

PRACTICAL DEVOPS TOOLS :

Preparing for the LPI DevOps Tools Engineer
Certification

v1.2

FONGAN TOUSSIDO Gilbert

2022

HELLO!



I am **FONGAN TOUSSIDO Gilbert**,

Cloud DevOps Engineer with five (05) years of experience across AWS and Azure Cloud platforms.

Active member of the **LPI (Linux Professional Institute)**.



SUMMARY

► **PART I: SOFTWARE ENGINEERING**

- Module I-1 : Modern Software Development
- Module I-2 : Standard Components and Platforms for Software
- Module I-3 : Source Code Management
- Module I-4 : Continuous Integration and Continuous Delivery

► **PART II: MACHINE DEPLOYMENT**

- Module II-1 : Virtual Machine Deployment
- Module II-2 : Cloud Deployment
- Module II-3 : System Image Creation

► **PART III: CONTAINER MANAGEMENT**

- Module III-1 : Container Usage
- Module III-2 : Container Deployment and Orchestration
- Module III-3 : Container Infrastructure

► **PART IV: CONFIGURATION MANAGEMENT**

- Module IV-1 : Ansible
- Module IV-2 : Other Configuration Management Tools

► **PART V: SERVICE OPERATIONS**

- Module V-1 : IT Operations and Monitoring
- Module V-2 : Log Management and Analysis

GETTING STARTED

- ▶ Always focus on the objective exams as they describe what will or will not be covered in the exam.
- ▶ Please take notes of the key concepts and commands related to the various topics covered, as you will often be asked to provide commands and sample configurations on the exam.
- ▶ Prepare the environment of its physical machine to be virtualized to carry out various manipulations on the technologies approached through a solution such as Virtualbox.
- ▶ Use the Gitlab repository to make practical examples of all the open-source DevOps tools in this exam.



Instructions for use

Target : This docs is intended for

Students who are interested to start their career in DevOps by preparing the **LPI DevOps Tools Engineer Certification**.
Junior Engineers who work as System Administrators, Software Engineers, DevOps and Cloud Engineer who need to improve their skills in a specific area.

Summary & Pre-requisites

It is presented a technological development environment to initiate the reader to the use of DevOps tools to enable the efficient management of infrastructure through practical cases and thus will help in the preparation of the certification,

It is necessary to have basic knowledge of Linux.

More info and updates

<https://learning.lpi.org/en/learning-materials/all-materials/#devops-version-10>

<https://gitlab.com/GilbertFongan/devops-book-labs>

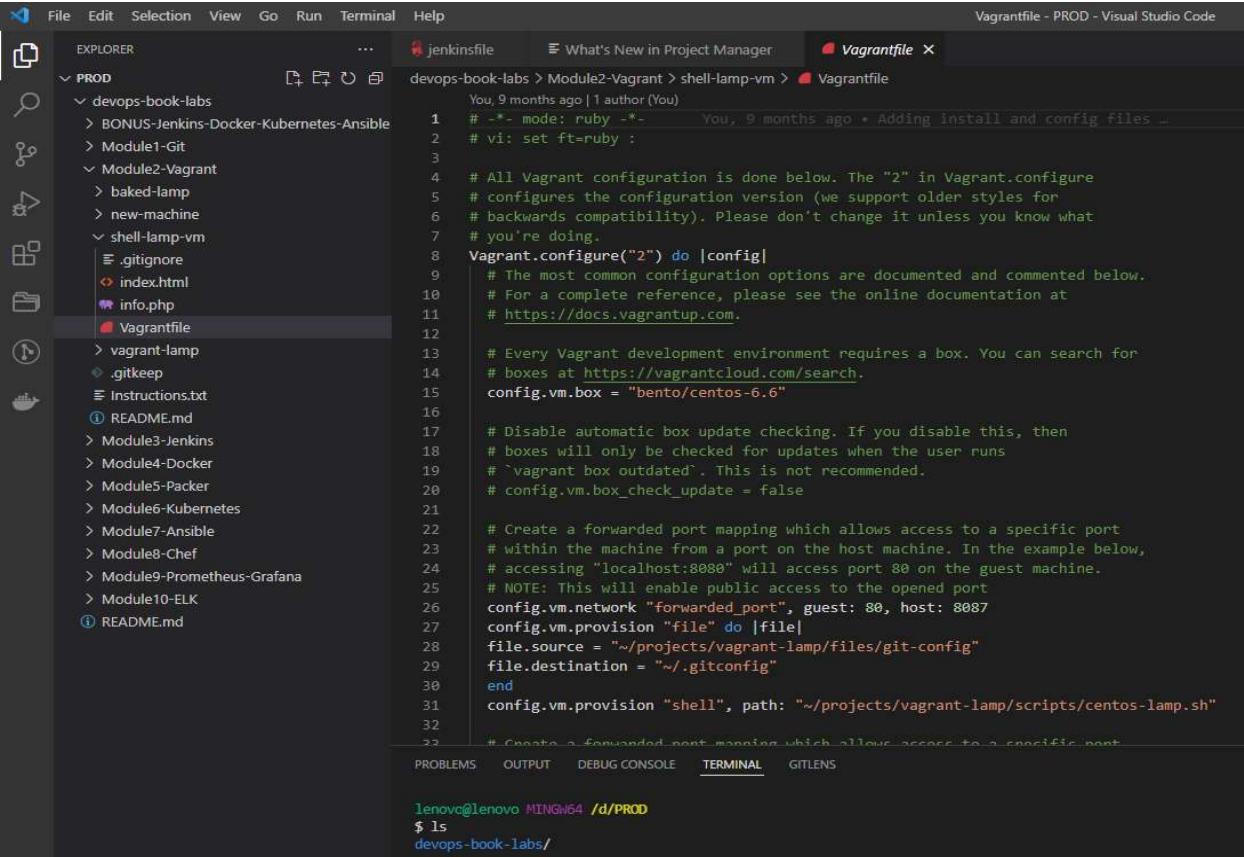
Instructions for use

Clone the Gitlab repository of the book in order to have all the codes in your local environment

The screenshot shows a GitLab project page for 'devops-book-labs'. The left sidebar contains navigation links: Project information, Repository, Issues (8), Merge requests (5), CI/CD, Deployments, Packages and registries, Monitor, Analytics, Wiki, and Snippets. The main content area displays project details: Project ID: 30554095, 143 Commits, 11 Branches, 0 Tags, and 27.9 MB Project Storage. A recent commit by Gilbert Fongan is shown: 'feat(Module3) : Update Jenkins and Tomcat scripts java version and upgrade CPU' (dbf339d8). Below this is a file list with 'main' selected: README, README.md, and several module folders (Module1-Git, Module2-Vagrant, etc.). A 'Clone' button is visible at the bottom right.

