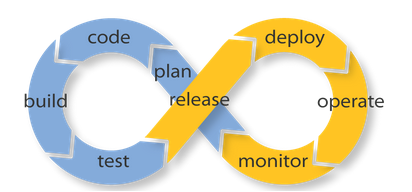
DevOps is all about:

1. culture
2. automation
3. monitoring metrics
4. sharing

DevOps is NOT solving a technical problem, it's trying to solve a business problem and bringing better value to the end user at a more sustainable pace.

This includes practices such as using Continuous Integration and Continuous Delivery, using trunk-based development, and using the cloud well.

 the cloud well means using a container management platform such as Kubernetes, which automates the running and load balancing of containers on any cloud infrastructure. It also means scaling the nodes and containers elastically, all of which can be managed by your cloud provider.



1. **Configuration management.**

**automated infrastructure**, **infrastructure as code**, and **programmable infrastructure**, they're talking about configuration management.

1. ***Application deployment.***

Application deployment tools enable the automation of releases, and are at the heart of [continuous delivery](https://en.wikipedia.org/wiki/Continuous_delivery), one of the primary tenets of DevOps.

For continuous integration and continuous deployment, we need a number of tools to help us there. We need to be able to build reproducible artifacts which we can test. And we need a reproducible infrastructure which we can manage in a fast and sane way. To do that we need a Continuous Integration framework like Jenkins:Other popular tools for automating application deployment include [Ansible](http://www.ansible.com/home" \t "_blank).

***3.Monitoring.***

DevOps requires two distinct types of monitoring. Application performance monitoring tools:

***4.*Version control.**

it's essential to version not just our application code but our infrastructure, configurations, and databases.

***5.Test and build systems.***

These tools automate common developer tasks including compiling source code into binary code, creating executables, running tests, and creating documentation. Tools in this category include

***6.Logstash***

Initially the [logstash](http://logstash.net/" \t "_blank) was just a tool to aggregate, index and search the log files of our platform, it is sometimes a huge missed source of relevant information about how our applications behave. Logstash now allows us to turn boring old logfiles that people only started searching upon failure into valuable information that is being used by product owners and business manager to learn from on the behavior of their users.

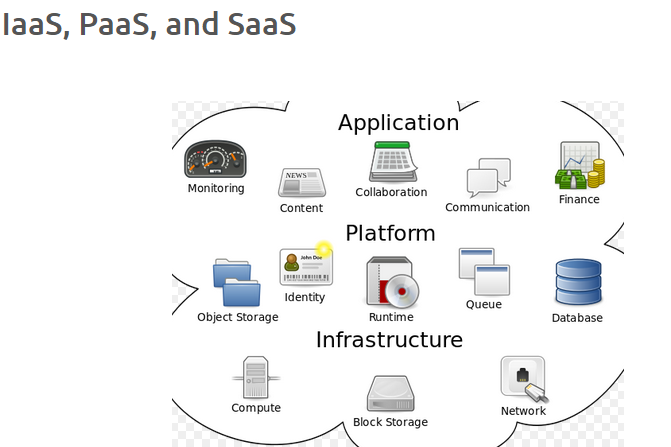
7. ***Elasticsearch***

[Elasticsearch](https://www.bogotobogo.com/Hadoop/ELK/ELK_Elastic_Search_Tutorial.php) is a distributed and scalable **search engine.**  It provides a distributed, multitenant-capable full-text search engine with a RESTful web interface and schema-free JSON documents.

8. [Vagrant](http://www.vagrantup.com/): Create and configure lightweight, reproducible, and portable development environments. It provides a **reproducible** way to generate [fully virtualized machines](https://en.wikipedia.org/wiki/Virtualization) using either Oracle's VirtualBox or VMWare technology as [providers](http://docs.vagrantup.com/v2/providers/index.html). Vagrant can coordinate with a configuration management software to continue the process of installation where the operating system's installer finishes. This is known as [provisioning](http://docs.vagrantup.com/v2/provisioning/index.html).

9. [Docker](http://www.docker.io/): An open source project to pack, ship and run any application as a lightweight container.

10. [OpenStack](http://www.openstack.org/): Open source software for building private and public clouds.



