jun  1 09:28 definitions
drwxr-xr-x.  2 root root   23 Jun  1 09:28 recipes
drwxr-xr-x.  2 root root   23 Jun  1 09:28 attributes
-rw-r--r--.  1 root root 1472 Jun  1 09:28 README.md
-rw-r--r--.  1 root root  282 Jun  1 09:28 metadata.rb
-rw-r--r--.  1 root root  463 Jun  1 09:28 CHANGELOG.md
```

- Chef cookbook generate cookbook_name

```
drwxr-xr-x. 3 root root   24 Jun  1 09:28 test
drwxr-xr-x. 3 root root   38 Jun  1 09:28 spec
drwxr-xr-x. 2 root root   23 Jun  1 09:28 recipes
drwxr-xr-x. 7 root root 4096 Jun  1 09:28 .git
-rw-r--r--. 1 root root  231 Jun  1 09:28 metadata.rb
-rw-r--r--. 1 root root   65 Jun  1 09:28 README.md
-rw-r--r--. 1 root root 1067 Jun  1 09:28 chefignore
-rw-r--r--. 1 root root   47 Jun  1 09:28 Berksfile
-rw-r--r--. 1 root root  343 Jun  1 09:28 .kitchen.yml
-rw-r--r--. 1 root root  126 Jun  1 09:28 .gitignore
```

ATTRIBUTES PRECEDENCE

Sometimes you might use hard-coded values (for example, directory name, filename, username, etc.) at multiple locations inside your recipes. Later when you want to change this value, it becomes a tedious process, as you must browse through all the recipes that contains this value and change them accordingly. Instead, you can define the hard-code value as variable inside an attribute file, and use the attribute name inside the recipe. This way when you want to change the value, you are changing only at one place in the attribute file.

The attribute precedence is as follows:

1. A default attribute located in a cookbook attribute file
2. A default attribute located in a recipe
3. A default attribute located in an environment
4. A default attribute located in a role
5. A force_default attribute located in a cookbook attribute file
6. A force_default attribute located in a recipe
7. A normal attribute located in a cookbook attribute file
8. A normal attribute located in a recipe
9. An override attribute located in a cookbook attribute file
10. An override attribute located in a recipe
11. An override attribute located in a role
12. An override attribute located in an environment
13. A force_override attribute located in a cookbook attribute file

14. A force_override attribute located in a recipe
15. An automatic attribute identified by Ohai at the start of the chef-client run

TYPES OF COOKBOOKS

1. Library cookbooks: An existing cookbook, typically an open-source contribution from a user in the Chef community, designed for server configuration purposes. Eg: database, GitHub, libarchive, artifact etc.
2. Application Cookbooks: Which contains at least one recipe which installs a piece of software and shares the same name as the cookbook. Eg: mysql. Nginix etc.
3. Wrapper cookbook: Depends on an application cookbook and exposes the required /recipes from each cookbook.

   Provisioning, configuration, deployment cookbooks

PREDEFINED RESOURCES

Apt_package: Use the apt_package resource to manage packages for the Debian and Ubuntu platforms.

Apt_repository: Use the apt_repository resource to additional APT repositories. Adding a new repository will update apt package cache immediately.

Apt_update: Use the apt_update resource to manage Apt repository updates on Debian and Ubuntu platforms.

Bash: Use the bash resource to execute scripts using the Bash interpreter. This resource may also use any of the actions and properties that are available to the execute resource.

Batch: Use the batch resource to execute a batch script using the cmd.exe interpreter. The batch resource creates and executes a temporary file (like how the script resource behaves), rather than running the command inline.

Bff_package: Use the bff_package resource to manage packages for the AIX platform using the install utility.

Breakpoint: Use the breakpoint resource to add breakpoints to recipes. Run the chef-shell in chef-client mode, and then use those breakpoints to debug recipes. Breakpoints are ignored by the chef-client during an actual chef-client run. That said, breakpoints are typically used to debug recipes only when running them in a non-production environment, after which they are removed from those recipes before the parent cookbook is uploaded to the Chef server.

Chef_gem: Use the chef_gem resource to install a gem only for the instance of Ruby that is dedicated to the chef-client. When a gem is installed from a local file, it must be added to the node using the remote_file or cookbook_file resources.

The chef_gem resource works with all the same properties and options as the gem_package resource, but does not accept the gem_binary property because it always uses the CurrentGemEnvironment under which the chef-client is running. In addition to performing actions like the gem_package resource, the chef_gem resource does the following:

Runs its actions immediately, before convergence, allowing a gem to be used in a recipe immediately after it is installed

Runs Gem.clear_paths after the action, ensuring that gem is aware of changes so that it can be required immediately after it is installed

Chef_handler: Use the chef_handler resource to enable handlers during a chef-client run. The resource allows arguments to be passed to the chef-client, which then applies the conditions defined by the custom handler to the node attribute data collected during the chef-client run, and then processes the handler based on that data.

Chocolatey_package: Use the chocolatey_package resource to manage packages using Chocolatey for the Microsoft Windows platform.

Cookbook_file: Use the cookbook_file resource to transfer files from a sub-directory of COOKBOOK_NAME/files/ to a specified path located on a host that is running the chef-client.

Cron: Use the Cron resource to manage cron entries for time-based job scheduling. Properties for a schedule will default to * if not provided. The cron resource requires access to a crontab program, typically cron.

Csh: Use the csh resource to execute scripts using the csh interpreter. This resource may also use any of the actions and properties that are available to the execute resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not_if and only_if to guard this resource for idempotence.

Deploy: Use the deploy resource to manage and control deployments. This is a popular resource, but is also complex, having the most properties, multiple providers, the added complexity of callbacks, plus four attributes that support layout modifications from within a recipe.

Directory: Use the directory resource to manage a directory, which is a hierarchy of folders that comprises all the information stored on a computer. The root directory is the top-level, under which the rest of the directory is organized. The directory resource uses the name property to specify the path to a location in a directory. Typically, permission to access that location in the directory is required

Recursive Directories: The directory resource can be used to create directory structures, if each directory within that structure is created explicitly. This is because the recursive attribute only applies group, mode, and owner attribute values to the leaf directory.

Dpkg_package: Use the dpkg_package resource to manage packages for the dpkg platform. When a package is installed from a local file, it must be added to the node using the remote_file or cookbook_file resources.

Dsc_resource: Desired State Configuration (DSC) is a feature of Windows PowerShell that provides a set of language extensions, cmdlets, and resources that can be used to declaratively configure software. The dsc_resource resource allows any DSC resource to be used in a Chef recipe, as well as any custom resources that

have been added to your Windows PowerShell environment. Microsoft frequently adds new resources to the DSC resource collection.

Dsc_script: The dsc_script resource is most useful for those DSC resources that do not have a direct comparison to a resource in Chef, such as the Archive resource, a custom DSC resource, an existing DSC script that performs an important task, and so on. Use the dsc_script resource to embed the code that defines a DSC configuration directly within a Chef recipe.

Env: Use the env resource to manage environment keys in Microsoft Windows. After an environment key is set, Microsoft Windows must be restarted before the environment key will be available to the Task Scheduler.

Erl_call: Use the erl_call resource to connect to a node located within a distributed Erlang system. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not_if and only_if to guard this resource for idempotence.

Execute: Use the execute resource to execute a single command. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not_if and only_if to guard this resource for idempotence.

File: Use the file resource to manage files directly on a node.

Freebsd_package: Use the freebsd_package resource to manage packages for the FreeBSD platform. A freebsd_package resource block manages a package on a node, typically by installing it.

Gem_package: Use the gem_package resource to manage gem packages that are only included in recipes. When a package is installed from a local file, it must be added to the node using the remote_file or cookbook_file resources.

Git: Use the git resource to manage source control resources that exist in a git repository. git version 1.6.5 (or higher) is required to use all the functionality in the git resource.

Group: Use the group resource to manage a local group.

Homebrew_package: Use the homebrew_package resource to manage packages for the macOS platform.

http_request: Use the http_request resource to send an HTTP request (GET, PUT, POST, DELETE, HEAD, or OPTIONS) with an arbitrary message. This resource is often useful when custom callbacks are necessary.

Ifconfig: Use the ifconfig resource to manage interfaces.

Ips_package: Use the ips_package resource to manage packages (using Image Packaging System (IPS)) on the Solaris 11 platform.

Link: Use the link resource to create symbolic or hard links.

Log: Use the log resource to create log entries. The log resource behaves like any other resource: built into the resource collection during the compile phase, and then run during the execution phase. (To create a log entry that is not built into the resource collection, use Chef::Log instead of the log resource.)

Macports_package: Use the macports_package resource to manage packages for the macOS platform.

Mdadm: Use the mdadm resource to manage RAID devices in a Linux environment using the mdadm utility. The mdadm provider will create and assemble an array, but it will not create the config file that is used to persist the array upon reboot. If the config file is required, it must be done by specifying a template with the correct array layout, and then by using the mount provider to create a file systems table (fstab) entry.

Mount: Use the mount resource to manage a mounted file system.

Ohai: Use the ohai resource to reload the Ohai configuration on a node. This allows recipes that change system attributes (like a recipe that adds a user) to refer to those attributes later during the chef-client run.

Openbsd_package: Use the openbsd_package resource to manage packages for the OpenBSD platform.

Osx_profile: Use the osx_profile resource to manage configuration profiles (.mobileconfig files) on the macOS platform. The osx_profile resource installs profiles by using the uuidgen library to generate a unique ProfileUUID, and then using the profiles command to install the profile on the system.

Package: Use the package resource to manage packages. When the package is installed from a local file (such as with RubyGems, dpkg, or RPM Package Manager), the file must be added to the node using the remote_file or cookbook_file resources.

Pacman_package: Use the pacman_package resource to manage packages (using pacman) on the Arch Linux platform.

Perl: Use the perl resource to execute scripts using the Perl interpreter. This resource may also use any of the actions and properties that are available to the execute resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not_if and only_if to guard this resource for idempotence.

Powershell_script: Use the powershell_script resource to execute a script using the Windows PowerShell interpreter, much like how the script and script-based resources—bash, csh, perl, python, and ruby—are used. The powershell_script is specific to the Microsoft Windows platform and the Windows PowerShell interpreter.

Python: Use the python resource to execute scripts using the Python interpreter. This resource may also use any of the actions and properties that are available to the execute resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not_if and only_if to guard this resource for idempotence.

Reboot: Use the reboot resource to reboot a node, a necessary step with some installations on certain platforms. This resource is supported for use on the Microsoft Windows, macOS, and Linux platforms. New in Chef Client 12.0.

Registry_key: Use the registry_key resource to create and delete registry keys in Microsoft Windows.

Remote_directory: Use the remote_directory resource to incrementally transfer a directory from a cookbook to a node. The directory that is copied from the cookbook should be located under COOKBOOK_NAME/files/default/REMOTE_DIRECTORY. The remote_directory resource will obey file specificity.

Remote_file: Use the remote_file resource to transfer a file from a remote location using file specificity. This resource is similar to the file resource.

Route: Use the route resource to manage the system routing table in a Linux environment.

Rpm_package: Use the **rpm_package** resource to manage packages for the RPM Package Manager platform.

Ruby: Use the ruby resource to execute scripts using the Ruby interpreter. This resource may also use any of the actions and properties that are available to the execute resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not_if and only_if to guard this resource for idempotence.

Ruby_block: Use the ruby_block resource to execute Ruby code during a chef-client run. Ruby code in the ruby_block resource is evaluated with other resources during convergence, whereas Ruby code outside of a ruby_block resource is evaluated before other resources, as the recipe is compiled.

Script: Use the script resource to execute scripts using a specified interpreter, such as Bash, csh, Perl, Python, or Ruby. This resource may also use any of the actions and properties that are available to the execute resource. Commands that are executed with this resource are (by their nature) not idempotent, as they are typically unique to the environment in which they are run. Use not_if and only_if to guard this resource for idempotence.

Service: Use the service resource to manage a service.

Smartos_package: Use the smartos_package resource to manage packages for the SmartOS platform.

Solaris_package: The solaris_package resource is used to manage packages for the Solaris platform.

Subversion: Use the subversion resource to manage source control resources that exist in a Subversion repository.

Systemd_unit: Use the systemd_unit resource to create, manage, and run systemd units.

Template: A cookbook template is an Embedded Ruby (ERB) template that is used to dynamically generate static text files. Templates may contain Ruby expressions and statements, and are a great way to manage configuration files. Use the template resource to add cookbook templates to recipes; place the corresponding Embedded Ruby (ERB) template file in a cookbook's /templates directory.

To use a template, two things must happen:

- A template resource must be added to a recipe
- An Embedded Ruby (ERB) template must be added to a cookbook

User: Use the user resource to add users, update existing users, remove users, and to lock/unlock user passwords.

Windows_package: Use the windows_package resource to manage Microsoft Installer Package (MSI) packages for the Microsoft Windows platform.

Windows_service: Use the windows_service resource to manage a service on the Microsoft Windows platform. New in Chef Client 12.0.

Yum_package: Use the yum_package resource to install, upgrade, and remove packages with Yum for the Red Hat and CentOS platforms. The yum_package resource is able to resolve provides data for packages much like Yum can do when it is run from the command line. This allows a variety of options for installing packages, like minimum versions, virtual provides, and library names.

Yum_repository: Use the yum_repository resource to manage a Yum repository configuration file located at /etc/yum.repos.d/repositoryid.repo on the local machine. This configuration file specifies which repositories to reference, how to handle cached data, etc.

Zypper_package: Use the zypper_package resource to install, upgrade, and remove packages with Zypper for the SUSE Enterprise and OpenSUSE platforms.

Linux Academy – Chef Development:

Chef Generate

- Generators are used to quickly generate cookbooks or components of a cookbook. Generators are invoked by using the `chef generate` command.

- What can `chef generate` create?
  - `app` – An application repository
  - `cookbook` – A single single cookbook
  - `recipe` – A single recipe
  - `attribute` – An attributes file
  - `template` – A file template
  - `file` – A cookbook file
  - `lwrp` – A lightweight resource/provider
  - `repo` – A Chef code repository
  - `policyfile` – A policyfile for use with the install/push commands
  - `generator` – A copy of the ChefDK generator cookbook so you can customize it
  - `build-cookbook` – For use with Delivery

Generating with a Generator

- Using `chef generate generator <my_generator>` will create a copy of the ChefDK generator which we can use to customize to our specifications.
- Generating a generator will help enforce standards on the contents of a newly created cookbook or cookbook component to ensure it meets organizational requirements.
- Use cases for generators include:
  - Boilerplate text in README or generated recipes
  - Default licensing choice
  - Commonly used code

RSpec and ChefSpec Syntax

- ChefSpec is made available by including a `require 'chefspec'` statement which loads ChefSpec matchers. The default configuration when generating a cookbook is to include this in `spec_helper.rb`.
- The `describe` keyword followed by a cookbook and recipe name tells ChefSpec what to run.
- The `let` keyword is a block which defines how the chef-client run is simulated. SoloRunner or ServerRunner.
- The `it` keyword is the opening block for a test.
- The `expect` keyword sets up an expectation for a resource can be used in conjunction with `to` or `to_not` methods.