Instructions for use

Clone the Gitlab repository of the book in order to have all the codes in your local environment



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a project structure under "PROD".
 - "devops-book-labs" folder contains:
 - BONUS-Jenkins-Docker-Kubernetes-Ansible
 - Module1-Git
 - Module2-Vagrant
 - baked-lamp
 - new-machine
 - shell-lamp-vm
 - .gitignore
 - index.html
 - info.php
 - Vagrantfile** (selected)
 - vagrant-lamp
 - .gitkeep
 - Instructions.txt
 - README.md
 - Module3-Jenkins
 - Module4-Docker
 - Module5-Packer
 - Module6-Kubernetes
 - Module7-Ansible
 - Module8-Chef
 - Module9-Prometheus-Grafana
 - Module10-ELK
 - README.md- Code Editor (Right):** Displays the content of the selected "Vagrantfile".

```
jenkinsfile  What's New in Project Manager  Vagrantfile X
devops-book-labs > Module2-Vagrant > shell-lamp-vm > Vagrantfile
You, 9 months ago | 1 author (You)
1 # -*- mode: ruby -*- You, 9 months ago * Adding install and config files ...
2 # vi: set ft=ruby :
3
4 # All Vagrant configuration is done below. The "2" in Vagrant.configure
5 # configures the configuration version (we support older styles for
6 # backwards compatibility). Please don't change it unless you know what
7 # you're doing.
8 Vagrant.configure("2") do |config|
9   # The most common configuration options are documented and commented below.
10  # For a complete reference, please see the online documentation at
11  # https://docs.vagrantup.com.
12
13  # Every Vagrant development environment requires a box. You can search for
14  # boxes at https://vagrantcloud.com/search.
15  config.vm.box = "bento/centos-6.6"
16
17  # Disable automatic box update checking. If you disable this, then
18  # boxes will only be checked for updates when the user runs
19  # `vagrant box outdated`. This is not recommended.
20  # config.vm.box_check_update = false
21
22  # Create a forwarded port mapping which allows access to a specific port
23  # within the machine from a port on the host machine. In the example below,
24  # accessing "localhost:8080" will access port 80 on the guest machine.
25  # NOTE: This will enable public access to the opened port
26  config.vm.network "forwarded_port", guest: 80, host: 8087
27  config.vm.provision "file" do |file|
28    file.source = "~/projects/vagrant-lamp/files/git-config"
29    file.destination = "~/.gitconfig"
30  end
31  config.vm.provision "shell", path: "~/projects/vagrant-lamp/scripts/centos-lamp.sh"
32
33  # Create a forwarded port mapping which allows access to a specific port
34
```
- Terminal (Bottom):** Shows a terminal session with the command "ls" and the output "devops-book-labs/".

PART I.

Software Engineering

...

MODULE I-1 : Modern Software Development

PLAN

► **SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) :**

- Waterfall
- Iterative
- V-model
- Agile model
- DevOps

► **API CONCEPTS AND STANDARDS :**

- REST API,
- CORS Headers,
- CSRF Token

► **MODERN SOFTWARE ARCHITECTURE :**

- Monolithic
- Service Oriented Architecture
- Microservice
- Serverless



SDLC

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

The SDLC methodology focuses on the following phases of software development :

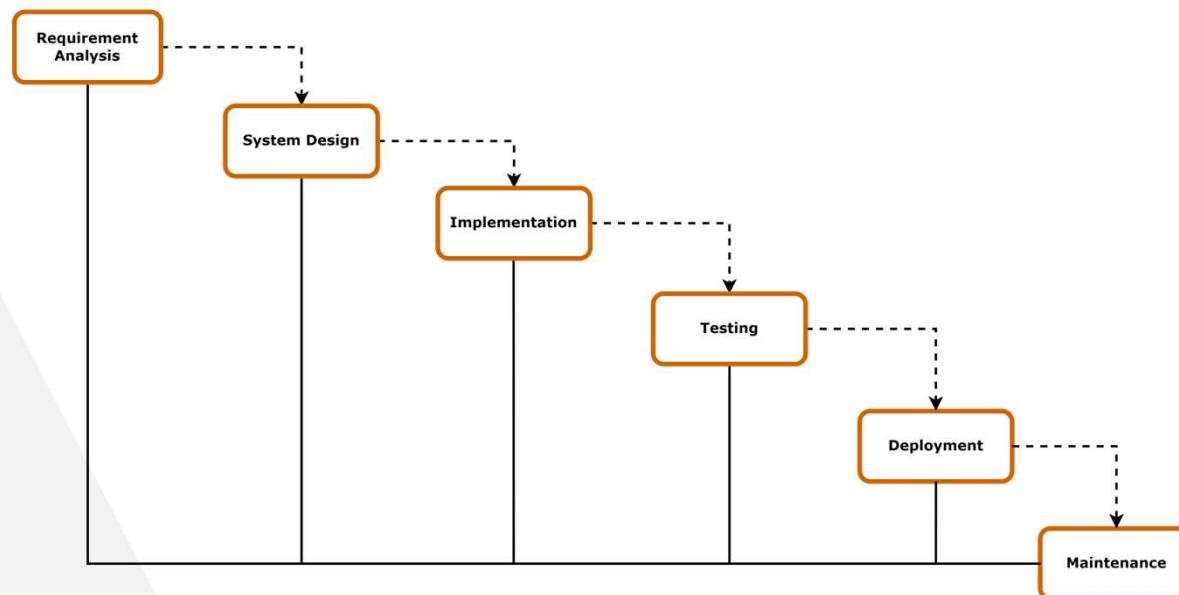
- ▶ Requirement analysis
- ▶ Planning
- ▶ Software design
- ▶ Software development
- ▶ Testing
- ▶ Deployment

Following are the most important and popular SDLC models:

- ▶ Waterfall model
- ▶ Iterative model
- ▶ V model
- ▶ Agile model

SDLC/ Waterfall model

- ▶ First process model to be used widely in Software Engineering.
- ▶ Linear sequential flow.
- ▶ Phase in the **development process** begins only if the **previous phase is complete**. The outcome of one phase acts as the input for the next phase sequentially.



SDLC/ Waterfall model

The sequential phases in Waterfall model are :

- ▶ **Requirement Gathering and analysis** : Requirements of the system to be develop are captured and documented in a requirement specification document
- ▶ **System Design** : Helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- ▶ **Implementation** : The system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality.
- ▶ **Integration and testing** : All units developed in the previous phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- ▶ **Deployment of system** : Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.
- ▶ **Maintenance** : Patches are released to fix some issues which come up in the client environment. Maintenance is done to deliver these changes in the customer environment.