1. **Iaas** : Amazon Web Services (AWS), Microsoft Azure, Google Compute Engine (GCE)
2. **Paas** : AWS Elastic Beanstalk, Cloud Foundry, Google App Engine, Heroku, OpenShift
3. **Saas** : Salesforce, Workday, New Relic, Microsoft Office 365
4. Continuous Delivery => Software can be deployed to customers at any time with the "push of a button" (i.e. by running a deployment script).
5. Continuous Deployment => Software is automatically deployed to customers once it passes through the continuous integration system.

Git

====

**pull request** is a way of offering our changes to the central repo.

To check the status of git, we issue "**git status**".

"**git add .**", we can add all files in the folder and make them be tracked.

when we want to know exactly what we changed not just which files were changed, we can use the **git diff** command

to commit our changes. The simplest way to commit is to type **git commit**:

**git commit --amend** command is a convenient way to fix up the most recent commit.

Using "**git mv**" command ,This time we want to rename 'Book2' to "Introduction".

**UNDOING THINGS : FILE CHECKOUT & UNSTAGING:**

we want to get it back from the repo's previous version. We can do it using **git checkout** command.

Simply unstaging it using **git reset HEAD**

**git log**, we can see our new commit in at the top of the log:

**Soft**

1. The HEAD will be moved without changing the Index (staging area) or Working Directory.
2. **git reset --soft 4ff01e7**

**Mixed (default reset)**

this default reset undoes **git add AND git commit**.

Hard

1. Dangerous, since it overwrites (without checking) any files in the working directory.

we'll integrate the feature branch with a **rebase** to maintain a linear history:

merge

We're now on \*master, and check the README:

resolving the merge conflict – mergetool

git remote add origin account@bogotobogo.com:MyProject.git

The **git push origin master** command says "push the commits in the local branch named master to the remote named origin".

 updates the remote using **push**:

A **fetch** command does not make any changes to local branches, so we will need to merge a remote branch with a paired local branch to incorporate newly fetch changes.

A fork is a copy of a repository.

[fetch](https://www.bogotobogo.com/DevOps/SCM/Git/Git_Terminologies.php#fetch)  
To synchronize our work with remote/origin,

HEAD in Git is the pointer to the current branch reference,

origin is the default name for a remote

git pull is essentially (git fetch + git merge).

Staging all (modified + new + deleted):

git stash - saves changes working directory into "stash".

**Vagrant**

======

We use **Vagrantfile** to configure Vagrant:

**vagrant init** command is a built-in command and it initialize a directory for usage with Vagrant.

Boxes are added to Vagrant with **vagrant box add**.

Boot Vagrant environment - vagrant up

[vagrant halt](https://docs.vagrantup.com/v2/cli/halt.html) command shuts down the running machine Vagrant is managing

**vagrant ssh** command will drop us into a full-fledged SSH session.

**vagrant destroy** back on our host machine, and Vagrant will terminate the use of any resources by the virtual machine.

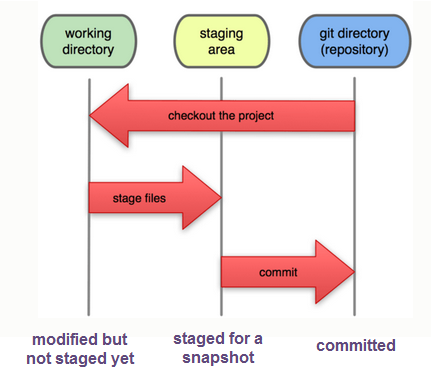
1. **vagrant suspend**: Sleep a virtual machine.
2. **vagrant provision**: Force the provisioners to be run again a virtual machine. Useful for updating the configuration of existing virtual machines.

Using **synced folders**, Vagrant will automatically sync our files to and from the guest machine

Port forwarding allows us to specify ports on the guest machine to share via a port on the host machine

Vagrant share has three main features:

**HTTP sharing, SSH sharing, General sharing**



Jenkins

=========

**Continuous Integration (CI)** is a development practice that requires developers to integrate code into a shared repository frequently.

Each check-in is then verified by an automated build, allowing teams to detect problems early so that we can detect errors quickly, and locate them more easily.

Jenkins is an open source tool to perform continuous integration: monitor a version control system and to start a build system.

Jenkins monitors the whole build process and provides reports and notifications.

1. Log file will be placed in **/var/log/jenkins/jenkins.log**. Check this file if troubleshooting Jenkins. **/etc/default/jenkins** will capture configuration parameters for the launch like e.g **JENKINS\_HOME**.
2. Log file will be placed in **/var/log/jenkins/jenkins.log**. Check this file if troubleshooting Jenkins. **/etc/default/jenkins** will capture configuration parameters for the launch like e.g **JENKINS\_HOME**.