```
require 'chefspec'

describe 'example::default' do
  let(:chef_run) { ChefSpec::SoloRunner.converge(described_recipe) }

it 'does something' do
  expect(chef_run).to ACTION_RESOURCE(NAME)
  end
end
```

Linux Academy

RSpec and ChefSpec Syntax, Cont.

- The `context` keyword describes a block of tests and can be used to group tests together, this is useful for differentiating sets of tests for specific platforms.

```
describe 'lcd_web::default' do
  context 'CentOS' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(platform: 'centos', version: '7.2.1511')
      runner.converge(described_recipe)
    end

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'installs httpd'
      expect(chef_run).to install_package 'httpd'
    end

  end

  context 'Ubuntu' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(platform: 'ubuntu', version: '16.04')
      runner.converge(described_recipe)
    end

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'installs apache2'
      expect(chef_run).to install_package 'apache2'
    end

  end
end
```

Linux Academy

```
#
# Cookbook:: lcd_web
# Spec:: default
#
# Copyright:: 2017, The Authors, All Rights Reserved.

require 'spec_helper'

describe 'lcd_web::default' do
  context 'CentOS' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new(platform: 'centos', version: '7.2.1511')
      runner.converge(described_recipe)
    end

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'installs httpd' do
      expect(chef_run).to install_package('httpd')
    end

    it 'enables the httpd service' do
      expect(chef_run).to enable_service('httpd')
    end

    it 'starts the httpd service' do
      expect(chef_run).to start_service('httpd')
    end


  end
end
~
~
~
~
-- INSERT --                                                                    30,8          All
```


.kitchen.yml Structure

```
driver:
  name: driver_name

provisioner:
  name: provisioner_name

verifier:
  name: verifier_name

transport:
  name: transport_name

platforms:
  - name: platform-version
    driver:
      name: driver_name
  - name: platform-version

suites:
  - name: suite_name
    run_list:
      - recipe[cookbook_name::recipe_name]
    attributes: { foo: "bar" }
    excludes:
      - platform-version
  - name: suite_name
    driver:
      name: driver_name
    run_list:
      - recipe[cookbook_name::recipe_name]
    attributes: { foo: "bar" }
    includes:
      - platform-version
```

An Example of an Iterative Development Workflow

1. Set up test kitchen by editing on your kitchen.yml.
2. Write the relevant tests.
3. Run `kitchen verify` (implies create, converge and setup)
4. Examine your failures.
5. Write the Chef code to pass your tests.
6. Run `kitchen converge` to apply your new cookbook code.
7. Run `kitchen verify` to see that you have passed your tests.
8. Iterate on new functionality by writing additional tests.
9. Write Chef to code pass your additional tests.
10. Run `kitchen verify` to ensure your new functionality correctly implemented.
11. When satisfied that your work is complete run `kitchen destroy`.

InSpec Syntax - Matchers

- InSpec matchers are used to compare resource values to expectations
  - be – Can be followed by a numerical comparison operator.
  - cmp – Flexible comparison operator.
  - eq – Test for equality between two values of the same type.
  - include – Test if a value is included in a list.
  - match – Check if a string matches a regular expression.

- Each *resource* can define additional resource specific matchers relavant to that *resource*.
  - For example the *package* resource defines a matcher called `be_installed`.
  - The *file* resource defines a matcher called `be_writable`.
  - Check the resource reference documentation for matcher functionality related to that resource.

InSpec User Resource Example

- Common resource example with 'its' to compare the value of a setting with a matcher. Additional tests of a resource could look as follows:

```
describe user('david') do
  it { should exist }
  its('uid') { should eq 1234 }
  its('gid') { should eq 1234 }
  its('group') { should eq 'root' }
  its('groups') { should eq ['wheel', 'users']}
  its('home') { should eq '/home/david' }
  its('shell') { should eq '/bin/bash' }
end
```

Rubocop Workflow

- When working with Rubocop you can use a basic workflow to help work through issues it finds.

1. Within your cookbook run: `rubocop --auto-gen-config`
2. Add `inherit_from: .rubocop_todo.yml` to `.rubocop.yml` assuming it's in the same directory.
3. Remove or comment out a single entry within the `.rubocop_todo.yml`.
4. Run `rubocop` and fix the reported offense.
5. Verify it has been fixed by running `rubocop` again to ensure it is no longer present.
6. Repeat steps 3 through 5 until there are no issues left.
7. Remove the .rubocop_todo.yml when done.

- You might also choose to move offenses which you are not concerned about into the .rubocop.yml to ensure they are not reported in the future.

Rubocop Output

- Rubocop's default output format is a progress meter mode which will display a "." for a clean file and capital letters in the form of C for convention, W for warning, E for error or F for fatal. This shows you at a glance how many issues you have and the kinds of issues.

- Rubocop will display some metrics, including the number of offenses and files checked. Offenses will appear with the filename, line number and character number where the issue is identified. Also included will be the category of error C,W,E or F and a description of the offense in practice this might appear as follows:

```
Inspecting 13 files
..........C.
Offenses:
recipes/default.rb:7:13: C: Space missing after comma.
['net-tools','httpd'].each do |pkg|
            ^
13 files inspected, 1 offense detected
```


AWS

Cloud services refers to various IT resources provided on demand by a service provider to its clients over the internet. It also covers the professional services that assist the selection, deployment and management of various cloud-based resources. Characteristics include scalability, on-demand, self-provisioning etc.

Cloud service providers like AWS include: Microsoft Azure, Google Cloud Platform, IBM Cloud, Rackspace, Oracle and Verizon clouds.

Services within AWS used are: Ec2, VPC, IAM, Elastic Beanstalk, Cloud watch, Autoscaling and Route 53.

PROVISIONING AN EC2 INSTANCE

An Ec2 instance can be provisioned by the following steps:

- Choose an AMI which is an acronym for server template which is a typical OS with few packages installed.

- In the next step choose the instance type which defines the size of the virtual server that we are launching. The t and m families are for general use while the c family for compute optimized family, g family for graphic processing optimized, r family is for memory optimized and the d and I families are memory optimized.
- The third step allows you to configure various things for the server like:
  - Number of instances
  - Vpc and subnet
  - Auto assign public IP and IAM role
  - Shutdown behavior, enable termination protection and option to enable cloudwatch
- In step four, we can configure the attached disk volume for the server where the default is general purpose SSD and we can also choose either Iops based SSD or magnetic storage. We can provision the size apart from the default 8 gb.
- In the next step we can tag the instance based on the organization's naming convention like name, department, etc.
- In the next step we will be configuring the security groups which determine which ports are open on the server and the source from which they can be accessed. In this step we can either choose from an existing security group or create a new security group.
- Finally, we will be reviewing and launching the server, post which we will be either choosing an existing key pair or generating and downloading new key pair.

BASIC AWS SERVICES

EC2 – refers to the web service which provides scalable computing capacity – literally servers on Amazon's data center - that is used to build and host the software applications.

VPC – This enables to launch AWS resources into a virtual network defined by us which closely resemble a traditional network with the benefits of scalable AWS resources.

RDS – this service makes it easier to set up, operate and scale a cost-efficient and resizable relational database in cloud and handles the regular administration tasks.

IAM – refers to the service which assists in securing the access to AWS resources for users by authentication and authorization processes where authentication refers to who can use AWS resources while authorization refers to what resources can be used and in what ways.

Elastic Beanstalk – using which we can easily deploy and manage applications in AWS cloud without worrying about the infrastructure required as it reduces the management complexity without restricting choice or control. Through this service, we will be just uploading the application and it automatically handles the capacity, provisioning, load balancing, scaling and application health monitoring.

Auto scaling – refers to the service designed to launch and terminate ec2 instances automatically based on user-defined policies and schedules which defined for ensuring the smooth functioning of the application. Auto scaling groups, minimum/desired/maximum instances and scaling policy will be defined.

Elastic load balancing – this service distributes the incoming application traffic across multiple targets such as EC2 instances through routing the traffic to healthy targets by monitoring their health. It is the single point of contact for clients and enhances the application availability.

S3- It is the simple storage service which can be used to store and retrieve any amount of data at any time and from anywhere on the web. S3 is a scalable, high-speed, low-cost, web-based object storage service designed for online backup and archiving of data and application programs.

Elastic Block Store – EBS provides highly available and reliable block level storage volumes which can be used with EC2 instances in the same availability zones. EBS is suggested for quick access and long-term persistence.

Route 53 – is a highly available and scalable Domain Naming Service (DNS) and performs three main functions – register domain names, route internet traffic to the resources for your domain and check the health of your resources.

Cloudwatch – provides a reliable, scalable, and flexible monitoring solution through which we can monitor AWS resources and applications ran on AWS. Using which we can either send notifications and or automatically make changes to the resource according to predefined monitoring based rules.

Cloudfront - web service that speeds up distribution of your static and dynamic web content, such as .html, .css, .php, and image files, to your users. CloudFront delivers your content through a worldwide network of data centers called edge locations. When a user requests content that you're serving with CloudFront, the user is routed to the edge location that provides the lowest latency (time delay), so that content is delivered with the best possible performance.

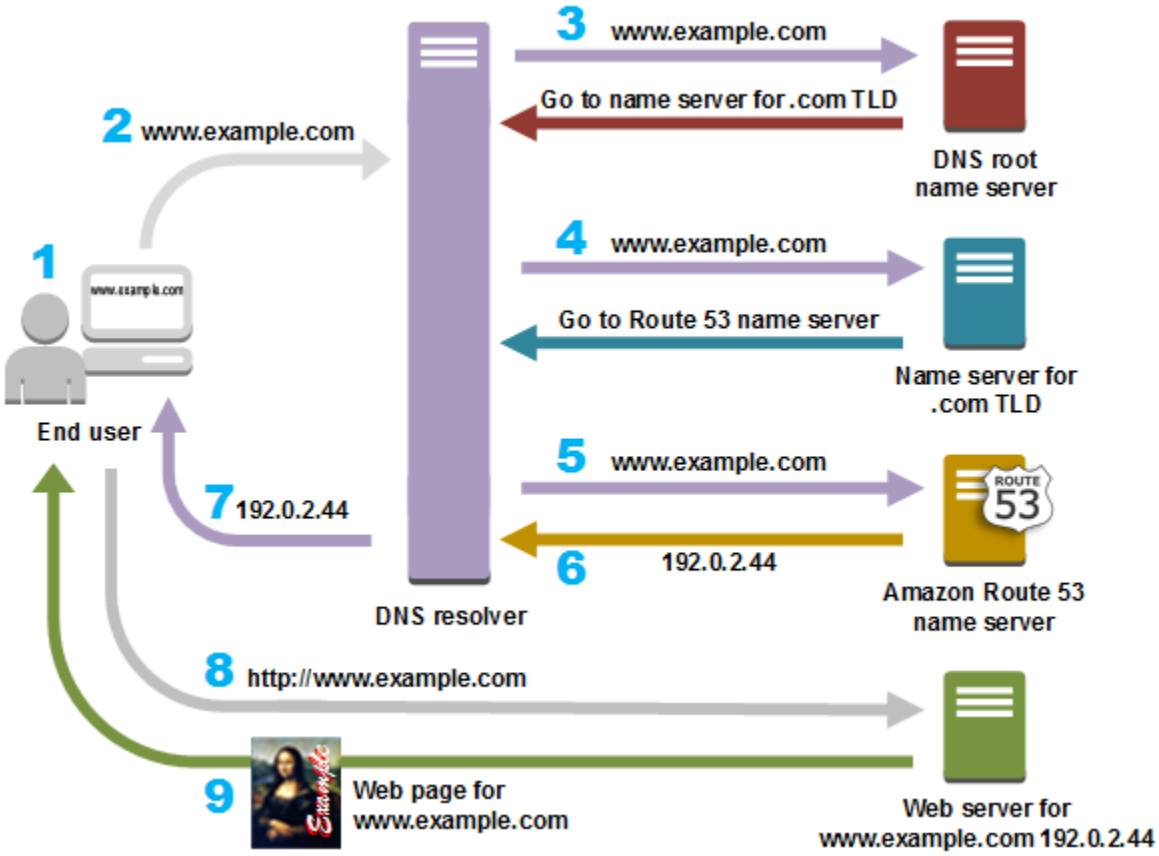
Cloudformation - enables you to create and provision AWS infrastructure deployments predictably and repeatedly. It helps you leverage AWS products such as Amazon EC2, Amazon Elastic Block Store, Amazon SNS, Elastic Load Balancing, and Auto Scaling to build highly reliable, highly scalable, cost-effective applications in the cloud without worrying about creating and configuring the underlying AWS infrastructure. AWS CloudFormation enables you to use a template file to create and delete a collection of resources together as a single unit (a stack).

AWS ROUTE 53

- Launch an instance
- Create an ELB and choose security group so that it allows port 80, configure health checks, add the ec2 instance.
- Go to route 53, get started, create hosted zone where we enter the domain name (ex-

www.aws.com), type public hosted zone

- Creates two records by default – start of authority record and name server records
- Now add these name server records in the domain details of your domain registrar.
- Create naked domain name or apex record (without www – example.com)
- Since we don't have a public ip of load balancer, we create an alias record and choose alias record and the routing policy (simple, weighted, latency, failover and geolocation) and choose whether you wish to evaluate the health of resources or not.



AWS MACHINE TYPES

- M – general purpose
- C – Compute optimized
- R – Memory optimized
- D/I – Storage optimized D – large storage I – large I/O
- G – GPU instances
- T – Micro instances

AWS S3

- Simple storage service – object based stored and a key value store
- Highly-scalable, reliable, and low-latency data storage infrastructure at very low costs
- Per object limit is 0 – 5 TB
- Object with more than 100 MB should follow Multi-part upload – which uploads the parts of huge object parallelly and then stores them in compatible manner.
- Data is spread across multiple device and facilities
- Files are stored in Buckets and S3 name should be unique
- Link: https://s3-region-name.amazonaws.com/bucket-name
- When you upload an object for the first time, then it is ready to be read immediately and from next time, it takes time to propagate (which we refer to as eventual consistency).
- The objects in S3 have: Key (name), Value (data), version ID, Metadata, subresources and Access control lists.
- 99.99% availability and 11X9's durability (reverse of losing an object)
- S3 storage tiers/Classes:
  - Standard tier – 99.99% availability, 11 9's durability and sustain 2 facility loss
  - Infrequently accessed – lower fee than S3 standard and availability 99.9%
  - Reduced Redundancy Storage – 99.99% durability and objects
  - Glacier – used for archival only and takes 3-5 hours to restore objects from glacier
- Bucket is a container for objects stored in S3 and in other words we can refer to it as folder and its name had to be unique (should be in lowercase)
- Buckets has four section objects, properties, permissions and management.
  - Under objects we can find the contents/objects of the bucket
  - Under properties we will find the versioning, logging, static website hosting, tags and events (notification)
  - Under permission we have access control list (manage users and manage public permissions at an object level), bucket policy – a easy way to grant cross-account access to your s3 environment through permissions and Cross Origin Resource Sharing (CORS) – which regions should the bucket be accessible from (versioning should be enabled).
  - Under management we have lifecycle, analysis, metrics and inventory.
- Versioning allows you to preserve and retrieve and restore version of every object of the S3 bucket. Once enabled cannot be disabled but can be just suspended.

- Lifecycle management s3: We can add a rule either to the whole bucket or folders within the bucket for lifecycle management of the objects which reduce the costs. Lifecycle rules allow the objects to be automatically changed within the available storage tiers. The flow will be standard to infrequent and from infrequent to glacier and later, from glacier it will be deleted, with a 30 day minimum life at each tier.
  - It can used in conjunction with versioning
  - Applied to current and previous versions
- Encryption: SSL/TLS for securing data in transit and for data at rest, we use: server side encryption – S3 managed Keys (SSE-S3), AWS KMS managed keys – SSE-KMS (additional charges), Server side encryption with customer provided keys – SSE-c and finally client side encryption.
- Storage Gateway: a service which connects the on-premise software with cloud-based storage and secures integration while enabling the storing of data to the cloud. AWS storage gateway S/W appliance is available for download as a VM image which can be installed and activated on local datacenter and it support VMware ESXi or Microsoft Hyper-V
  - Three types of storage gateways:
    - GW stored volumes: data is on site and is backed up on to the Amazon S3, it is durable and inexpensive
    - GW Cached volumes: most frequently accessed data is stored locally and data set is stored in S3.
    - GW virtual tape library: Virtual tapes on site can be stored in virtual tape library backed by S3 or shelf backed by Amazon glacier.
    - Import/Export: two types IE Disk and IE snowball. Through IE disk, you export your data onto AWS using portable storage devices. Snowball is a petabyte-scale data transport solution by AWS. Supports around TB per snowball and addresses challenges with large-scale data transfers, high network costs, long transfer time and security concerns.

AWS IAM

Identity Access Management: allows you to manage users and the access to the AWS Console. It gives:

- Centralized control and shared access of AWS account
- Enables granular permissions, identity Federation – where we provide access to AWS account through temporary security credentials (active directory, facebook etc) and Multifactor authentication for users
- Provide temporary access for users, devices and services and allows to set own password rotation policy

Policies can be applied to users, groups and roles which defines one or more permissions and a policy document sits on top of these three.

- IAM is not region specific. We usually provide a sign-in link (a customized one) to all IAM users through which they can access the AWS account. **Access Key ID** refers to the username and **secret access key** is the password which is used to access AWS console using CLI.
- We are required to download the security credentials of users and store them in a secure location.
- By default, users willn't have any permissions.
- Policy documents are authored in JSON format.
- Role types in IAM are: service roles, cross-account access and identity provider access.
- Role can't be assigned to groups.
- 250 is the IAM role limit per account and can be increased by request.
- Three types of policies: Managed, custom and inline policies.

AWS CLOUDFORMATION

We can validate a cft by checking file for syntax errors by using **aws cloudformation validate-template --template-body/file**.

We can be notified when the resource creation using CFT fails using cfn signal option.

There are nine major sections in a CFT. They are:

- Format Version - specifies the CFT version to which the template conforms to
- Description – describes the template
- Metadata - Objects that provide additional information about the template.
- Parameters - specifies the values that can be passed during the runtime
- Mappings – set of keys and values that can be used to specify conditional parameter values
- Conditions – specifies the conditions which control the resource creation and value assignments during stack creation
- Transform – used to specify the AWS Serverless Application Model(SAM) for serverless applications
- Resources – required, rest all are optional – specifies the stack resources and their properties
- Output – describes the values that would return when you view your stack's properties

CFT BEST PRACTICES

The best practices of CFT are:

Planning and organizing

- Organize Your Stacks By Lifecycle and Ownership
- Use Cross-Stack References to Export Shared Resources
- Use IAM to Control Access
- Reuse Templates to Replicate Stacks in Multiple Environments

- Verify Quotas for All Resource Types
- Use Nested Stacks to Reuse Common Template Patterns

Creating templates

- Do Not Embed Credentials in Your Templates
- Use AWS-Specific Parameter Types
- Use Parameter Constraints
- Use AWS::CloudFormation::Init to Deploy Software Applications on Amazon EC2 Instances
- Use the Latest Helper Scripts
- Validate Templates Before Using Them

Managing stacks

- Manage All Stack Resources Through AWS CloudFormation
- Create Change Sets Before Updating Your Stacks
- Use Stack Policies
- Use AWS CloudTrail to Log AWS CloudFormation Calls
- Use Code Reviews and Revision Controls to Manage Your Templates
- Update Your Amazon EC2 Linux Instances Regularly

AWS HELPER SCRIPTS

AWS CloudFormation includes a set of helper scripts:

cfn-init: This script is used to fetch and parse metadata, install packages, write files to disk and start or stop the services by reading template metadata from AWS::CloudFormation::Init Key.

Syntax: cfn-init --stack|-s *stack.name.or.id* \
--resource|-r *logical.resource.id* \
--region *region*
--access-key *access.key* \
--secret-key *secret.key* \
--role *rolename*\
--credential-file|-f *credential.file* \
--configsets|-c *config.sets* \
--url|-u *service.url* \
--http-proxy *HTTP.proxy* \
--https-proxy *HTTPS.proxy* \
--verbose|-v

cfn-signal: This script is used to signal AWS CF to indicate the progress of creation/updation of ec2 instance and or software applications (if at all) when they are ready. WaitOnResource signals are used to hold the work on stack until a predefined number of signals are received or until the timeout period is exceeded.

Syntax: cfn-signal --success|-s *signal.to.send* \
--access-key *access.key* \
--credential-file|-f *credential.file* \
--exit-code|-e *exit.code* \
--http-proxy *HTTP.proxy* \
--https-proxy *HTTPS.proxy* \
--id|-i *unique.id* \
--region *AWS.region* \
--resource *resource.logical.ID* \
--role *IAM.role.name* \
--secret-key *secret.key* \
--stack *stack.name.or.stack.ID* \
--url *AWS CloudFormation.endpoint*

cfn-get-metadata: This script is used to fetch a metadata block from CloudFormation and print it to standard out.

Syntax: cfn-get-metadata --access-key *access.key* \
--secret-key *secret.key* \
--credential-file|f *credential.file* \
--key|k *key* \
--stack|-s *stack.name.or.id* \
--resource|-r *logical.resource.id* \
--url|-u *service.url* \
--region *region*

cfn-hup: This is script is a daemon which detects the changes in the resources metadata and executes user specified actions when a change is detected. Majorly used to make configuration updates.

Syntax: cfn-hup --config|-c config.dir \
--no-daemon \