SDLC/ Waterfall model

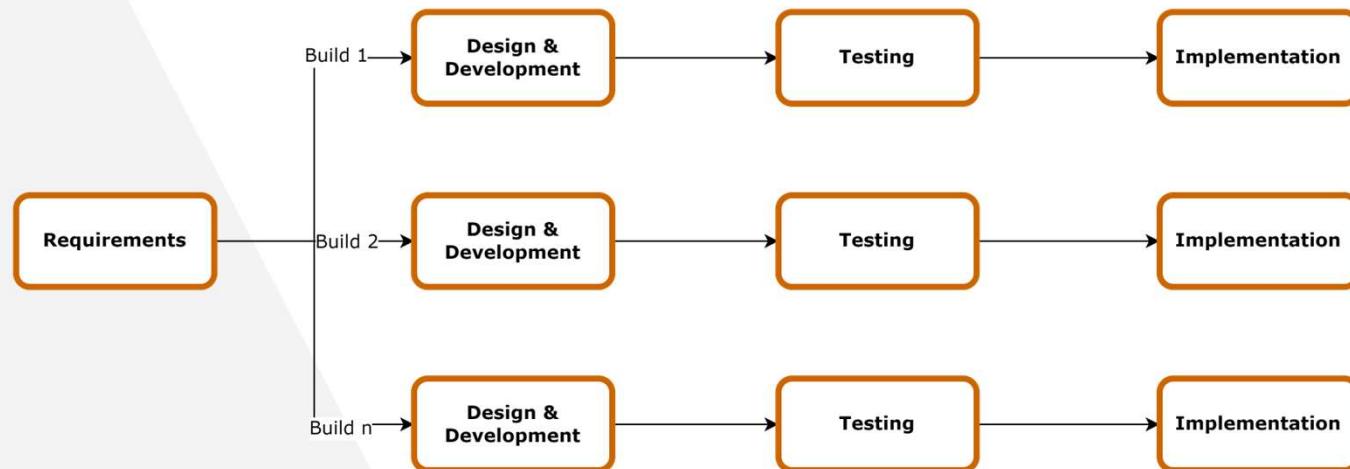
Advantage and disadvantages are :

Advantages	Disadvantages
Simple and easy to understand. Works well for smaller projects where requirements are very well understood	High amounts of risk and uncertainty
Highly-disciplined model and Phases are completed one at a time	Not a good model for complex and object-oriented projects
Clearly defined stages. Process and results are well documented	Difficulty to go back and change the functionality in testing stage

SDLC/ Iterative model

The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time

- ▶ Starts with a simple implementation of a subset of the software requirements
- ▶ Iteratively enhances the evolving versions until the full system is implemented
- ▶ Design modifications are made, and new functional capabilities are added (at each iteration)



SDLC/ Iterative model

Advantage and disadvantages are :

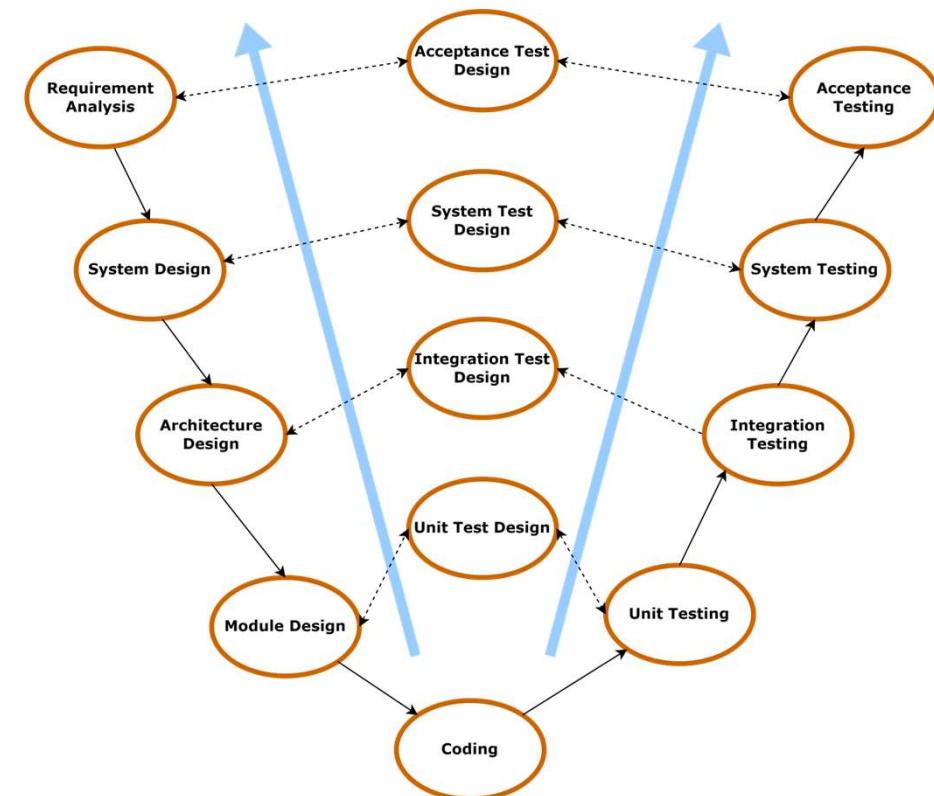
Advantages	Disadvantages
Simple and easy to understand. Works well for smaller projects where requirements are very well understood	High amounts of risk and uncertainty
Highly-disciplined model and Phases are completed one at a time	Not a good model for complex and object-oriented projects
Clearly defined stages. Process and results are well documented	Difficulty to go back and change the functionality in testing stage

SDLC/ V-model

It is also known as Verification and Validation model

- ▶ execution of processes happens in a sequential manner in a V-shape
- ▶ extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage

The following illustration is a representation of the different phases :



SDLC/ V-model

Advantage and disadvantages are :

Advantages	Disadvantages
highly-disciplined model and Phases are completed one at a time	High risk and uncertainty.
Simple and easy to understand and use.	Not a good model for complex and object-oriented projects
Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.	Poor model for long and ongoing projects.