Security setup : Matrix-based security" as the authorization.

'Build Triggers' and check 'Build periodically'

We'll set it to 0/1 \* \* \* \*, starting it every every minute.

The most basic function of any Continuous Integration tool is to **monitor** source code in a version control system and to **fetch and build** the latest version of our source code whenever any changes are committed. So we'll need a version control system. In our case, we'll be using Git. The central source code repository for our simple project is stored on GitHub.

The Jenkins GitHub plugin provides support for receiving push notifications from Github post-commit hooks and using those notifications to trigger jobs within Jenkins. It can setup the post-commit hooks in GitHub for us.

GitHub uses SSH keys to establish a secure connection between our computer and the GitHub servers.

Build jobs are at the heart of the Jenkins build process.

 Jenkins build job as a particular task or step in our build process. This may involve simply compiling our source code and running our unit tests. Or we might want a build job to do other related tasks, such as running our integration tests, measuring code coverage or code quality metrics, generating technical documentation, or even deploying our application to a web server. A real project usually requires many separate but related build jobs.

Once we have told Jenkins where to find the source code for our application, we need to tell it how often it should check for updates. We want Jenkins to monitor the repository and start a build whenever any changes have been committed. This is a common way to set up a build job in a Continuous Integration context, as it provides fast feedback if the build fails.

Other approaches include building on regular intervals (for example, once a day), requiring a user to kick of the build manually, or even triggering a build remotely using a "post-commit" hook in our SCM.

We may want to pick the **Poll SCM** option and enter **H \* \* \* \*** in the Schedule box ("every hour")

build job, we can break down our build job into a number of build steps. This makes it easier to organize builds in clean, separate stages.

 a build might run a suite of functional tests in one step, and then tag the build in a second step if all of the functional tests succeed.

a build step might involve invoking an Ant task or a Maven target, or running a shell script.

This setup will delete any previous build artifacts, compile our code, run our unit tests, and generate a JAR file.

**mvn clean package** will remove the target folder, compile our code and also package it.  if our pom says the project is a jar, it will create a jar for us when we package it and put it in the target directory (by default).

**mvn clean install** will compile and package, but it will also put the package in our local repository.

The standard for test reporting in Java is an XML format used by JUnit.

Post-build Actions section and check "Publish JUnit test result report" checkbox.

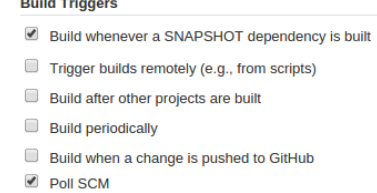
When Maven runs unit tests in a project, it automatically generates the XML test reports in a directory called surefire-reports in the target directory. "\*\*/target/surefire-reports/\*.xml"

Jenkins uses a weather metaphor to help give us an idea of the stability of our builds.

To see how the plugins work, we are going to integrate code coverage metrics using the **Cobertura** plugin (1.9.6). Code coverage is an indication of how much of our application code is actually executed during our tests - it can be a useful tool in particular for finding areas of code that have not been tested by our test suites.

Additional setup such as sys. admin. email, ssh to Jenkins, and Notification Email.

Let's go back to GitHub, under "Settings"=>"Webhooks & Services"=>"Add services"=>"Add Jenkins (Git plugin)".



Then, we're going to run two jobs on Jenkins master and see how the loads are distributed across server/slave nodes.

SSH key : copy master's public key to slave nodes

To install the build pipeline plugin, simply put a tick in the checkbox next to "build pipeline plugin"

To create a pipeline in Jenkins, we need to create the build jobs. Each pipeline section represents one build job. Then, we have to then tell each build job about the downstream build which is must trigger, using the **build other projects** option.

We do not want to build all the jobs when we have a failed task in the flow. Indeed, this Build Flow plugin is doing the right thing. The third job has failed because it returns 1 while executing shell:

On master Jenkins, click "Build Executor Status", and select "New Node". Type in "node-1" for "Node name", and select "Permanent Agent".The "Configure" page looks like this:

**TERRAFORM**

As an open source tool, Terraform provides a flexible abstraction of resources and providers.

strengths: bootstrapping and initializing resources.