```
--verbose|-v
```

All these scripts are based on cloud-init. You call these helper scripts from your AWS CloudFormation templates to install, configure, and update applications on Amazon EC2 instances that are in the same template.

AWS VPC

Virtual private cloud which we can simply refer to as a private sub-section of AWS which can be controlled by the user who can place the AWS resources within it and thus creating a logically isolated section. The components of a VPC are: subnets, network ACL, Nat Gateway, VP gateways, internet gateway, route table, elastic IP, endpoint, security group, VPN connections and customer gateways.

When you create an AWS account: a VPC with Internet gateway, route table with predefined routes to the default subnets, NACL with predefined rules and subnets to provision resources in.

Internet Gateway: is the one which routes the connection between a VPC and internet. Only one vpc can be attached to one internet gateway. We cannot detach an IGW when there are active resources in VPC.

Route tables: consists a set of rules called routes which are used to determine where the network traffic is directed. You cannot delete a route tables when you have active dependencies. To route the traffic to AWS resources in VPC.

Network access control lists (NACL): an optional layer of security for VPC that acts as a firewall for traffic control in subnets. These have inbound and outbound rules. In default NACL all traffic is allowed. The rules are evaluated based on the rule number. In NACL we have a catch all rule which denies all the traffic by default and this cannot be modified.

NACL VS SG

While NACL is at subnet level, SG is at instance level and in NACL, we can define both allow and deny rules while in SG you can just define only allow rules and by default all the traffic is denied.

The best practice is to allow to only the traffic that is required for example, for a webserver we should allow only http and https traffic. One subnet can have only one NACL while one NACL can be associated with multiple subnets.

Availability zones in VPC: Any resource should be in a VPC subnet and any subnet should be located in only one availability zones and to ensure high availability and fault tolerant systems, multiple AZs are to be utilized. VPCs can span multiple AZs. AZs are designed to deal with the failure of the applications.

VPC options:

- VPC with public subnet
- VPC with public and private subnet
- VPC with public and private subnet with a private VPN
- VPC with a private subnet only and Hardware VPN access

VPC PEERING

This is primarily used to enable the communication between the servers lying in different VPCs as if there were in the same VPC. It allows the machines to connect using private IP addresses. The phases in VPC peering are:

- Initiating-request
- It might Fail
- Or else it moves to Pending-acceptance
- Later it might get Expired
- Or Rejected
- Or if accepted then it will enter into Provisioning phase
- And it will enter Active
- Deleted

Limitations

- Cannot be created between VPCs those having a matching or overlapping IPV4/IPV6 CIDR blocks
- Cannot be created for VPCs lying in different regions
- Will not support transitive peering
- Cannot have more than one peering connection between same VPCs

AWS ELB

Classic elastic load balancer: distributes incoming application traffic evenly across the available servers in multiple AZs thus ensuring fault tolerance. In the architecture, the ELB is ahead of route tables. Not available for free tier.

In ELB we have both ALB and CLB. And the load balancer should be associated with an VPC. And a proper protocol too needs to be specified based on the traffic that ELB will be dealing with. We should assign the ELB to a SG. We can also configure the health checks of the resources that ELB is serving by defining the ping parameters like protocol, port and path and under details, we should specify the response timeout, interval and unhealthy threshold which define the health of the resources. And finally we add the resources to the ELB.

Concept of cross-zone balancing which ensures that the traffic is evenly distributed across all the servers in all the available zones. When disabled just ensures that the traffic is just balanced between the zones only. It can be either internet facing or internal facing. And can be connected to an auto-scaling group or instances directly.

APPLICATION LOAD BALANCER (ALB) VS CLASSIC LOAD BALANCER (CLB)

- CLB operates at layer 4 of the OSI (Open System Interconnection) model which implies that it routes the traffic based on the ip address and the port number while the ALB operates at layer 7 of the model which means that it operates not just based on the ip address and the port number but also based on the application-level content.

- By default the cross-zone load balancing is enabled in ALB and we need to enable the same in CLB.
- While http and https are supported by both the load balancers, CLB can support TCP and SSL in addition there by enabling the ssl termination at the load balancer level itself.
- ALB support path based routing which enables a listener to forward the requests based on the url path while CLB cannot.
- ALB supports deletion protection which prevents the accidental deletion of the load balancer while CLB doesn't.

AWS AUTOSCALING

Refers to the process of scaling up or scaling down the resources based on the load through the collection of AWS resources groups called as Autoscaling group and by defining the minimum, desired and maximum number of resources for each group and the autoscaling policy. It can span multiple subnets and multiple AZs but will remain within a VPC. It goes hand in hand with ELB. It has two components launch configuration and auto scaling group. Launch configuration defines the ec2 specifications like which AMI, size of the volume etc. Auto scaling group refers to the rules and settings that govern the scaling. Finally it is free service.

In autoscaling group we have minimum, maximum and desired number of instances

We do that using cloudwatch

AWS RDS

Data storage considerations:

- Data format
- Size
- Query frequency
- Write frequency
- Access speed

Types of storage: Unstructured or BLOB and Relational Database.

The AWS RDS is considered as DBaaS.

Six different SQL engines supported by AWS are: Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, SQL server.

Benefits of a RDS:

- Simple and easy to deploy and Cost effective
- Compatible with your applications
- Manages common administrative tasks and is Secure
- Simple and fast to scale.

Backup options: automatic backup - daily backup, configure backup to custom, point-in-time recovery and geo replicated. And Manual snapshots – user initiated, persisted in S3, only recoverable to the point snapshot is taken.

NoSQL vs SQL:

- **Data storage:** Rows and cloumns vs Key-value pairs
- **Schemas:** Fixed vs Dynamic
- **Querying:** using SQL vs focus on collection of documents by querying individual items
- **Scalability:** Vertical vs horizontal

AWS RESOURCE LIMITS

Resource	Default limits
VPCs per region	5
Subnets per region	200
Elastic IPs per region	5
Internet/Virtual Private gateways per region	5
NAT gateways per AZ	5
NACLs per VPC	200
Rules per NACL	20
Route tables per VPC	200
Routes per Route table	50
Security groups per VPC	500
Inbound/outbound rules per SG	50 each
Active VPC peering connections per VPC	50
Outstanding VPC peering connections per VPC	25
Expiry time of a peering connection	1 week/168hr
VPC endpoints per region	20
VPN connections per VPC/region	10/50

CLOUDFRONT

- Content Delivery Network – system of distributed servers which delivers the web content to users based on user's geographic location, origin of webpage and the content delivery server.
- Edge Location – a location where content will be cached and is separate to an AWS region/AZ.
- Origin – simply refers to the location of the files.
- Distribution – refers to the collection of edge locations
- CloudFront – handles the delivery of website (dynamic, static, streaming and interactive content) using global network of edge locations to ensure optimized performance.

AWS DYNAMODB

- At top-level organized into TABLES EX: employee
- TABLES contain ITEMS EX: each employee
- ITEMS contain ATTRIBUTES EX: details of employee

Each table contains a primary key and sort key (optional second key) and up to five local secondary indexes.

Throughput is provisioned at table level and read capacity and write capacity are separate.

Perform queries using partition key and sort key and optionally use local secondary index.

Operations can be performed using management console through restful api or through code using an SDK. Basic operations are available: create, read, update and query.

MIGRATION TO CLOUD

A phased strategy is the best solution while migrating to cloud. The phases in this strategy are:

Cloud Assessment Phase: In this phase, a business case is developed for the decision of moving to cloud. Assessments that will be conducted during this phase include: financial (TCO), security and compliance and technical. Tools that can be reused and are required to be built are identified and licensed products are migrated. It helps to identify the gap between the current traditional legacy architecture and the next-generation cloud architecture. By the end of this phase, we would have identified the applications that can be migrated to cloud in the order of priority, define success criteria and create a roadmap and plan.

Proof of concept:

In this phase, the primary goal is to test whether the decision to migrate to cloud by deploying a small application on to it and testing its suitability through a small proof of concept. This can be achieved by testing the critical functionality of the application. In this stage, you can build support in your organization, validate the technology, test legacy software in the cloud, perform necessary benchmarks and set expectations. By the end of this phase, we will be able to gain far more better visibility into the service provider's offerings and would have had an hands-on experience of them.

Data Migration Phase:

	Amazon S3 + CloudFront	Amazon EC2 Ephemeral Store	Amazon EBS	Amazon SimpleDB	Amazon RDS
Ideal for	Storing large write-once, read-many types of objects, Static Content Distribution	Storing non-persistent transient updates	Off-instance persistent storage for any kind of data,	Query-able light-weight attribute data	Storing and querying structured relational and referential data
Ideal examples	Media files, audio, video, images, Backups, archives, versioning	Config data, scratch files, TempDB	Clusters, boot data, Log or data of commercial RDBMS like Oracle, DB2	Querying, Indexing Mapping, tagging, click-stream logs, metadata, Configuration, catalogs	Web apps, Complex transactional systems, inventory management and order fulfillment systems
Not recommended for	Querying, Searching	Storing database logs or backups, customer data	Static data, Web-facing content, key-value data	Complex joins or transactions, BLOBs Relational, Typed data	Clusters
Not recommended examples	Database, File Systems	Shared drives, Sensitive data	Content Distribution	OLTP, DW cube rollups	Clustered DB, Simple lookups

Table 3: Data Storage Options in AWS cloud

As one size doesn't fit all, by making right tradeoffs on various dimensions like – cost, durability, query-ability, availability, latency, performance (response time), relational (SQL joins), size of object stored (large, small), accessibility, read heavy vs. write heavy, update frequency, cache-ability, consistency (strict, eventual) and transience (short-lived) – a decision is made on type of database used.

	Amazon RDS	RDBMS AMIs	3rd Party Database Service
RDBMS	MySQL	Oracle 11g, Microsoft SQL Server, MySQL, IBM DB2, Sybase, Informix, PostGreSQL	Vertica, AsterData
Support provided by	AWS Premium Support	AWS and Vendor	Vendor
Managed by AWS	Yes	No	No
Pricing Model	Pay-as-you-go	BYOL, Pay-as-you-go	Various
Scalability	Scale compute and storage with a single API call or a click	Manual	Vendor responsibility

Table 4: Relational Database Options

The first step is to point your pipe of the File servers, Log servers, Storage Area Networks and systems to S3 so that new data is stored in that and the old data can be moved using batch process. Many organizations use existing encryption tools like 256-bit AES for data at-rest, 128-bit SSL for data in-transit to encrypt data in Amazon S3. And Migrate your MySQL Databases toz3 Amazon RDS, Migrate your Commercial Databases to Amazon EC2 using Relational DB AMIs and Move Large Amounts of Data using Amazon Import/Export Service
which allows the organizations to load its data on USB 2.0 or eSATA storage and get them shipped to AWS which uploads that into S3.

Application Migration Phase:

There are two main application migration strategies:

1, *Forklift migration strategy*: Stateless applications, tightly coupled applications, or self-contained applications might be better served by using the forklift approach which picks it up all at once and moves to cloud. Self-

contained web applications, backup/archival systems and components of a 3-tier web applications which calls for low-latency are the ones suitable for this strategy. Tasks performed are: copying your application binaries, creating and configuring Amazon Machine Images, setting up security groups and elastic IP addresses, DNS, switching to Amazon RDS databases.