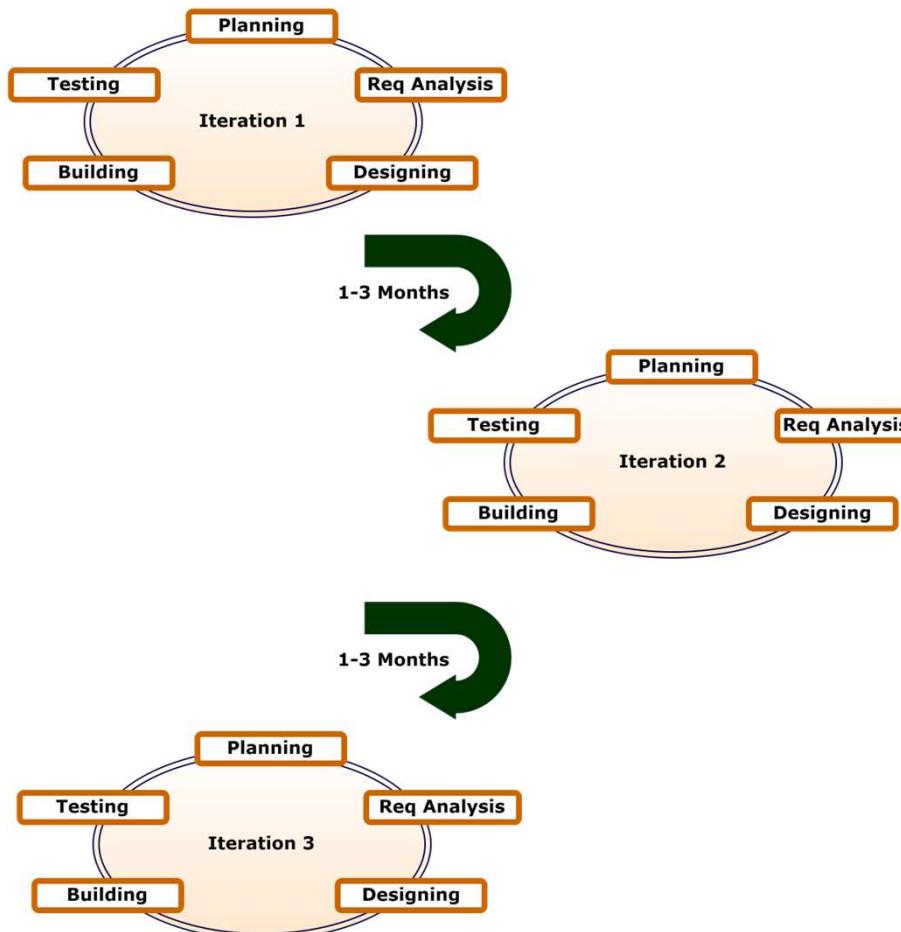
SDLC/ Agile model

Agile SDLC model is a combination of **iterative and incremental process models** with focus on process adaptability and customer satisfaction by **rapid delivery** of working software product.

Following are the Agile Manifesto principles

- ▶ **Individuals and interactions** : self-organization and motivation are important, as are interactions like co-location and pair programming.
- ▶ **Working software** : Communication with the customers to understand their requirements, instead of just depending on documentation.
- ▶ **Customer collaboration** : Continuous customer interaction is very important to get proper product requirements.
- ▶ **Responding to change** : Focused on quick responses to change and continuous development.

SDLC/ Agile model



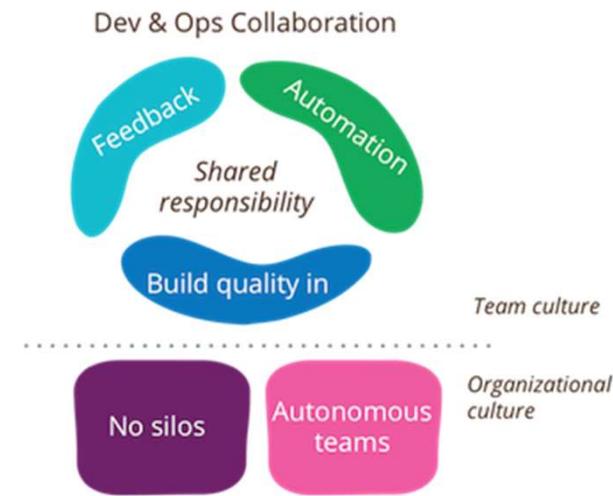
SDLC/ Agile model

Advantage and disadvantages are :

Advantages	Disadvantages
Promotes teamwork and cross training and Suitable for fixed or changing requirements	An overall plan, an agile leader and agile PM practice is a must without which it will not work.
Delivers early partial working solutions and Minimal rules, documentation easily employed.	Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
Enables concurrent development and delivery within an overall planned context.	Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.

SDLC/ Synthesis

- ▶ Agile software development has broken down some of the silos between requirements analysis, testing and development.
- ▶ Deployment, operations and maintenance are other activities which have suffered a similar separation from the rest of the software development process.
- ▶ The DevOps movement is aimed at removing these silos and encouraging collaboration between development and operations.
- ▶ DevOps has become possible largely due to a combination of new operations tools and established agile engineering practices,



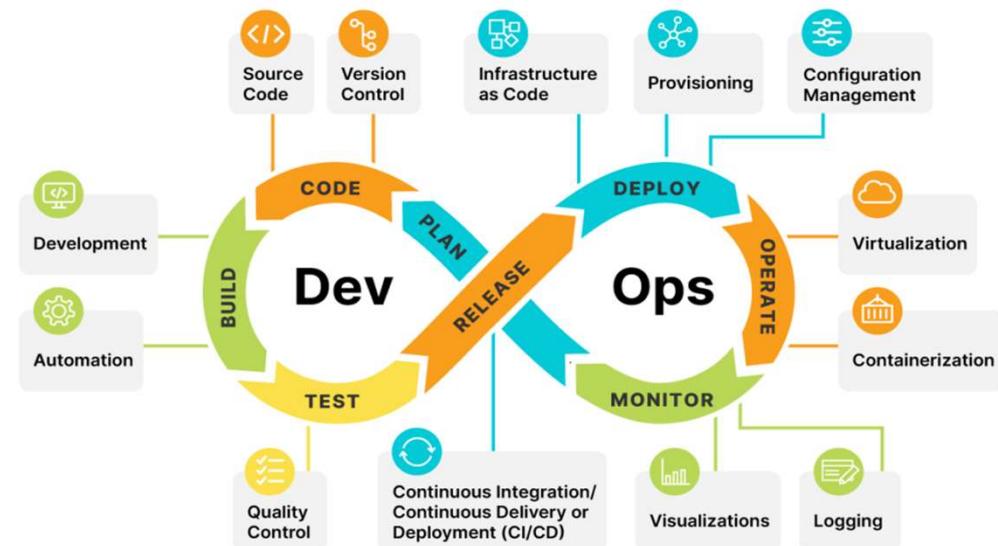
SDLC/ DevOps

DevOps combines development (**dev**) and operations(**ops**) to increase the efficiency, speed and security of software development and delivery compared to traditional processes.

It is defined as a **software engineering methodology** which aims to integrate the work of software development and software operations teams by facilitating a culture of **collaboration** and **shared responsibility**.

These four (04) key principles can improve the organization's software development practice :

- ▶ Automation of the Software development lifecycle
- ▶ Collaboration and communication
- ▶ Continuous improvement and minimization of waste
- ▶ Hyperfocus on user needs with short feedback loops.