Terraform separates the planning phase from the execution phase, by using the concept of an execution plan.

Terraform is an ideal tool for building and managing these infrastructures

The resource block creates a resource of the given TYPE (first parameter) and NAME (second parameter)

terraform init command is used to initialize a working directory containing Terraform configuration files.

"Provider" is its own encapsulated binary distributed separately from Terraform itself.  Provider binary for the providers to use within the configuration, whether the provider is AWS or Google

terraform plan shows what changes Terraform will apply to our infrastructure given the current state of our infrastructure as well as the current contents of our configuration

The plan looks good, our configuration appears valid, so it's time to create real resources. Run terraform apply in the same directory as our **terra-sample0.tf**,

The terraform destroy command should ask us to verify that we really want to destroy the infrastructure.

Provisioning is important for being able to bootstrap instances.

Let's first extract our access key, secret key, and region into a few variables. Create another file **variables.tf**:

We can use output variables as a way to organize data to be easily queried and shown back to the Terraform user.

Maps are a way to create variables that are lookup tables.

count is one of the **meta-parameters** available to all resources. We can specify the number of identical resources to create.

Terraform must store state about our managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.

This state is stored by default in a local file named **terraform.tfstate**

Terraform is able to import existing infrastructure. This allows us take resources we've created by some other means (i.e. via console) and bring it under Terraform management.

terraform import command is used to import existing infrastructure.

Modules are used to create reusable components, improve organization, and to treat pieces of infrastructure as a black box.

The [Terraform Registry](https://registry.terraform.io/) includes a directory

of ready-to-use modules for various common purposes, which can serve as

larger building-blocks for our infrastructur

To install Vault, find the [appropriate package](https://www.vaultproject.io/downloads.html) for the system and download it. Vault is packaged as a zip archive.

Vault operates as a client/server application. The Vault server is the only piece of the Vault architecture that interacts with the data storage and backends.

Vault is the ability to read and write arbitrary secrets securely.

Secrets written to Vault are encrypted and then written to backend storage.

**Ansible 2.0**

**Ansible** distinguishes two types of servers: **controlling machines** and **nodes**.

1. First, there is a **single controlling machine** which is where orchestration begins. The controlling machine describes the location of nodes through its inventory.
2. **Nodes** are managed by a controlling machine over SSH and run the configured Tasks.

Ansible Tasks are **idempotent**.

Modules are temporarily stored in the nodes and communicate with the controlling machine through a JSON protocol over the standard output.

Ansible is **agentless** –

To place the public key at our remote host, instead of logging into that machine, we can use **ssh-copy-id**.

Ansible has a default inventory file (**/etc/ansible/hosts**) used to define which servers it will be managing.

"gather\_facts: False" on the playbook allows implicit fact gathering to be skipped.

Ansible allows us to reference variables in our playbooks using the [Jinja2 templating](http://docs.ansible.com/ansible/playbooks_variables.html) system.

The new playbook does:

1. **hosts**: Find our servers names as a group **mywebservers** in **hosts** inentory file.
2. **vars**: Assign a string for the variable (**MyMessage**) used in our app (index.html).
3. **pre\_tasks**: Added bootstrapping task - installing python 2.
4. **tasks**: Then in the tasks, we setup Nginx with "apt" Ubuntu package tool, and then copy our app using **template** by specifying "src" and "dest".

**Modules** (also referred to as **task plugins** or **library plugins**) are the ones that do the actual work in Ansible, they are what gets executed in each playbook **task**.

**service** module: **command** module: **apt** module.

Each playbook is composed of one or more **plays** in a list.

role is represented by things ansible calls tasks.

 Roles can have multiple task files, and tasks can have multiple modules.

A Handler is exactly the same as a Task, but it will run when called by another Task.

We can add a **notify** directive to the installation Task. This notifies any Handler named "Start Nginx" after the Task is run.

 the **register** and **when** directives. These tell Ansible to run a Task when something else happens.

Role directory structure

A role's directory structure consists of **files**, **handlers**, **meta**, **templates**, **tasks**, and **vars**

Agile

**Iterative** and **incremental** development.

Compressing/Archiving Files - bzip2

rsync

I used **rsync** to copy about 3TB to **NAS (Network Attached Storage)**. Source side has **/data**partition and target side has **/data2** share.

Using **issue** command, we can find what OS is currently running.