Like with any other migration, having a backup strategy, a rollback strategy and performing end-to-end testing is a must when using this strategy

2. *Hybrid migration strategy*: A low risk strategy which moves the parts of the application to cloud and leaving the rest behind and is optimal for large applications. It calls to design, architect and build temporary "wrappers" to enable communication between parts residing in your traditional datacenter and those that will reside in the cloud.

Leverage the Cloud:

In this phase, the additional benefits of the cloud are explored through leveraging other cloud services which in the case of AWS are: Autoscaling, Automate elasticity, Harden security etc.

Optimization Phase:

This aims at reducing the costs and optimizing the applications by understanding usage patterns, terminating the under-utilized instances, leveraging reserved Ec2 instances, Improving efficiency etc.

BLUE-GREEN DEPLOYMENT

This methodology is adopted to ensure the real-time delivery of the new applications and services to the customers with near zero-downtime release and rollback capabilities. It works on the basic idea of switching the traffic between two identical environments running different versions of an application where BLUE refers to the current version and GREEN refers to the future version intended for deployment. Once the green environment is tested and ready, then the production traffic is redirected to it from blue environment through CANARY ANALYSIS approach where a 20% of traffic is first diverted to GREEN and once it is known that there are no problems in GREEN environment, then the rest of the traffic too is diverted and the BLUE environment is placed to stand-by for a certain period of time and then terminate it. This is mainly to enable the option of roll-back if we find any issues with the green environment.

Benefits of this approach include ability to deploy the current and future versions in parallel and isolated environments, the ability to roll back the application, reduce the blast radius through the adoption of canary testing, minimize the impaired operation/downtime, fits well with the CI/CD workflows through limiting the complexity by eliminating the need for considering too many dependencies on the existing environment as the Green environment uses entirely new resources and finally the cost optimization benefits through the elimination of the need to run an overprovisioned architecture for an extended period of time.

The initial step towards approaching this methodology is to define the environmental boundary to get an understanding of the changes and this is defined by factors like application architecture (dependencies), organizational factors (speed and number of iteration), risk and complexity (blast radius & impact), people, process (QA & rollback) and cost.

The tools/services that enable blue/green deployments are Route 53, ELB, Auto scaling, EBS, Opsworks, Cloudformation, cloud watch etc.

There are multiple techniques for implementing/adopting this blue-green deployment approach like

- Update DNS Routing with Amazon Route 53
- Swap the Auto Scaling Group Behind Elastic Load Balancer
- Update Auto Scaling Group Launch Configurations
- Swap the Environment of an Elastic Beanstalk application
- Clone a Stack in AWS OpsWorks and Update DNS

Best practices and recommendations for this approach are:

- To meet data consistency requirements, both the blue and green environments should share the same data stores.
- Decoupling Schema Changes from Code Changes

The approach that we have followed in my earlier project was to swap the auto scaling groups behind the ELB. In this technique, while Autoscaling is used to manage the EC2 resources for Blue and Green environment, ELB is used to handle the production traffic between them. Whenever a new instance is added to the autoscaling group, it is automatically added to the load balancing pool provided they pass the health checks which can be a simple ping or complex connection requests which would occur at configurable intervals and have defined thresholds.

Through this technique, as usual, while the blue environment represents the current version of the application, new version of the application will be stage in the green environment and when it is time to deploy the code, we simply attach the new autoscaling group in the green environment to the load balancer which uses the least outstanding requests routing algorithm and thereby diverting traffic to the green environment. And the amount of traffic can be controlled by adding or deleting instances to the autoscaling group in the green environment. By scaling up the instances in the green group, we can either terminate or place them in stand-by mode thus enabling rollback option whenever we find issue with the new version.

DOCKER

VM VS DOCKER CONTAINERS

Majority of people link Docker containers with VMs, they are two technologies which differ in the way they operate. They share some similarities like both are designed to provide an isolated environment in which to run

an application. The key difference is that the underlying architecture is different. Virtual machines have a full OS with its own memory management. Every guest OS runs as an individual entity from the host system.

Whereas containers run on the host OS itself and use the ephemeral storage. Containers are therefore smaller than Virtual Machines and enable faster start up with better performance, less isolation and greater compatibility possible due to sharing of the host's kernel. Docker is not a virtualization technology, it's an application delivery technology.

Finally, the benefits of Docker containers include:

- They are lightweight in nature use less memory and
- Lesser time to start a container
- Comparably, applications hosted in containers offer superior performance (almost twice according to Docker estimate).

Future is about integrating the Docker containers with VMs which will provide them with pros like proven isolation, security properties, mobility, dynamic virtual networking, software-defined storage and massive ecosystem.

DOCKERFILE

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

The Docker daemon runs the instructions in the Dockerfile one-by-one, committing the result of each instruction to a new image if necessary, before finally outputting the ID of your new image.

The format of Dockerfile is:

#comment

INSTRUCTION/DIRECTIVES arguments

Although the instruction is not case-sensitive, it is convention to specify them in uppercase to distinguish them from arguments.

The instructions frequently used in a Dockerfile are:

- FROM
- MAINTAINER
- ADD
- CMD
- COPY
- ENTRYPOINT
- ENV
- EXPOSE
- LABEL
- USER
- WORKDIR
- VOLUME
- RUN

Before the Docker CLI sends the context to the Docker daemon, it looks for a file named .dockerignore in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it. This helps to avoid unnecessarily sending large or sensitive files and directories to the daemon and potentially adding them to images using ADD or COPY.

DOCKER COMMANDS

Docker build -t

Docker run -d -it –rm -p –name -P(for all ports) -v -- net --net=bridge –ip --link

Docker ps & -a

Docker images

Docker commit -m -a container ID/ Image ID

Docker rm & rmi

Docker rm/stop $(Docker ps -a -q)

Docker inspect

Docker search

Docker login and logout

Docker pull & push

Docker attach and Docker exec

docker rmi $(docker images --quiet --filter "dangling=true")

Docker system prune

Docker --version – displays docker version

Docker version – will display the details of client and server – version, api version, go version, built no etc.

Docker info – detailed information about Docker engine

Docker network ls – list the available docker networks

Docker network inspect bridgename – info about docker bridge

Docker network create --subnet <10.1.0.0./16> --gateway <10.0.0.1> --ip-range <10.1.4.0/24> --driver < > --label < >bridgename

Docker network rm bridgename – caution never remove a default one u cannot retrieve it

Docker

Command	**Description**
docker attach	Attach to a running container
docker build	Build an image from a Dockerfile
docker checkpoint	Manage checkpoints
docker commit	Create a new image from a container’s changes
docker container	Manage containers
docker cp	Copy files/folders between a container and the local filesystem
docker create	Create a new container
docker deploy	Deploy a new stack or update an existing stack
docker diff	Inspect changes to files or directories on a container’s filesystem
docker events	Get real time events from the server

docker exec	Run a command in a running container
docker export	Export a container's filesystem as a tar archive
docker history	Show the history of an image
docker image	Manage images
docker images	List images
docker import	Import the contents from a tarball to create a filesystem image
docker info	Display system-wide information
docker inspect	Return low-level information on Docker objects
docker kill	Kill one or more running containers
docker load	Load an image from a tar archive or STDIN
docker login	Log in to a Docker registry
docker logout	Log out from a Docker registry
docker logs	Fetch the logs of a container
docker network	Manage networks
docker node	Manage Swarm nodes
docker pause	Pause all processes within one or more containers
docker plugin	Manage plugins
docker port	List port mappings or a specific mapping for the container
docker ps	List containers
docker pull	Pull an image or a repository from a registry
docker push	Push an image or a repository to a registry
docker rename	Rename a container
docker restart	Restart one or more containers
docker rm	Remove one or more containers
docker rmi	Remove one or more images
docker run	Run a command in a new container
docker save	Save one or more images to a tar archive (streamed to STDOUT by default)
docker search	Search the Docker Hub for images
docker secret	Manage Docker secrets
docker service	Manage services
docker stack	Manage Docker stacks
docker start	Start one or more stopped containers
docker stats	Display a live stream of container(s) resource usage statistics

docker stop Stop one or more running containers

docker swarm Manage Swarm

docker system Manage Docker

docker tag Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

docker top Display the running processes of a container

docker unpause Unpause all processes within one or more containers

docker update Update configuration of one or more containers

docker version Show the Docker version information

docker volume Manage volumes

docker wait Block until one or more containers stop, then print their exit codes

DOCKER REGISTRY & DOCKER HUB

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

Docker Hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

DOCKER ADD VS COPY

the major difference is that ADD can do more than COPY:

- ADD allows <src> to be an URL
- If the <src> parameter of ADD is an archive in a recognized compression format, it will be unpacked

DOCKER STOP VS KILL

- docker stop: Stop a running container by sending SIGTERM and then SIGKILL after a grace period
- docker kill: Kill a running container using SIGKILL or a specified signal

DOCKER DEBUGGING

- Docker logs <container_ID>
- Docker stats <container_ID>
- Docker cp <container_ID>:path_to_logs /local/path
- Docker exec -it <container_ID> /bin/bash
- docker commit <container_id> my-broken-container && docker run -it my-broken-container /bin/bash

DOCKERFILE BEST PRACTICES

- Containers should be ephemeral

- Use .dockerignore file
- Avoid installing unnecessary packages
- Each container should have only one concern
- Minimize the number of layers
- Sort multi-line arguments

KUBERNETES

CHALLENGES ADDRESSED BY KUBERNETES

- Service Discovery
- Load Balancing
- Secrets/configuration/storage management
- Health checks
- Auto-[scaling/restart/healing] of containers and nodes
- Zero-downtime deploys

Kubernetes is a system developed by google to manage containerized applications in a clustered environment. It is primarily meant to address the gap between the modern cluster infrastructure and the presumptions of majority of applications about the environments.

The controlling services in a Kubernetes cluster are called the **master, or control plane**, components. These operate as the main management contact points for administrators, and provide many cluster-wide systems for the relatively dumb worker nodes. These services can be installed on a single machine, or distributed across multiple machines.

MASTER OR CONTROL PLANE COMPONENTS

Etcd: Kubernetes uses etcd, which is a distributed key-value store that can be distributed across multiple nodes, to store configuration data that can be used by each of the nodes in the cluster. It can be configured on a single master server or distributed among various machines while ensuring its connectivity to each of kubernetes machines.

API Server: It is the management point of the entire cluster which allows for configuration of Kubernetes workloads and organizational units. Acts as a bridge between various components to maintain cluster health.