SDLC/ DevOps benefits

The benefits of DevOps are :



SDLC/ DevOps and Agile Model

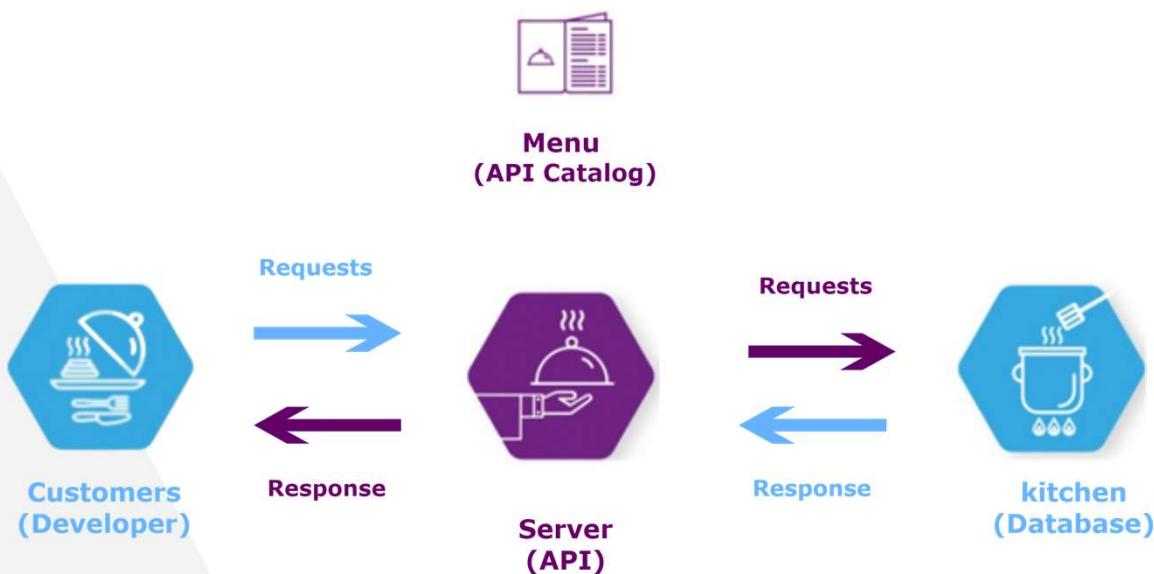
The following show the difference between Agile and DevOps:

Parameter/Software Development Technologies	Agile	DevOps
Purpose	Manage complex projects	Manage end-to-end engineering processes
Task	Focuses on constant changes	Focuses on constant testing and delivery
Implementation	Possible within a range of tactical frameworks (sprint, safe and scrum)	Collaboration (doesn't have any commonly accepted framework)
Feedback	By the customer	By the internal team
Goal	Gap between customer need and development & testing teams	Gap between development + testing and Ops
Advantage	Shorter development cycle and improved defect detection	Supports Agile's release cycle
Tools used	JIRA, Kanboard	Ansible, Jenkins, Git

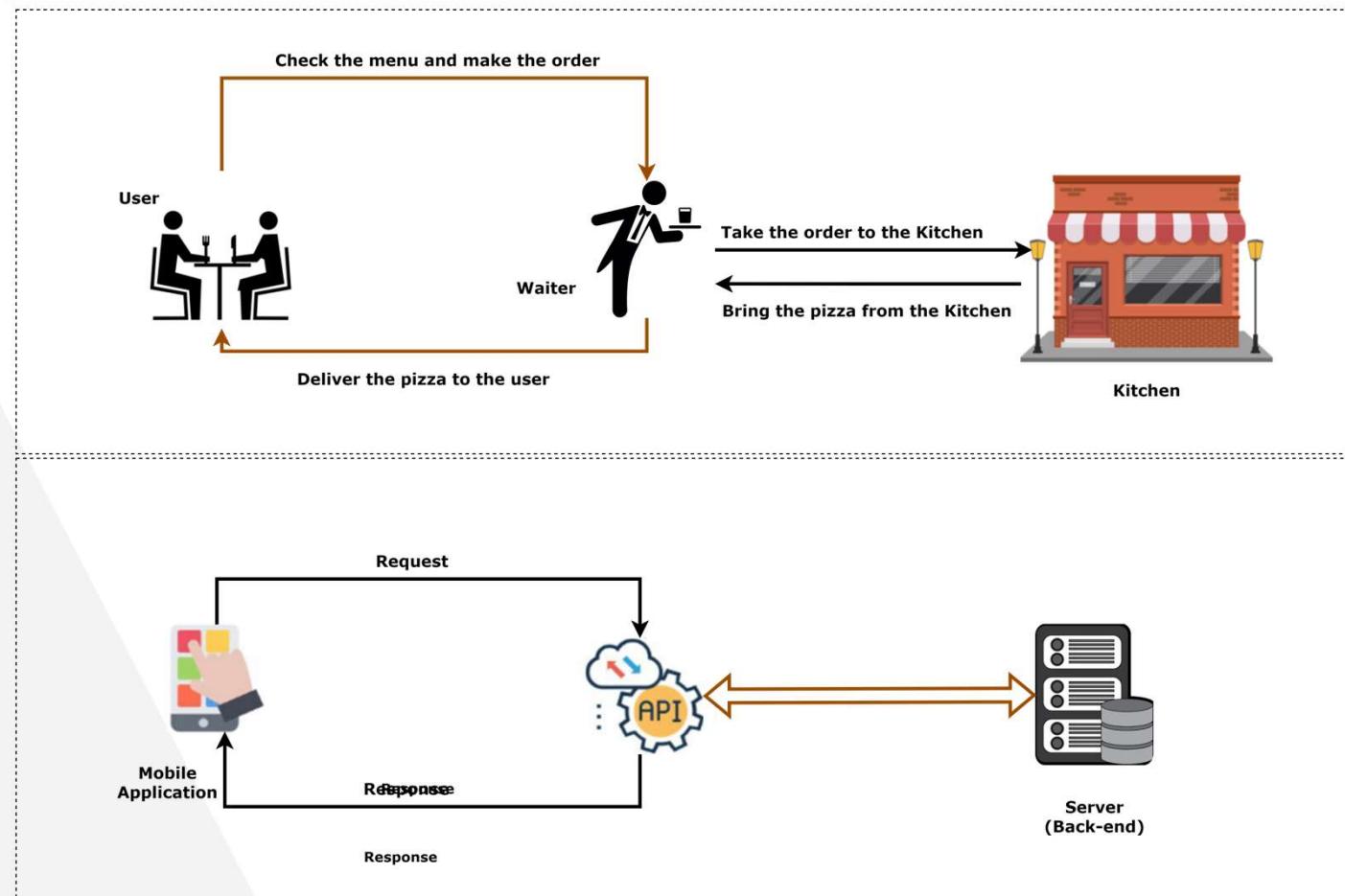
API Concepts and Standards

The term **API** is an acronym that stands for **Application Programming Interface**.

- ▶ For example, think of an API like a menu in a restaurant.
- ▶ The menu provides a list of pizzas you can order, along with a description of each pizza.
- ▶ **You don't know exactly how the restaurant prepares this food, and you don't really need to.**



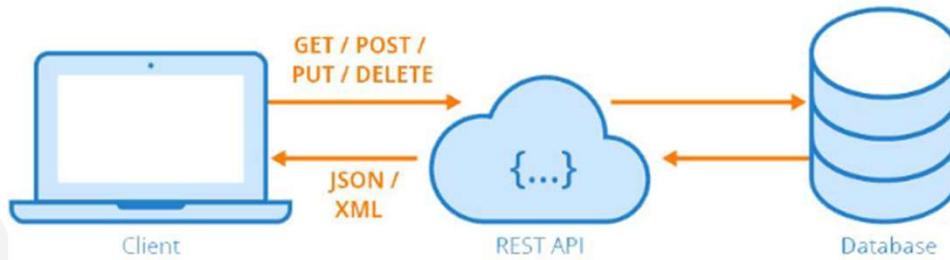
API Concepts and Standards



API Concepts and Standards / REST API

REST is an acronym for REpresentational State Transfer and an architectural style for **distributed hypermedia systems**.