Using **cpuinfo** command, we can find what CPU we are using.

The **awk** is most useful when handling text files that are formatted in a predictable way

**awk**'s **search filtering** capabilities:

**sed** is a stream editor, we can use it to replacing a string with another either in-place or to a new file.

The command **cut** is used for text processing. We can use this command to extract portion of text from a file by selecting columns.

We can execute a command periodically using **watch** command.

The **/etc/init.d** directory contains a number of start/stop scripts for various services on our system.

**inode** is a "database" of all file information that tells about file structure.

How to setup passwordless `sudo` on Linux?

We can add a user to a **wheel** group:

The **apt-get update** doesn't actually install new versions of software. It simply does update the Package Index!.

To upgrade packages of our system, first we need to update our package index as outlined above, and then type

The command used for applying patches is the **patch** command but in order to apply a patch

1. **A** is the TYPE. Here, A stands for mapping a domain name to an IPv4 address.

The **ping** command tests the connection between the local machine and a remote address or machine.

The **traceroute** command expands on the functionality of the ping command. It provides a report on the path that the packets take to get from the local machine to the remote machine.

The **netcat** or **nc** is a networking utility for debugging, monitoring, testing, and sending data across network connections.

$ nc google.com 80

We can check **gateway ip** using **netstat** or **route** command:

**netstat** command is used to display network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.

The **nmap** is most commonly used to see which services or ports are open or available on a host.

Runlevel is a mode of operation in OS, and a runlevel represents the different system state of a Linux system.

Firewall decides fate of packets incoming and outgoing in system.

We have three chains:

1. INPUT : Default chain originating to system.
2. OUTPUT : Default chain generating from system.
3. FORWARD : Default chain packets are send through another interface.

The **vmstat** tool provides information about memory, swap utilization, IO wait, and system activity. It is particularly useful for diagnosing I/O-related issues.

vmstat 1 20

To monitor disk read/write rates of individual disks, we can use **iostat**.

The **restricted deletion flag** or **sticky bit** is a permission bit that is set on a directory that allows only the owner of the file within that directory or the root user to delete or rename the file.

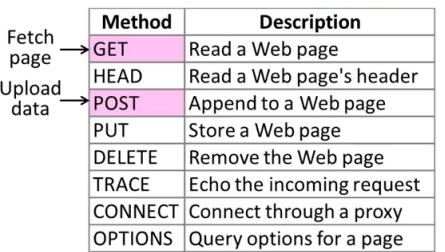
The most frequent use of **lsof** command is when a disk cannot be unmounted as it says the files are being used.

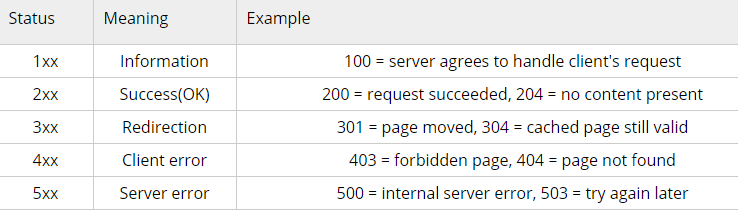
**dmidecode** command reads the system DMI table to display hardware and BIOS information of a linux box:

To change the time zone, we must find the proper zone file in **/usr/share/zoneinfo/** and link that file to **/etc/localtime**:

The **mtr** command, like the **traceroute** tool, provides information about the route that Internet traffic takes between the local system and a remote host.

System - Monitor Processes, Memory, and CPU Usage (htop)





Telnet - check if a web server is running

Telnet can be used for troubleshooting connection issues. It attempts to connect to a server on a given port. Generally we can quickly check if a remote server is "listening" on http port.

$ telnet example.com 80

Web server trouble shooting

1. **Log files**:  
   Log files are in **/var/log** in a sub-folder specific to the service.
2. **Is server running?**:
3. $ sudo netstat -plnt | grep apache2
4. **Is port opened?**:  
   Web servers run on port 80/443, and these ports must be accessible. We can use **netcat/nc** command:
5. $ netcat -z 54.183.114.189 80
6. **Configuration files**:  
   If our web server won't start, we need to look at configuration files in **/etc/apache2** or **/etc/httpd**. To check the configuration syntax:
7. $ apache2ctl configtest
8. Syntax OK

Backup and Restore MySQL Database Using mysqldump

1. uptime - load average

We can get the load average from commands like **top** or **uptime**.

ps - process status listing

To monitor disk read/write rates of individual disks, we can use **iostat**.

sar - system activity report

**sar -P ALL 1 2** displays real time CPU usage for ALL cores every 1 second for 2 times (broken down by all cores).

**sar -P 1 1 2** displays real time CPU usage for core number 1, every 1 second for 2 times.

**sar -n DEV 1 1** displays network devices vital statistics for eth0, eth1, etc. every 1 second for 1 times.

IS THE WEBSITE DOWN?