Controller Manager Service: It is the one which maintains the state of the cluster which reads the latest information and implements the procedure that fulfills the desired state. This can involve scaling an application up or down, adjusting endpoints, etc.

Scheduler Service: This assigns the workload to the nodes and tracks resource utilization on each host to ensure that they are not overloaded.

NODE SERVER COMPONENTS

Are the one on which actual work is done. They have the following requirements to communicate with the master components and configure networking for containers:

Docker running as a Dedicated subnet

Kubelet service: The main contact point with master components and is a service. It receives commands and work and interacts with etcd to read configuration details of the nodes.

Proxy Service: Used to deal with the individual host level subnetting and make the services available to external parties through forwarding the requests to the correct containers.

Kubernetes Work Units: While containers are the used to deploy applications, the workloads that define each type of work are specific to Kubernetes.

Pods: The basic unit which generally represents one or more containers that should be controlled as a single "application". It models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled. In Pod, Horizontal scaling is generally discouraged on the pod level because there are other units more suited for the task.

Services: A service, when described this way, is a unit that acts as a basic load balancer and ambassador for other containers. A service groups together logical collection of pods that perform the same function to present them as a single entity. Services are an interface to a group of containers so that consumers do not have to worry about anything beyond a single access location.

Replicated Controllers:

A more complex version of a pod is a replicated pod. These are handled by a type of work unit known as a replication controller.

A replication controller is a framework for defining pods that are meant to be horizontally scaled. The work unit is a nested unit. A template is provided, which is basically a complete pod definition. This is wrapped with additional details about the replication work that should be done.

Kubernetes for creating new Projects, Services for load balancing and adding them to Routes to be accessible from outside, Creation of Pods through new application and control the scaling of pods, troubleshooting pods through ssh and logs, writing/modification of Buildconfigs, templates, Imagestreams etc

KUBERNETES IN A NUTSHELL

It is one of the most popular and stable management platforms for Docker containers – it powers Google Containers Engine (GCE) on the Google Cloud platform.

In Kubernetes, a group of one or more containers is called a ***pod***. Containers in a pod are deployed together, and are started, stopped, and replicated as a group. A **pod** could represent e.g. a web server with a database that run together as a microservice including shared network and storage resources. **Replication controllers** manage the

deployment of pods to the cluster nodes and are responsible for creation, scaling and termination of pods. For example, in case of a node shutdown, the replication controller moves the pods to other nodes to ensure the desired number of replicas for this pod is available. Kubernetes **services** provide the connectivity with a load balancing proxy for multiple pods that belong to a service. This way clients don't need to know which node runs a pod for the current service request. Each pod could have multiple **labels.** These labels are used to select resources for operations. For example, a replication controller and services discover pods by label selectors for various operations.

CONFIGURING MASTER AND MINIONS:

- The first pre-requisite is to install ntpd package on master and as well as minions and we need to enable and start this service to ensure that all the servers are time synchronized.
- Name the master and minions accordingly and save that in /etc/hosts file in order to refer to them based on those names rather than public ips.
- Next is to create a repo to pull latest docker package in /etc/yum.repos.d/virt7-docker-common-release and add the content with name and base url of the repo along with gpgcheck=0 and then run yum update to pull packages on to all the servers.
- Note: For our lab, we just need to ensure iptables and firewall.d services are disabled.
- Now we need to install two packages on all the servers docker and kubernetes by **yum install -y –enablerepo=virt7-docker-common-release kubernetes docker**
- Now we need to configure master server. The first step is to edit the config file in /etc/kubernetes/ where we edit the KUBE_MASTER part to bind it to an interface that we can communicate with and we change its value to the master name that we have changed in /etc/hosts file and leave the default port to 8080. And we will add KUBE-ETCD-SERVERS="--etcd-servers=http://master_name:2379"
- Next step is to configure etcd in master by editing config file in the /etc/etcd/ by changing the listen client and advertise client urls to listen to all the servers on port number 2379.
- The third step is to edit the api server file in /etc/kubernetes/ and change the kube api address to bind to all servers and ensure that port on the local server is listening on 8080 and the kubelet port is 10250 which is default and we can edit admission control to restrict addition kubes and kublets entering our environment.
- Finally, we need to enable the services etcd, kube-apiserver, kube-controller-manager and kube-scheduler.
- For configuring minions, the first step is to edit the kubernetes config file by changing the kube master to look out for the name rather than ip and add etcd servers value to interact with the master etcd server.

- Next is to edit the kubelet config file where we change the kubelet address to bind all addresses, enable the kubelet port, set the kubelet hostname to bind to the minion name, map the kubelet api server to interact with the one in master
- Now enable and start the services: kube-proxy, kubelet and docker.
- The kubectl is the cli used to work with k8s:
  - **Kubectl get nodes** – to get registered nodes with master, use -o flag to define the output path and filter it out
  - **Kubectl describe nodes –** details about nodes
  - **Kubectl get pods –** to list out pods on cluster
- **Sample POD definition:**

  apiVersion: 1

  kind: pod

  metadata:

  name: ngnix

  spec:

  containers:

- name: ngnix

  image: ngnix: 1.7.9

  ports:

- containerPort: 80

- Now use **kubectl create -f file_path** to create a pod. We will find a interim container google container which is required for kubernetes to run.
- We need to do port-forward in order for the master to interact with the containers.
- We can add a label (key/value pair) field to the metadata portion of yaml file to define labels which can be used to filter the pod details by providing the -l flag to list certain pods which match our defined key-value pair (label).
- At times we might use an extension api version to support particular tasks like deployments.
- We will definitely have a docker container in the name of google pause which is part of the implementation of pods.
- We will add replicas in the spec portion and mention our desired number and we will add metadata under the spec again to add labels along with spec again, while using replicas.
- We can update a deployment by just using **kubectl apply -f file_path**

- Autoscale command – **kubectl autoscale deployment <deployment_name> --min=n --max=n –cpu-percent=x%**
- To edit an autoscaled deployment - **Kubectl scale –current-replicas=n –replicas=n deployment/<deployment_name>**
- To create a deployment – **kubectl run <some_name> --port=x --replicas=x --labels=a=b**

RUNDECK

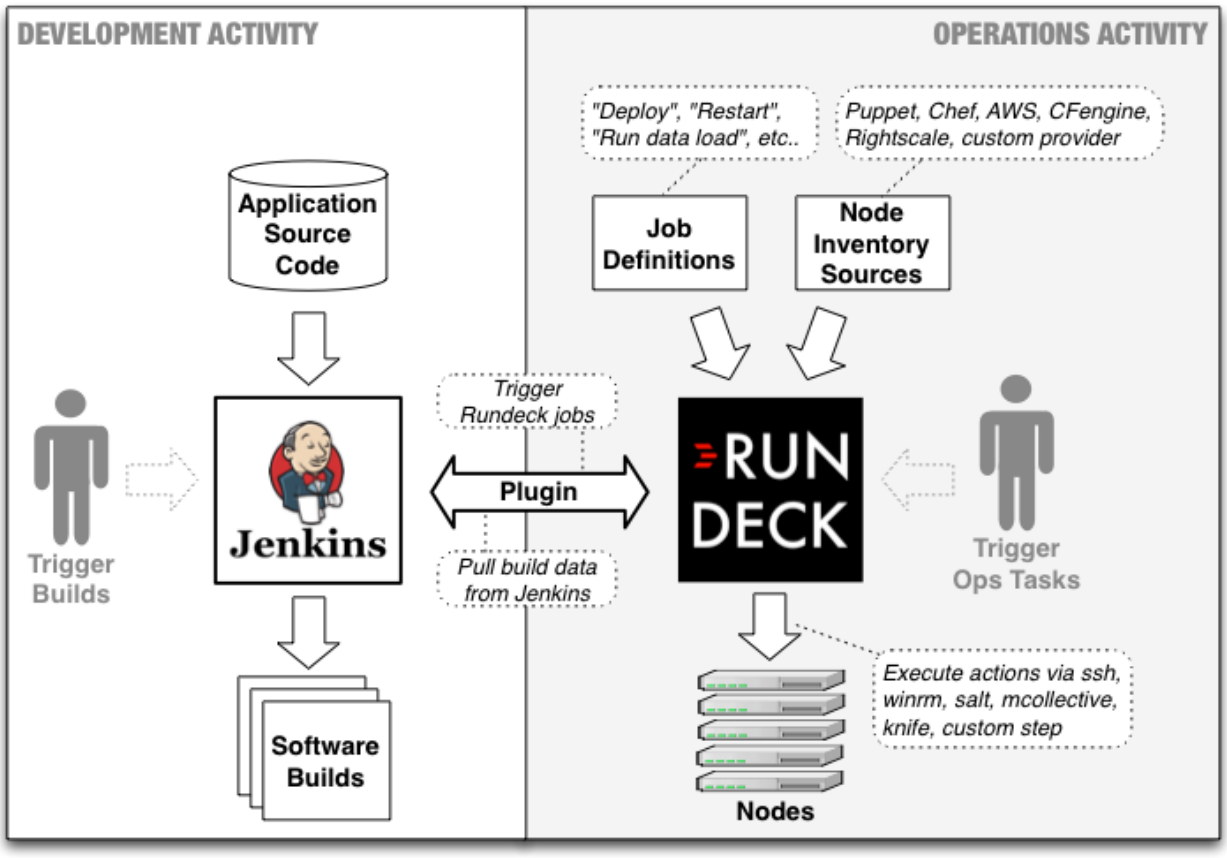
Jenkins and Rundeck are complementary tools that serve two necessary -- but different -- purposes. Jenkins is a development tool, designed for automating software builds. Rundeck is an operations tool, designed for executing operations tasks.

Rundeck is built specifically to turn any operations procedure into a repeatable and secure service that can be accessed securely via a Web GUI or API. Of Rundeck's features that support that operations-centric purpose, the one that stands out as the most unique from Jenkins is Rundeck's inherent knowledge of the detail of your environments (nodes, services, etc.).

Tasks performed by rundeck that any build tool cannot perform are:

- Executes workflows across any remote nodes/servers
- Helps you to better understand the output from remote servers
- Enables the execution of adhoc commands in addition to the saved jobs
- Makes it easy to define and execute multiple types of workflows
- Has built-in error handling features
- Collects logs and sends out notifications
- Provides with access control policy which knows about your environments

If you want to have high quality and throughput throughout your application lifecycle, you need tools that do their specific tasks well and work well together. Jenkins and Rundeck integrate well and are commonly used in the same delivery toolchain that spans from Development to Operations Jenkins will manage software builds and Rundeck will manage any and all operations tasks. Jenkins can trigger Rundeck to do deployments in a Continuous Deployment scenario. The Rundeck Jenkins plugin plays a key role in this.



TERRAFORM

Terraform is an open source tool that allows you to define infrastructure for a variety of cloud providers (e.g. AWS, Azure, Google Cloud, DigitalOcean, etc) using a simple, declarative programming language and to deploy and manage that infrastructure using a few CLI commands.

Why Terraform? To orchestrate multiple services/providers in a single definition like create instances with a cloud provider, create DNS records with a DNS provider, and register key/value entries in Consul. And you can use a single solution that supports multiple services.

Terraform Components:

Terraform allows you to create infrastructure configurations that affect resources across multiple cloud services and cloud platforms with the following components:

- Configurations: It is a text file which holds the infrastructure resource definitions in .tf or .tf.json formats.
- Providers: Terraform leverages multiple providers to talk to back-end platforms and services, like AWS, Azure, DigitalOcean, Docker, or OpenStack.

- Resources: Resources are the basic building blocks of a Terraform configuration. When you define a configuration, you are defining one or more (typically more) resources. These are provider specific and calls to recreate configuration during migrations.
- Variables: Supports use of variables making configurations more portable and more flexible. Re-use a single configuration multiple times through changing variable values.

  We define all the data for terraform in four files:
  - provider.tf.json – contains information specific to aws (the provider)
  - vars.tf.json – contains variables that will be later used by terraform
  - main.tf.json – contains the bulk of terraform configuration (resources)
  - output.tf.json – specifies the information that should be output

After the authoring of these files, the first step is use the terraform plan command. The plan command lets you see what Terraform will do before actually doing it. This is a great way to sanity check your changes before unleashing them onto the world. The output of the plan command is a little like the output of the diff command: resources with a plus sign (+) are going to be created, resources with a minus sign (-) are going to be deleted, and resources with a tilde sign (~) are going to be modified.

Next is to use terraform apply which actually does the work.

NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more. It started out as a web server designed for maximum performance and stability. In addition to its HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP, TCP, and UDP servers.

Database: Oracle 11g (MYSQL)

The types of statements in MYSQL are:

Database Manipulation Language (DML)

DML statements are used to work with data in an existing database. The most common DML statements are:

- SELECT
- INSERT
- UPDATE
- DELETE

Database Definition Language (DDL)

DDL statements are used to structure objects in a database. The most common DDL statements are:

- CREATE
- ALTER

- DROP

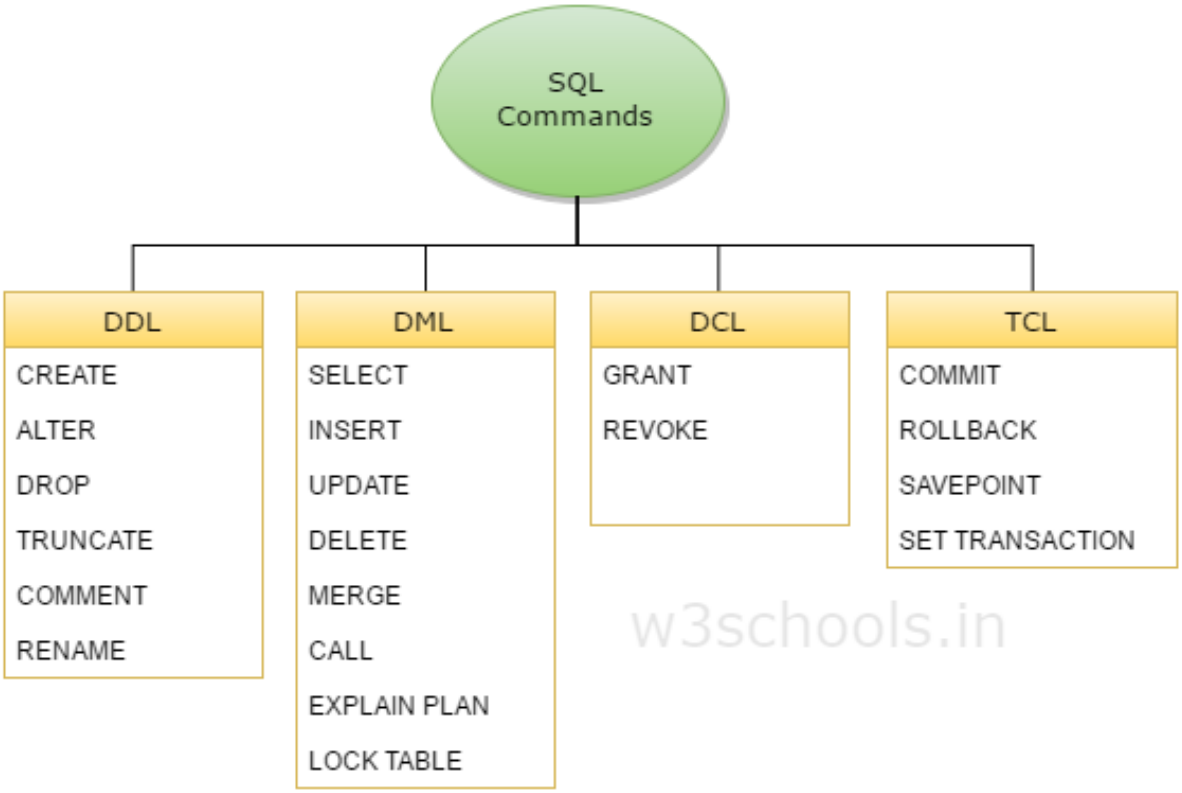
Database Control Language (DCL)

DCL statements are used for database administration. The most common DCL statements are:

- GRANT
- DENY (SQL Server Only)
- REVOKE

We used to handle the DMLs and the DDLs and DCLs are handled by the database admins. I do know a little bit of querying the databases like I will be able to retrieve data using select statements.

Ex: SELECT Column
FROM Table
ORDER BY Column



SHELL SCRIPTING

We can develop various shell scripts for the purpose of automating certain tasks. Shell scripts that are usually developed were meant to serve the tasks like: deployment of the artifacts, log rotation, check disk space and notify through email alerts etc.

- **Check disk space and send email alerts:**

MAX=95