REST is a way for two computer systems to communicate over HTTP in a similar way to web browsers and servers.



API Concepts and Standards / 6 REST API Constraints

- ▶ **Client-Server Architecture** : Client and server systems can be improved and updated independently each other
- ▶ **Statelessness** : All client requests are treated equally. There's no special, server-side memory of past client activity. The responsibility of managing state is on the client.
- ▶ **Cacheability** : Clients and servers should be able to cache resource data that changes infrequently further improving scalability and performance.
- ▶ **Layered System** : A client cannot ordinarily tell whether it is connected directly to the end server or an intermediary along the way. Intermediary servers can also improve system scalability.
- ▶ **Code on demand (optional)** : Servers can temporarily extend or customize the functionality of a client by transferring executable code.
- ▶ **Uniform interface** : All resources should be accessible through a common approach such as HTTP GET and similarly modified using a consistent approach.

API Concepts and Standards / What is JSON ?

- ▶ JSON stands for **JavaScript Object Notation**
- ▶ JSON is a lightweight format for storing and transporting data
- ▶ JSON is often used when data is sent from a server to a web page
- ▶ JSON is "self-describing" and easy to understand

Syntax rules

- ▶ Data is in name/value pairs
- ▶ Data is separated by commas
- ▶ Curly braces hold objects
- ▶ Square brackets hold arrays

```
[  
  {  
    "id": 1,  
    "name": "Leanne Graham",  
    "username": "Bret",  
    "email": "Sincere@april.biz",  
    "address": {  
      "street": "Kulas Light",  
      "suite": "Apt. 556",  
      "city": "Gwenborough",  
      "zipcode": "92998-3874",  
      "geo": {  
        "lat": "-37.3159",  
        "lng": "81.1496"  
      }  
    },  
    "phone": "1-770-736-8031 x56442",  
    "website": "hildegard.org",  
    "company": {  
      "name": "Romaguera-Crona",  
      "catchPhrase": "Multi-layered client-server neural-net",  
      "bs": "harness real-time e-markets"  
    }  
  },  
  {  
    "id": 2,  
    "name": "Ervin Howell",  
    "username": "Antonette",  
    "email": "Shanna@melissa.tv",  
    "address": {  
      "street": "Victor Plains",  
      "suite": "Suite 879",  
      "city": "Wisokylburgh",  
      "zipcode": "90566-7771",  
      "geo": {}  
    }  
  }]
```

RESTful Web Service (Request)

A RESTful web service request contains :

- ▶ **An Endpoint URL** : An application implementing a RESTful API will define one or more URL endpoints with a domain, port, path, and/or query string for example, <https://mydomain/user/123?format=json>
- ▶ **The HTTP method** : Differing HTTP methods can be used on any endpoint which map to application create, read, update, and delete (CRUD) operations :

HTTP method	CRUD	Action
GET/HEAD/OPTIONS	read	Returns requested data/Same as GET but does not return a body>Returns the supported HTTP methods
POST	create	Returns requested data
PUT or PATCH	update	Updates an existing record
DELETE	delete	Deletes an existing record

- ▶ **HTTP headers** : Information such as authentication tokens or cookies can be contained in the HTTP request header.
- ▶ **Body Data** : Data is normally transmitted in the HTTP body in an identical way to HTML <form> submissions or by sending a single JSON-encoded data string

RESTful Web Service(Response)

- ▶ The response payload can be whatever : data, HTML, an image, an audio file, and so on.
- ▶ **Data responses** are typically **JSON-encoded**, but XML, CSV, simple strings, or any other format can be used.
- ▶ Return format could be specified in the request. For example, `/user/123?format=json` or `/user/123?format=xml`

An appropriate **HTTP status code** should also be set in the response header.



REST challenges

Several challenges are possible :

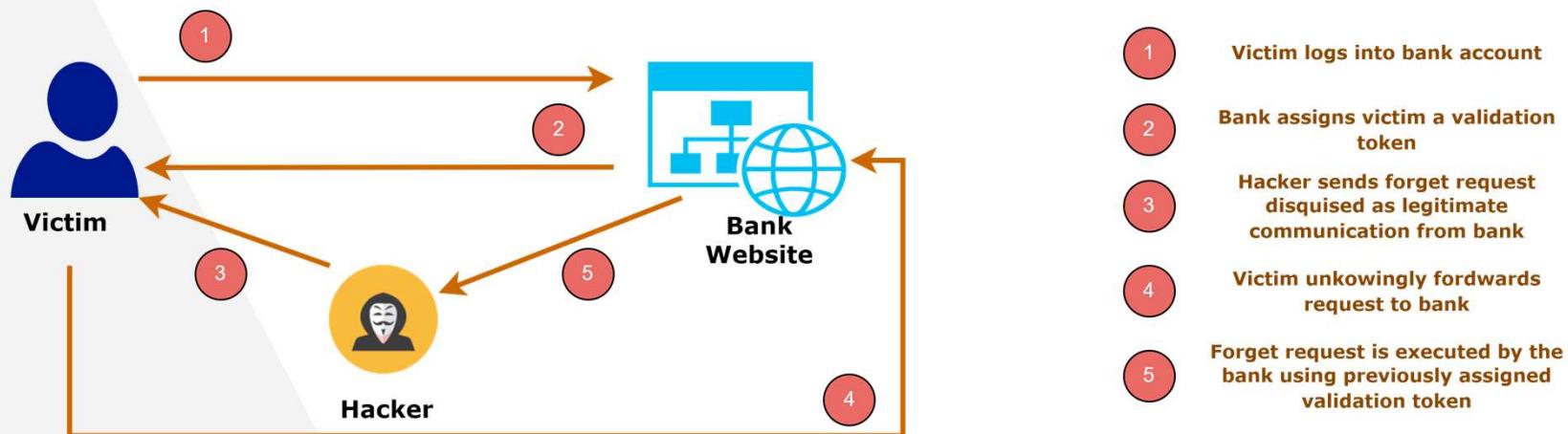
- ▶ **API Versioning** : API changes are inevitable, but endpoint URLs should never be invalidated when they're being used internally and/or by third-party applications
- ▶ **Authentication** : Client-side applications on the same domain as the RESTful API will send and receive cookies. An API request can therefore be validated to ensure a user is logged in and has appropriate rights.
- ▶ **Security** : A RESTful API provides another route to access and manipulate your application.

Common best practices :

- ▶ Use HTTPS
- ▶ Use a robust authentication method
- ▶ Use **CORS** to limit client-side calls to specific domains
- ▶ Provide minimum functionality
- ▶ Validate all endpoint URLs and body data
- ▶ Avoid exposing **API tokens** in client-side Javascript
- ▶ Block unexpectedly large payloads.