Is the Server Running?

Is the Remote Port Open? - Nmap (Network Mapper)

Test for Listening Ports

HTTP Status Codes

Command line response test

IS MY SERVER DOWN?

Is my link up (Am I connected)?

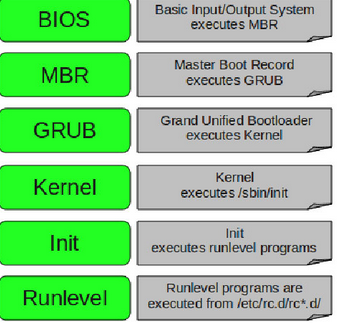
Can we access default gateway? (Is it on the local network?)

How about DNS?

## WHY IS THE SERVER SLOW?

System Load

High Load?

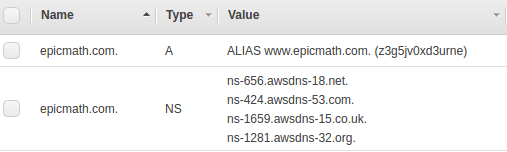


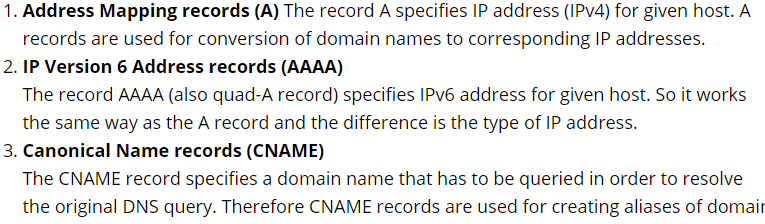
DNS: the mechanics of DNS resolution, including name servers, zone files, and individual DNS records.

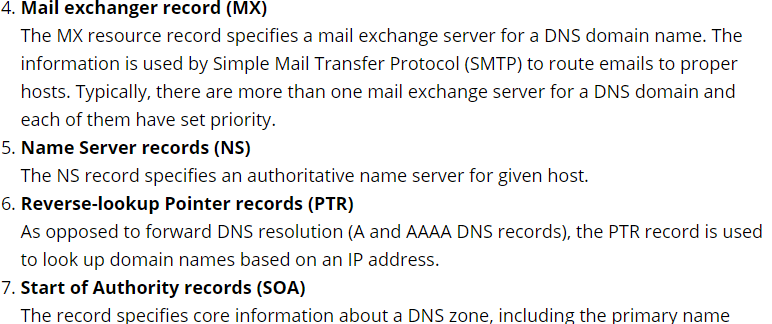
**Name servers** host a domain's DNS information in a text file called the **zone file**. They're are also known as **Servers of Authority (SOAs)**

The following shows names servers (NS) on Godaddy registrar for [xophist.com](http://www.xophist.com/) which is actually hosted by AWS:

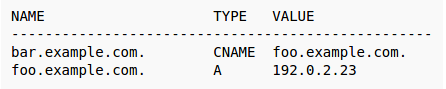
**DNS records** match domain names to IP addresses. The DNS records are then automatically bundled up into a **zone file**, which is what allows the Internet to look up the correct IP address for our domain.







The **A** and **CNAME** records are the two common ways to **map** a **host name** (name hereafter) to an **IP** address.



The **A** record points a name to a specific IP.

A **CNAME(Canonical Name)** record always point to another domain name, never directly to an IP-address.



How SSL works

Secure Socket Layer (SSL) also known as Transport Layer Security (TLS).