```
EMAIL=USER@domain.com
PART=sda1
USE=`df -h |grep $PART | awk '{ print $5 }' | cut -d'%' -f1`
if [ $USE -gt $MAX ]; then
echo "Percent used: $USE" | mail -s "Running out of disk space" $EMAIL
        fi.
```

- **Checking server utilization:**

```
#!/bin/bash
date;
echo "uptime:"
uptime
echo "Currently connected:"
w
echo "---------------------"
echo "Last logins:"
last -a |head -3
echo "---------------------"
echo "Disk and memory usage:"
df -h | xargs | awk '{print "Free/total disk: " $11 " / " $9}'
free -m | xargs | awk '{print "Free/total memory: " $17 " / " $8 " MB"}'
echo "---------------------"
start_log=`head -1 /var/log/messages |cut -c 1-12`
oom=`grep -ci kill /var/log/messages`
echo -n "OOM errors since $start_log :" $oom
echo ""
echo "---------------------"
echo "Utilization and most expensive processes:"
top -b |head -34
echo
top -b |head -10 |tail -
echo "---------------------"
echo "Open TCP ports:"
nmap -p- -T4 127.0.0.1
```

```
echo "--------------------"
echo "Current connections:"
ss -s
echo "--------------------"
echo "processes:"
ps auxf --width=200
echo "--------------------"
echo "vmstat:"
vmstat 1 5
```

Script to clear the cache from RAM

```
#!/bin/bash
## Bash Script to clear cached memory on (Ubuntu/Debian) Linux
## By Philipp Klaus
## see <http://blog.philippklaus.de/2011/02/clear-cached-memory-on-ubuntu/>
if [ "$(whoami)" != "root" ]
Then
  echo "You have to run this script as Superuser!"
  exit 1
Fi
# Get Memory Information
freemem_before=$(cat /proc/meminfo | grep MemFree | tr -s ' ' | cut -d ' ' -f2) && freemem_before=$(echo "$freemem_before/1024.0" | bc)
cachedmem_before=$(cat /proc/meminfo | grep "^Cached" | tr -s ' ' | cut -d ' ' -f2) && cachedmem_before=$(echo "$cachedmem_before/1024.0" | bc)
# Output Information
echo -e "This script will clear cached memory and free up your ram.\n\nAt the moment you have $cachedmem_before MiB cached and $freemem_before MiB free memory."
# Test sync
if [ "$?" != "0" ]
Then
  echo "Something went wrong, It's impossible to sync the filesystem."
  exit 1
```

```
Fi
# Clear Filesystem Buffer using "sync" and Clear Caches
sync && echo 3 > /proc/sys/vm/drop_caches
freemem_after=$(cat /proc/meminfo | grep MemFree | tr -s ' ' | cut -d ' ' -f2) && freemem_after=$(echo "$freemem_after/1024.0" | bc)
# Output Summary
echo -e "This freed $(echo "$freemem_after - $freemem_before" | bc) MiB, so now you have $freemem_after MiB of free RAM."
exit 0
```

ANSIBLE

Ansible is a simple and powerful IT automation tool whose functions include:

- Provisioning
- Configuration management
- Continuous delivery
- Application deployment
- Security & compliance

All ansible playbooks are written in YAML (yet another markup language or ain't markup language). Ansible playbooks are simple text files. YAML file is used to represent configuration data. Sample of data representation in YAML:

- Key/value pair:- Fruit: Apple
- Array/Lists:- Fruits:
  - Orange
  - Apple
  - Banana

  Vegetable:
  - Carrot
  - Brinjal
  - Potato
- Dictionary/Map:- Banana:

  Calories – 140

  Proteins – 0.4

  Grapes:

Calories – 140

Proteins – 0.4

In YAML, Dictionary is unordered while list is ordered.

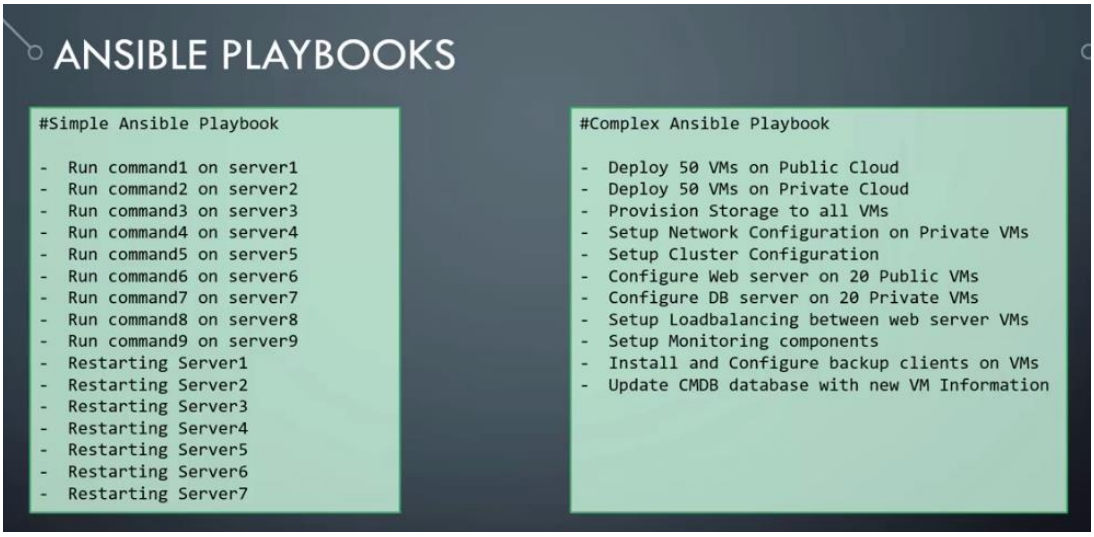
Inventory:

Ansible can work with one to many servers at the same time and uses SSH/Powershell to connect to those servers and thus making it Agentless. The information about these servers which are called target systems in the inventory file. If we don't create an inventory file, then Ansible uses the default file located at etc/ansible/hosts. Inventory file is simple INI file which holds the information like hostname, groups containing hostnames. The inventory parameters of ansible are:

- ansible_host – to use alias for FQDN
- ansible_connection – to indicate the type of connection SSH/winrm/localhost
- ansible_port – defines which port to connect where 22 is default
- ansible_user – to define the user
- ansible_ssh_pass – to define the password (passwords can be encrypted and stored in ansible vault)

PLAYBOOKS

It is a set of instructions that we provide to ansible to do some work on behalf of us.



All playbooks are written in YAML. Play defines the set of activities (tasks) to be run on hosts which include: execute a command, run a script, install a package, shutdown/restart etc. Syntax:

Name: name_of_action

Hosts: name_of_host

Tasks: define the set of tasks here. Using different modules like name, command, script, yum, service etc. Each item is defined by '–' in a separate line. The hosts that we intend to use in playbook should be specified in the inventory file.

The different actions run by tasks are called modules. Ex: command, script, yum, service. Ansible-doc -l will show all the available modules.

To apply the playbook, we simply need to execute the playbook using the command syntax: ansible-playbook <playbook_name>

Modules are categorized into groups like:

- system – actions to be performed at system level: user, group, hostname, iptables, lvg, lvol, make, mount, ping, timezone, systemd, service etc.
- commands – used to execute commands on hosts: command, expect, raw, shell, script etc.
- files – to work with files: acl, archive, copy, file, find, lineinfile, replace, stat template, unarchive etc.
- database – helps in working with databases: mongodb, mssql, mysql, postgresql, proxysql, vertica etc.
- cloud – modules for cloud service providers: module in the name of each cloud provider.
- Windows – to work with windows hosts: win_copy, command, domain, file, iis_website, msg, msi, package, ping, path, robocopy, regedit, shell, service, user etc.

The parameters for command module include: chdir – to change directory, creates – to check the existence and then run the command, executable – change the shell, free_form – to take free form without the need for parameters, removes – to check the availability of the file so that this command executes, warn etc.

The script module executes a local script on one or more remote hosts. It copies that script on to the hosts and then runs the same.

The service module is used to manage the services on the hosts like to start, stop and restart the services on the hosts. We use parameters like name and state to define the service module where name defines the service and the state defines its action like started, stop, restarted. We are stating actions like this because it ensures the service is started if it is not already started and this is referred to as idempotency which implies that the result of the action performing once is exactly the same as performing it repeatedly without intervening actions.

The lineinfile module is used to search for a line in the file and either replace it or add it if it not present.

A variable is used to store the information that varies with each host. Examples of variables in inventory file include ansible_host, ansible_connection and ansible_ssh_pass etc. We will have all the variables declared in a separate file in variables file. To use the variable value, replace the value with variable mentioned in {{variable}}. It is advised to define the variables in either inventory file or a separate host specific file so that the playbooks can be reused.

Conditionals are used to check upon certain conditions. We use either equals ==, or, register and when etc. Register is used to store the outputs of a module.

With_items is a loop iterative which executes the same task multiple number of times.

Ex: Yum: name='{{item}}' state=present

With_items:

- httpd
- glibc
- nodejs

We organize our code into packages, modules, classes and functions. And this is implemented in the form of roles as we have inventory files, variables and playbooks. It is not advisable to mention all the actions in a single large playbook. That's where when include statement and roles come into play. Syntax for include is : - include <playbook_name>. We can even declare variables and tasks in a separate file and use them in the playbook by mentioning vars_files for variables and include for tasks.

Roles define a structure for the project and find standards for organizations folders and files in the project. Roles are used to simplify the functionality of a server by grouping them into webservers, db servers etc. In a way we can easily maintain their configuration by mentioning in the playbook that the host should assume that role. The folder structure of a role is:

- Ansible project
  - Inventory.txt
  - Setup_applications.yml
  - Roles
- Webservers
- Files
- Templates
- Tasks
- Handlers
- Vars
- Defaults
- Meta

The advantages of roles is that it is not required for us to import tasks and variables into the playbook as roles will take care of it.

Ansible control machine can only be linux and not windows but we can connect and control windows machines using winrm connection which is not configured by default.

There is some configuration required to set winrm on windows machine and nothing is required to be installed. Requirements include:

- Have the pywinrm installed on ansible control machine – pip install "pywinrm=>0.2.2"
- Now set up winrm on windows server using the scripts provided by ansible called configure remoting for ansible script which is a powershell script which can be downloaded on windows machine and can be run to setup winrm.
- We can set different modes of authentication like: basic, certificate, Kerberos, NTLM, CredSSP etc.

Ansible Galaxy is a free site for downloading, sharing and writing all kinds of community based ansible roles.

Patterns:

- Host1, Host2, Host3
- Group 1, Host1
- Host*

Dynamic inventory: inventory file that we mentioned earlier is a static and you need to change the information every time, thus, we can use a dynamic inventory script by giving it as an input to the playbook by using '-I' flag.

Syntax: ansible-playbook -i inventory.py playbook.yml

Custom modules: we can develop our own module using python program and place it in modules folder.

REST VS SOAP

What is REST?

REST stands for **RE**presentational **S**tate **T**ransfer. REST is a web standards based architecture and uses HTTP Protocol for data communication. It revolves around resources where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in year 2000.

In REST architecture, a REST Server simply provides access to resources and the REST client accesses and presents the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource like Text, JSON and XML. JSON is now the most popular format being used in Web Services.

API transaction scripts: with details like endpoint, method,

HTTP METHODS

The following HTTP methods are most commonly used in a REST based architecture.

- **GET** − Provides a read only access to a resource.
- **PUT** − Used to create a new resource.
- **DELETE** − Used to remove a resource.
- **POST** − Used to update an existing resource or create a new resource.
- **OPTIONS** − Used to get the supported operations on a resource.

WHEN TO USE SOAP

SOAP is most appropriately used for large **enterprise applications** rather than smaller, more mobile applications. SOAP architecture is most useful when a formal contract must be established to describe the interface that the web service offers, such as details regarding messages, operations, bindings, and the location of the web service. Therefore, SOAP should be used when more capability is needed. For example, including up-to-date stock prices to subscribing websites is a good time to use SOAP since a greater amount of program interaction between client and server is required than REST can provide.

WHEN TO USE REST

REST is implemented most easily using **ASP.NET** web API in MVC 4.0. REST is most appropriately used for smaller, more **mobile applications**, rather than large, enterprise applications. This is because REST is best used as a means of publishing information, components, and processes to make them more accessible to other users and machine processes. An online publisher could use REST to make syndicated content available by periodically preparing and activating a web page that included content and XML statements that described the content.

Overall, if the project requires a high level of security and a large amount of data exchange, then SOAP is the appropriate choice. But if there are resource constraints and you need the code to be written faster, then REST is better. Ultimately it depends on the project and the circumstances which architecture fits best.

GENERAL TOPICS

WHY MULTIPLE JVMS

Admins use multiple JVMs primarily to solve the following issues:

1. Garbage collection inefficiencies
2. Resource utilization
3. 64-bit issues
4. Availability

STATEFUL VS STATELESS SERVICES

Stateless web services do not maintain a session between requests. An example of this would be sites like search engines which just take your request, process it, and spit back some data.

Stateful web services maintain a session during your entire browsing experience. An example of this would be logging into your bank's website or your web based email like GMail.

APPLICATION VERSIONS

Nexus 2.14.3-02

Tomcat 7

Jenkins 2.46.3 LTS

Chef

Ant 1.9.7

Java 1.8.0-121

SVN 1.6.4

Maven 3.3.9

DB server NoSql

VM - VM Player 12.5.6

Centos - 6

Docker - 1.8.0

Kubernetes –

RHEL – 7.1

APACHE WEB SERVER

Troubleshooting Apache Webserver (server not starting/restarting):

1. Check for config syntax error using httpd -t/-s which returns saying SYNTAX OK or SYNTAX ERROR AT LINE SO AND SO.
2. Check Apache error log file `tail -f /var/log/httpd-error.log`
3. Check that your servername is set correctly in httpd.conf
4. Check log files over 2GB because they can cause problem or error 500, so make sure that log files are under limit by moving or removing them out of log directories through log rotation.
5. Check the availability of port 80 and 443

Directory structure of Apache HTTP server:

- Bin - executables
- Conf
- Error
- Icons
- Include
- Lib
- Logs
- modules

TOMCAT

Running multiple web applications in one Tomcat Server:

1. Go to server.xml in conf dir and add a new service tag with details for second application like: port, service name (webapps2), Engine name (webapps2), appbase (webapps2) etc.
2. Next is create a directory in the name webapps2 in order to accommodate the second application.

Benefits of Tomcat over other servers:

- Light weight.
- Widely Used.
- Much faster than other containers.
- Easy to configure.

- Very flexible

Application Server Tuning

Tuning for performance

The next three sections in this article delve into the details of tuning the application server, database server and application to achieve the performance results described above.

Tuning application server

The maximum number of open files descriptor is set at a higher value than the default on Linux/UNIX in order to support parallelism for the CLM/RTC server hosted on an application server. This value is set to:

```
ulimit -n 12000
```

Table 2. Application server tuning parameters

Application Server Configuration	WebSphere Application Server	Tomcat
Thread Pool minimum size	50 is default WebContainer minimum size.	minSpareThreads=25 and maxSpareThreads=72 are default Connector attributes in server.xml.
Thread Pool maximum size	500 is WebContainer maximum size.	500 is maxThreads Connector attribute in server.xml.
Thread/Connection inactivity timeout	60000ms is default.	20000ms is default.
JVM initial memory size -Xms	100M is default for WebSphere Application Server on Linux platform.	4M is default for JVM on Linux platform.
JVM maximum memory size -Xmx	4096M	4096M

Tomcat JDBC resource configuration:

Java Database Connectivity(JDBC) allows Java technologies to connect to a wide variety of database types, over a single protocol, without altering the Java source code and through a JDBC driver which translates java code into database queries.

1. Download and install JDBC driver: there are four drivers:

- JDBC-ODBC (Open database connectivity) bridge (inefficient due to the doubled transformation) ,
- Native-API driver (similar to 1 and uses client side OS specific native API),
- Network-protocol driver (forwards requests to a middleware server, which supports one or several different data formats) and
- Native-protocol driver. The simplest and most efficient of all the driver types, this is a driver written in pure java that performs a conversion of JDBC API calls to the necessary database format and directly connects to the database in question through a socket.

2. Configure your database as a JNDI (Java Naming and Directory Interface) resource:

- The database resource needs to be declared in two places:
- If you wish that database for an application specific, then we declare in the applications META-INF/Context.XML
- if you wish that database to be used by all applications then declare your resource in server.xml and mention the database as a resource reference in META-INF/Context.XML

- to make your application more portable in the application specific scenario, we need to provide resource reference information in WEB-INF/web.xml

Tomcat clustering and loadbalancing with HTTP:

- Download and install mod_jk: mod_jk is the Apache HTTPD module that will be used to provide our cluster with its load balancing and proxy capabilities. It uses the AJP protocol to facilitate fast communication between servers and the Apache Web Server that will receive the client requests. Download, unzip and place in modules directory of http.
- Next step is to configure/set-up mod_jk in httpd.conf file: skip

  # Load module

  LoadModule jk_module path/to/apache2/mod_jk.so

  # Specify path to worker configuration file Tomcat

  JkWorkersFile /path/to/apache2/conf/workers.properties

  # Configure logging and memory

  JkShmFile /path/to/desired/log/location/mod_jk.shm

  JkLogFile /path/to/desired/log/location/mod_jk.log

  JkLogLevel info

  # Configure monitoring

  JkMount /jkmanager/* jkstatus

  <Location /jkmanager>

  Order deny, allow

  Deny from all

  Allow from localhost

  </Location>

  # Configure applications

  JkMount /webapp-directory/* LoadBalancer
- Configure the cluster workers which refers to tomcat servers that process the requests and the virtual servers of the module which handle load balancing Workers.properties

  # Define worker names

  worker.list=jkstatus, LoadBalancer

  # Create virtual workers

  worker.jkstatus.type=status

  worker.loadbalancer.type=lb

Declare Tomcat server workers 1 through n

worker.worker1.type=ajp13

worker.worker1.host=hostname

worker.worker1.port=8009

...

Associate real workers with virtual LoadBalancer worker

worker.LoadBalancer.balance_workers=worker1,worker2,…worker[n]

- Configure tomcat workers – enabling session replication, serializable session attributes, sticky sessions, make your applications distributable, Setting jvm route, keeping your wokers in sync and then configure your clusters.
- Example clustering configuration

<**Engine name**="Catalina" defaultHost="www.mysite.com" jvmRoute="[worker name]">

<**Cluster className**="org.apache.catalina.ha.tcp.SimpleTcpCluster" channelSendOptions="8">

<**Manager className**="org.apache.catalina.ha.session.DeltaManager"

expireSessionsOnShutdown="false"

notifyListenersOnReplication="true"/>

<**Channel className=**"org.apache.catalina.tribes.group.GroupChannel">

<**Membership className**="org.apache.catalina.tribes.membership.McastService"

address="228.0.0.4"

port="45564" frequency="500"

dropTime="3000"/>

<**Sender className**="org.apache.catalina.tribes.transport.ReplicationTransmitter">

<**Transport className**="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>

</Sender>

<**Receiver className**="org.apache.catalina.tribes.transport.nio.NioReceiver"

address="auto" port="4000" autoBind="100"

selectorTimeout="5000" maxThreads="6"/>

<**Interceptor className**="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>

<Interceptor className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interceptor"/>

</Channel>

<**Valve className**="org.apache.catalina.ha.tcp.ReplicationValve" filter=""/>

<Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

<**ClusterListener className**="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener"/>

<ClusterListener className="org.apache.catalina.ha.session.ClusterSessionListener"/>

</Cluster>

SERVICENOW

Is a cloud-based computing model that provides the infrastructure needed to develop, run and manage applications. It is a PaaS. that transforms three areas of IT: consolidation – of all the tasks, consumerization – providing user-friendly experience and automation – process automation : tasks generation, approval and workflow etc.

SONARQUBE QUALITY GATES

Coverage on new code - <80%

Maintainability rating on new code – anything 20-100 is good

Reliability rating on new code – A – 0 bug, B – 1 minor, C – 1 major, D – 1 critical and E – 1 blocker

Security rating on new code - A – 0 vulnerability, B – 1 minor, C – 1 major, D – 1 critical and E – 1 blocker

Blocker	Operational/security ***risk***: This issue might make the whole application unstable in production. Ex: calling garbage collector, not closing a socket, etc.
Critical	Operational/security ***risk***: This issue might lead to an unexpected behavior in production without impacting the integrity of the whole application. Ex: NullPointerException, badly caught exceptions, lack of unit tests, etc.
Info	Unknown or not yet well-defined security risk or impact on productivity.
Major	This issue might have a substantial impact on ***productivity***. Ex: too complex methods, package cycles, etc.
Minor	This issue might have a potential and minor impact on ***productivity***. Ex: naming conventions, Finalizer does nothing but call superclass finalizer, etc.

JAR, WAR & EAR:

In J2EE application, modules are packaged as EAR, JAR and WAR based on their functionality

JAR: EJB modules which contain enterprise java beans (class files) and EJB deployment descriptor are packed as JAR files with .jar extenstion

WAR: Web modules which contain Servlet class files, JSP Files, supporting files, GIF and HTML files are packaged as JAR file with .war (web archive) extension

EAR: All above files (.jar and .war) are packaged as JAR file with .ear (enterprise archive) extension and deployed into Application Server

FORWARD VS REVERSE PROXY

The main purpose of a proxy service (which is the kind of service either of these two provide) is very similar to what a person aims to achieve when he proxies for another person. That is, to act on behalf of that other person. In our case, a proxy server acts on behalf of another machine - either a client or another server.

When people talk about a proxy server (often simply known as a "proxy"), more often than not they are referring to a forward proxy.

A forward proxy provides proxy services to a client or a group of clients. Oftentimes, these clients belong to a common internal network like the one shown below.

A reverse proxy does the exact opposite of what a forward proxy does. While a forward proxy proxies in behalf of clients (or requesting hosts), a reverse proxy proxies in behalf of servers. A reverse proxy accepts requests from external clients on behalf of servers stationed behind it.

Setting up the reverse-proxy:

- Enable proxy_http Apache module enabled: **$ sudo a2enmod proxy_http**
- create a new configuration file in your sites_available folder: **$ sudo vim /etc/apache2/sites-available/my-proxy.conf**
- Add the following to that file:

  <VirtualHost *:80>

  ServerName mydomain.com

  ServerAlias www.mydomain.com

  ErrorLog ${APACHE_LOG_DIR}/proxy-error.log

  CustomLog ${APACHE_LOG_DIR}/proxy-access.log combined

  ProxyRequests Off

  ProxyPass / http://555.867.530.9/

  ProxyPassReverse / http://555.867.530.9/

  </VirtualHost>

- Disable virtualhost for current site:

  ls /etc/apache2/sites-enabled

  > 000-default.conf

  $ sudo a2dissite 000-default

- Enable virtual host for reverse proxy and restart apache

  $ sudo a2ensite my-proxy

$ sudo service apache2 restart

- ***Considerations:***
  - *it applies only to http*
  - *for https, set up reverse proxy on 443 in addition to 80 for http*

ROUTING AND SWITCHING

The way a network operates is to connect computers and peripherals using two pieces of equipment - switches and routers. These two let the devices connected to your network talk with each other as well as talk to other networks.

Switches are used to connect multiple devices on the same network within a building or campus. For example, a switch can connect your computers, printers and servers, creating a network of shared resources.

Routers are used to tie multiple networks together. For example, you would use a router to connect your networked computers to the Internet and thereby share an Internet connection among many users.

RAID

Redundant Array of Independent Disks is traditionally implemented in businesses and organizations where disk fault tolerance and optimized performance are must-haves, not luxuries.

Software RAID means you can setup RAID without need for a dedicated hardware RAID controller.

The one you choose depends on whether you are using RAID for performance or fault tolerance (or both) and type of RAID software or hardware too matters.

RAID 0 is used to boost a server's performance. It's also known as "disk striping." With RAID 0, data is written across multiple disks. Needs minimum two disks and downside is lack of fault tolerance.

RAID 1 is a fault-tolerance configuration known as "disk mirroring." With RAID 1, data is copied seamlessly and simultaneously, from one disk to another, creating a replica, or mirror. Downside is drag on performance and it splits the disk capacity into two equal parts.

RAID 5 is by far the most common RAID configuration for business servers and enterprise NAS devices. This RAID level provides better performance than mirroring as well as fault tolerance. With RAID 5, data and parity (which is additional data used for recovery) are striped across three or more disks. If a disk gets an error or starts to fail, data is recreated from this distributed data and parity block— seamlessly and automatically. Downside is performance hit to those servers which perform more write operations.

RAID 6 is also used frequently in enterprises. It's identical to RAID 5, except it's an even more robust solution because it uses one more parity block than RAID 5.

RAID 10 is a combination of RAID 1 and 0 and is often denoted as RAID 1+0. It combines the mirroring of RAID 1 with the striping of RAID 0. It's the RAID level that gives the best performance, but it is also costly,

requiring twice as many disks as other RAID levels, for a minimum of four. This is the RAID level ideal for highly utilized database servers or any server that's performing many write operations.

WHY BINARY REPOSITORY MANAGER OR ARTIFACTORY?

It functions as a single access point for organizing all binary resources of an organizations. One should adopt a binary repository manager for the following reasons:

- Reliable and consistent access to remote artifacts
- Reduce network traffic and optimize builds
- Full support for Docker
- Full integration with your build eco system
- Security and access control
- License compliance and open source governance
- Distribute and share artifacts, smart search etc.

SSL CREATION

Secured Socket Layer is the standard security which ensures the integrity and privacy of the connection and data flow between a browser and a web server. It actually binds the domain name, server name and company's name together. SSL creation steps using open-ssl are:

1. **Generate a private key**- openssl tool kit is used to generate private key and CSR. This private key is 1024 bit key and is stored in pem format.

- **openssl genrsa -des3 -out server.key 1024**

2. **Generate a CSR**- Generally this CSR is sent to Certificate Authority, who will verify the identity of the requestor and issues a certificate.

- **openssl req -new -key server.key -out server.csr**

3. **Remove passphrase from key**- Important reason for the removal of passphrase is APACHE will ask for the passphrase every time you start the webserver.

- **cp server.key server.key.org**
- **openssl rsa -in server.key.org -out server.key**

4. **Generating a self-signed certificate**- The below command creates a SSL certificate which is temporary and good for 365 days

- **openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt**

5. **Installing the private key and certificate**-

- **cp server.crt /usr/local/apache/conf/ssl.crt**
- **cp server.key /usr/local/apache/conf/ssl.key**

6. **Configuring SSL enabled virtual hosts**
7. **Restart apache and Test**

JIRA INTEGRATION WITH GITHUB

The steps in integrating these two tools are:

- First install DVCS connector in JIRA account. You can do this by getting into administrator settings in toolbar and search for DVCS connector under choose add-ons.
- Next configure JIRA OAuth application in GITHUB. This can be achieved by setting up the JIRA under user account settings → applications → application settings where we register JIRA as new application and we provide the homepage URL of our JIRA account and authorization call back url of our JIRA account. This will provide us with a client ID and a secret. It is recommended to create a new user for this purpose.
- Finally we configure GitHub for JIRA. This can be done by adding GitHub as a new account under the developments tools in any of the project admin's account where we provide the client ID as OAuth key and secret as OAuth secret. This step marks the integration.
- It is advised to use smartcommits in GIT which should link it to an issue/ticket in JIRA

JBOSS

Installing and Configuring JBoss:

JBoss Application Server is the open source implementation of the Java EE suite of services. It comprises a set of offerings for enterprise customers who are looking for preconfigured profiles of JBoss Enterprise Middleware components that have been tested and certified together to provide an integrated experience. It's easy-to-use server architecture and high flexibility makes JBoss the ideal choice for users just starting out with J2EE, as well as senior architects looking for a customizable middleware platform.

Installation:

- Download jboss 7 zip file
- Install java
- Create a jboss user and set password
- Copy the jboss zip file to jboss user home directory and unzip and rename it and change ownership and permissions to 775
- Set the jboss and java path either in .bash_profile or .bashrc
- Update the changes by command #source .bash_profile or .bashrc
- Run the jboss application using the .sh file in jboss bin directory

Table 3.1. The JBoss top-level directory structure

Directory	Description
Bin	All the entry point JARs and start scripts included with the JBoss distribution are located in the bin directory.
client	The JARs that are required for clients that run outside of JBoss are located in the client directory.
server	The JBoss server configuration sets are located under the server directory. The default server configuration set is the server/default set. JBoss ships with minimal, default and all configuration sets. The subdirectories and key configuration files contained default configuration set are discussed in more detail in Chapter 4, *The Default Server Configuration File Set*
Lib	The lib directory contains startup JARs used by JBoss. Do not place your own libraries in this directory.

Table 3.2, "The JBoss server configuration directory structure" shows the directories inside of the server configuration directory and their function.

Table 3.2. The JBoss server configuration directory structure

Directory	Description
conf	The conf directory contains the jboss-service.xml bootstrap descriptor file for a given server configuration. This defines the core services that are fixed for the lifetime of the server.
data	The data directory is available for use by services that want to store content in the file system.
deploy	The deploy directory is the default location the hot deployment service looks to for dynamic deployment content. This may be overridden through theURLDeploymentScanner URLs attribute.
Lib	The lib directory is the default location for static Java libraries that should not be hot deployed. All JARs in this directory are loaded into the shared classpath at startup.
Log	The log directory is the directory log files are written to. This may be overridden through the conf/log4j.xml configuration file.
tmp	The tmp directory is used by JBoss to store temporarily files such as unpacked deployments.

HTTP RESPONSE CODES

- ***1XX – Informational***
  - 100 – continue
  - 101 – switching protocols
- ***2XX – Successful***
  - 200 – OK
  - 201 – created
  - **202 – accepted**
  - 203 – Non – authoritative information
  - 204 – No content
  - 205 – Reset Content
  - 206 – Partial Content
- ***3XX – Redirection***
  - 300 – Multiple choices
  - ***301 – Moved permanently***
  - 302 – Found
  - 303 – See other
  - 304 – Not modified
  - 305 – Use proxy
  - 306 – Unused
  - 307 – Temporary redirect
- ***4XX – Client Error***
  - ***400 – Bad request***
  - 401 – Unauthorized
  - 402 – Payment required
  - ***403 – Forbidden***
  - 404 – Not found
  - 405 – Method not allowed
  - 406 – Not accepted
  - ***407 – Proxy authentication required***
  - ***408 – Request time out***
  - 409 – Conflict
  - 410 – Gone
  - 411 – Length required
  - 412 – Precondition failed
  - 413 – Request entity too large
  - 414 – Request URI too long
  - ***415 – Unsupported media type***
  - 416 – Request range not satisfied
  - 417 – Expectation failed
- ***5XX – Server Error***
  - 501 - Not implemented
  - 502 - Bad gateway
  - ***503 – Service unavailable***
  - 504 – Gateway timeout
  - 505 – HTTP version not supported

REFERENCES

https://linuxacademy.com/

https://docs.chef.io/

https://docs.docker.com/engine/docker-overview/

https://jenkins.io/doc/

https://git-scm.com/doc

https://www.atlassian.com/git/tutorials

http://maven.apache.org/guides/

https://www.udemy.com/learn-ansible/

https://kubernetes.io/docs/home/

https://aws.amazon.com/documentation/

https://github.com/MaximAbramchuck/awesome-interview-questions#shell

https://www.youtube.com/watch?v=RIzcSz66uEI

http://www.tothenew.com/blog/launching-an-aws-ec2-instance-using-cloudformation-template/

http://www.techrepublic.com/article/how-to-free-disk-space-on-linux-systems/

https://www.slideshare.net/chef-software/building-a-private-supermarket-for-your-organization-chefconf-2015

https://d0.awsstatic.com/whitepapers/AWS_Blue_Green_Deployments.pdf

https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes

http://nvie.com/posts/a-successful-git-branching-model/

https://www.cloudbees.com/jenkins/juc-2015/presentations/JUC-2015-USEast-Groovy-With-Jenkins-McCollum.pdf

http://www.bogotobogo.com/DevOps/Jenkins/images/Intro_install/jenkins-the-definitive-guide.pdf

https://www.linux.com/learn/how-backup-files-linux-rsync-command-line

https://docs.chef.io/delivery_pipeline.html

http://josediazgonzalez.com/2013/12/17/deploy-all-the-things-with-bash/

https://www.digitalocean.com/community/tutorials/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu

https://devopscube.com/how-to-install-latest-sonatype-nexus-3-on-linux/

https://docs.chef.io/workflow.html

https://github.com/toddm92/aws/wiki/AWS-CLI-Cheat-Sheet

https://onedrive.live.com/view.aspx?ref=button&Bsrc=Share&Bpub=SDX.SkyDrive&resid=262565195CAD20FF!40982&cid=262565195cad20ff&app=OneNote&authkey=!AgYbT5MBv6Ll7jE

https://onedrive.live.com/view.aspx?ref=button&Bsrc=SMIT&resid=4FA884B1DF7735F5!106&cid=4fa884b1df7735f5&app=OneNote&authkey=AgQLLss011S0EVw

https://renzedevries.wordpress.com/2016/07/18/deploying-kubernetes-to-aws-using-jenkins/

http://richardwestenra.com/automating-git-deployment/

https://kubernetes.io/docs/user-guide/kubectl-cheatsheet/

http://www.agileweboperations.com/amazon-ec2-instances-with-opscode-chef-using-knife

https://www.slideshare.net/initcron/devops-skills-survey

https://blog.gruntwork.io/an-introduction-to-terraform-f17df9c6d180

https://plugins.jenkins.io/

https://blog.osones.com/en/kubernetes-on-aws-with-kube-aws.html

http://blog.scoutapp.com/articles/2014/07/31/slow_server_flow_chart

https://wiki.jenkins-ci.org/display/JENKINS/RunDeck+Plugin#RunDeckPlugin-DeploymentPipeline

http://www.thegeekstuff.com/2016/06/chef-cookbook-directory-structure/

https://github.com/jhotta/chef-fundamentals-ja/blob/master/slides/just-enough-ruby-for-chef/01_slide.md

http://tutorials.jumpstartlab.com/projects/ruby_in_100_minutes.html

http://www.scmgalaxy.com/tutorials/complete-guide-to-use-jenkins-cli-command-line

https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Best+Practices

https://www.c2b2.co.uk/middleware-blog/installing-weblogic-with-chef.php

http://www.ampedupdesigns.com/blog/show?bid=44

https://stelligent.com/2012/09/18/continuous-delivery-in-the-cloud-case-study-for-the-sea-to-shore-alliance-introduction-part-1-of-6/

https://forwardhq.com/help/ssh-tunneling-how-to

https://s3.amazonaws.com/cloudformation-examples/IntegratingAWSCloudFormationWithOpscodeChef.pdf

https://aws.amazon.com/blogs/devops/set-up-a-build-pipeline-with-jenkins-and-amazon-ecs/

https://www.tecmint.com/command-line-tools-to-monitor-linux-performance/

https://devcentral.f5.com/articles/intro-to-load-balancing-for-developers-ndash-the-algorithms

http://www.inf.fu-berlin.de/lehre/SS03/19560-P/Docs/JWSDP/tutorial/doc/GettingStarted10.html

http://www.pcmag.com/article2/0,2817,2370235,00.asp

http://www.guru99.com/jira-tutorial-a-complete-guide-for-beginners.html

https://www.linode.com/docs/applications/configuration-management/creating-your-first-chef-cookbook

https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs

https://github.com/FastRobot/chef_aws_demo/blob/master/cf_templates/Chef-Server-Workstation.template

http://www.bestdevops.com/3-git-hooks-for-continuous-integration/

https://www.atlassian.com/blog/jira-software/connecting-jira-6-2-github

https://www.youtube.com/watch?v=bd-igvtIr7k

https://devops.com/15-must-jenkins-plugins-increase-productivity/

http://www.benchresources.net/difference-between-rest-vs-soap-webservice/

https://jenkins.io/doc/pipeline/examples/

https://kafka.apache.org/intro

https://www.upcloud.com/support/haproxy-load-balancer-centos/

https://jenkins.io/doc/book/architecting-for-scale/

https://www.sumologic.com/aws/elb/aws-elastic-load-balancers-classic-vs-application/

https://www.lucidchart.com/documents/view/703f6119-4838-4bbb-bc7e-be2fb75e89e5/0

https://distinctplace.com/2017/04/20/haproxy-vs-nginx/

https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-14-04

https://www.dynatrace.com/resources/ebooks/javabook/how-garbage-collection-works/

https://stackoverflow.com/questions/2129044/java-heap-terminology-young-old-and-permanent-generations

https://github.com/wsargent/docker-cheat-sheet

https://github.com/pulseenergy/chef-style-guide