CORS headers and CSRF token

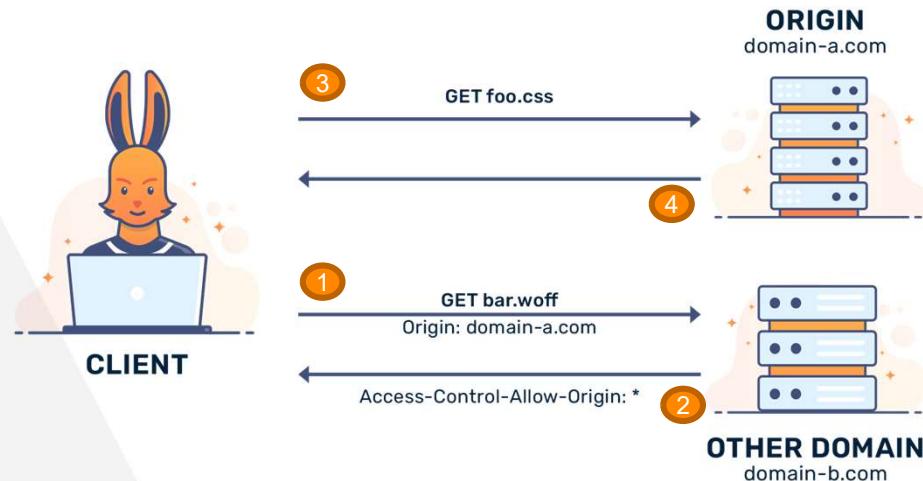
- ▶ A **CSRF (Cross-Site Request Forgery) token** is a unique, secret, unpredictable value that is generated by the server-side application
- ▶ CSRF token is transmitted to the client in such a way that it is included in a subsequent HTTP request made by the client.
- ▶ CSRF tokens can prevent **CSRF attacks** by making it impossible for an attacker to construct a fully valid HTTP request suitable for feeding to a victim user.



CORS headers and CSRF token

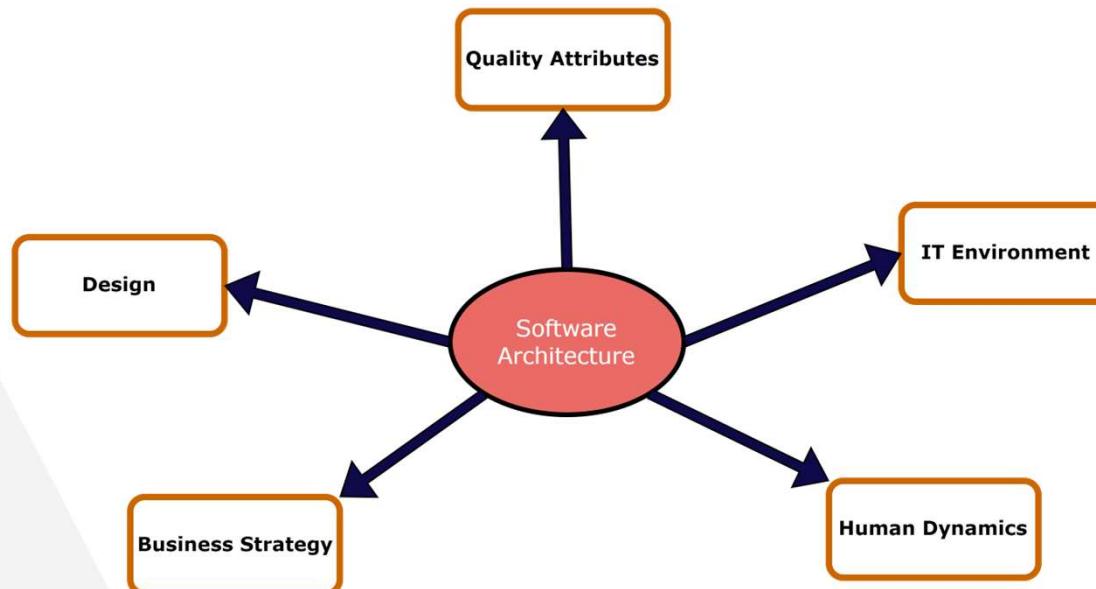
Cross-origin resource sharing (CORS) is a mechanism that consists of adding HTTP headers to allow a user agent to access resources on a server located on another origin than the current site.

- ▶ **CORS** is used to bypass a certain basic setting like SOP (Same-Origin Policy) which prohibits loading from other servers when visiting a website
- ▶ **CORS** does not protect against **CSRF attacks** or unwanted users



Modern Software Architecture

- ▶ The **architecture** of a system describes its major components, their relationships (structures), and how they interact with each other.
- ▶ **Software architecture** and **design** includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment. Software **architecture # Software design**.



Modern Software Architecture

Software Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- Fundamental properties and define guidelines
- Cross-cutting concerns and high-impact
- Communicate with business stakeholders
- Manage uncertainty
- Conceptual integrity
- Scope: System



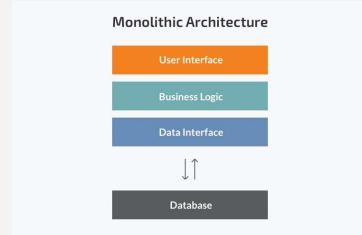
Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows

- Detailed properties
- Communicate with developers
- Individual components
- Use guidelines
- Avoid uncertainty
- Scope: Module



Modern Software Architecture / Types

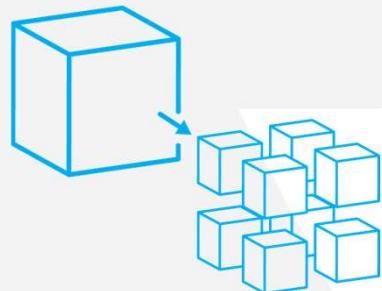
Monolithic Architecture



Service Oriented Architecture (SOA)



Microservices Architecture



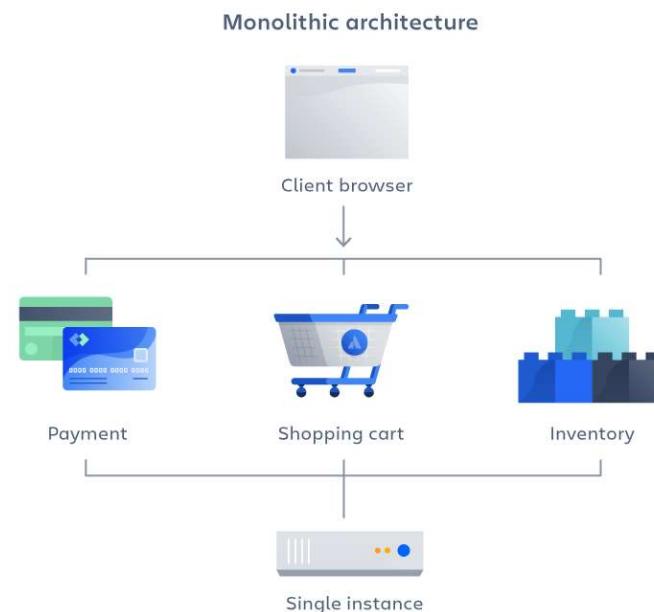
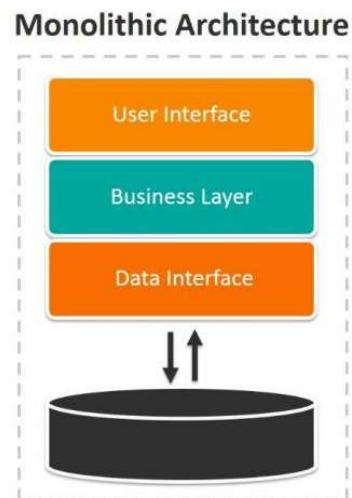
Serverless Architecture



Monolithic Architecture

A **Monolithic architecture** is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications.