The following protocols are most commonly used:

1. SSL 1.0 (not published)
2. SSL 2.0 (1995)
3. SSL 3.0 (1996), broken by POODLE (Nov 2014)
4. TLS 1.0 (SSL 3.1) (1999), minor tweak to SSLv3, weakened by BEAST(2011) and Lucky 13(2013)
5. TLS 1.1 (SSL 3.2) (2006), minor tweak, weakened by Lucky 13 (2013) and RC4(2013, 2015)
6. **TLS 1.2** (SSL 3.3) (2008), improved hashes and AEAD mode, only safe with AEAD mode ciphers
7. TLS 1.3 (n/a) (draft)

Virtual hosts can be "IP-based", meaning that we have a different IP address for every web site, or "name-based", meaning that we have multiple names running on each IP address."

In Nginx web server, **server blocks** can be used for configuration to host more than one domain off of a single server.

Ubuntu's Nginx package does not use **/var/www** as its document root by default. Instead it's using **/usr/share/nginx/html** as Doucment Root.

Once enabled, Versioning cannot be disabled, only suspended.

CloudFront is a service launched by Amazon, powered by cloud computing technology. It is a content delivery web service. It integrates with other Amazon Web Services products to give developers and businesses an easy way to distribute content to end users with low latency, high data transfer speeds, and no minimum usage commitments.

NTP is a protocol designed to synchronize the clocks of computers over a network.

Autoscaling scales up and down a group of servers based on computing or traffic demand by provisioning new services.

1. Autoscaling is done using the following methodologies (ref: [Autoscaling Best Practices](http://www.slideshare.net/lynxmanuk/autoscaling-best-practices" \t "_blank)):
   1. Time based
   2. Reactive
   3. Predictive
2. CloudWatch metrics drives scaling.

**Master-slave** data replication allows us to easily maintain multiple copies of a MySQL data by having them copied automatically from a master to a slave database. We can backup the data, analyze it without using the main database, or scale out our db.

1. The master server records all data changes to its binary logs (binary log events) and send it to the slave using a thread called **Binlog dump thread** once the slave connects to the master.
2. The slave copies the **binary log events** sent by the master's binlog dump thread to its relay logs using a thread called **Slave I/O thread**.
3. The slave applies these changes from the **relay logs** to its data by replaying (executing) all events using a thread called **Slave SQL thread**.

Amazon EC2 Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster of Amazon EC2 instances.

Amazon ECS uses Docker images in task definitions to launch containers on EC2 instances in our clusters.

Docker is a technology that allows us to build, run, test, and deploy distributed applications that are based on Linux containers.

ECS is basically a set of APIs that turn EC2 instances into compute cluster for container management:

1. EC2 instances must call [RegisterContainerInstance](http://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RegisterContainerInstance.html" \t "_blank) API to signal that they are ready to run containers.
2. Need to call [RegisterTaskDefinition API](http://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RegisterTaskDefinition.html" \t "_blank) to define the tasks (setting an image, command and memory for docker run etc.)
3. We use [RunTask](http://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_RunTask.html" \t "_blank) API to start a new task.
4. Lastly, we make a [CreateService](http://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_CreateService.html" \t "_blank) API call to run a long-running contain

We can start using Amazon EC2 Container Service (Amazon ECS) by creating a task definition, scheduling tasks, and configuring a cluster in the Amazon ECS console.

ECS first run wizard configuration

* 1. Configure Repository
  2. Build, tag, and push Docker image
  3. Create a task definition,A task definition is like a blue print for an application

1. Configure service

A service launches and maintains a specified number of copies of the task definition in our cluster.

1. Configure cluster
2. Run

Now, our docker is running on this agent node:

**Amazon Route 53 (Route 53)** is part of Amazon.com's cloud computing platform, Amazon Web Services (AWS).

AWS Elastic Beanstalk is a **PaaS (Platform as a Service)**.

Provisioning, load balancing, autoscaling, and application health monitoring are all automatically handled. By using this AWS PaaS, we can quickly deploy and manage applications in the AWS cloud without worrying about the infrastructure that runs those applications.

Let's check what processes are currently running:top

Getting out of a container without stopping it

"Ctrl + P + Q" will do the trick:

Running a container in a background

To run a container in background, we use "docker start" command:

The **docker attach** command allows us to attach to a running container using the container's ID or name, either to view its ongoing output or to control it interactively.

we could use the Docker **link** feature to allow containers to discover each other.

[Docker Compose](https://docs.docker.com/compose/overview/) is a orchestration tool for multi-container Docker applications. Docker Compose we can orchestrate processes of Docker containers such as starting up, shutting down, and setting up intra-container linking and volumes.

Instead of using the legacy **link** for communications between containers, we can use **networks**