- ▶ Traditional solution
- ▶ Comfortable for small teams
- ▶ Interconnected and interdependent
- ▶ Software self-contained



Monolithic Architecture

Advantage and disadvantages are :

Advantages	Disadvantages
Easy development and deployment	Slower development speed and not adapted to change for deployment
High performance task	Not scalability (for individual components) and not reliability
Simplified and fast testing	Lack of flexibility
Easy debugging	Barrier to technology adoption

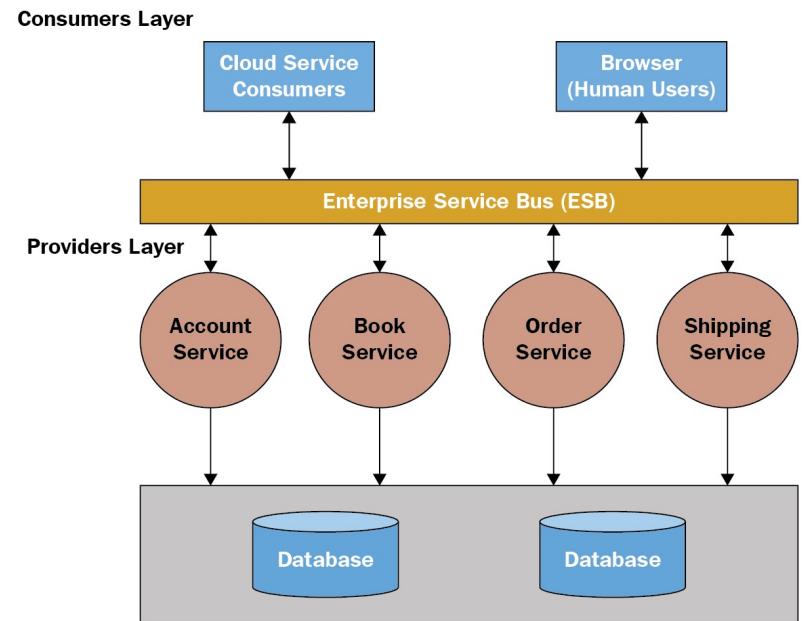
Service-Oriented Architecture (SOA)

In **SOA**, a **service** is a self-contained unit of software designed to complete a specific task.

Service-oriented architecture allows various services to communicate using a **loose coupling** system to either pass data or coordinate an activity.

The defining concepts of SOA are listed with high priority :

- ▶ **Business value**
- ▶ **Strategic goals**
- ▶ **Basic interoperability**
- ▶ **Shared services**
- ▶ **Continued improvement**



Service-Oriented Architecture (SOA)

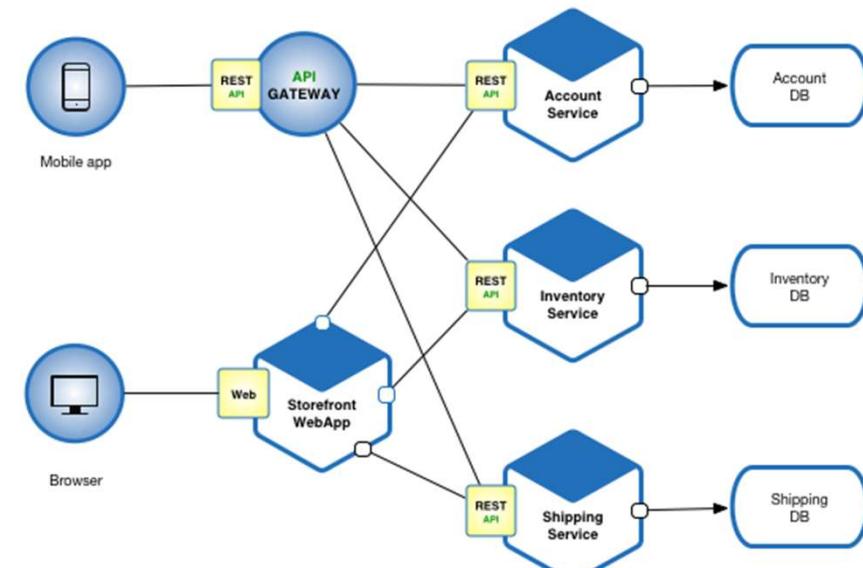
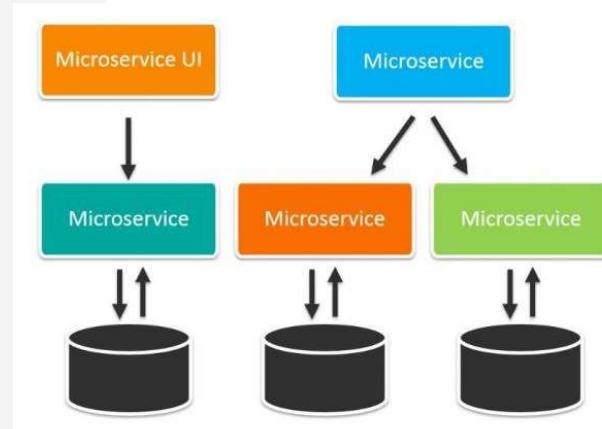
Advantage and disadvantages are :

Advantages	Disadvantages
Service reusability	High overhead
Easy maintenance	High investment
Platform independent	Complex service management
Availability, reliability and scalability	Complex maintenance

Microservice Architecture

Microservice is a type of service-oriented software architecture that focuses on building a series of autonomous components that make up an application. It is an architectural style that structures an application as a collection of services that are :

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team



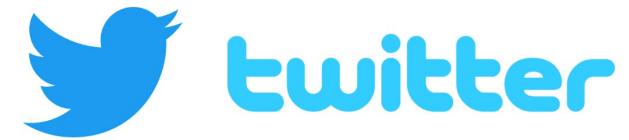
Microservice Architecture

Advantage and disadvantages are :

Advantages	Disadvantages
Increased agility	Increased complexity
Improved workflows	More expensive
Decreased the amount of time it takes to improve production	Greater security risks
Ability to scale horizontally	Different programming languages

Comparison of Architectures

Companies that have evolved from a monolithic approach to microservices :



Comparison of Architectures

In resume :

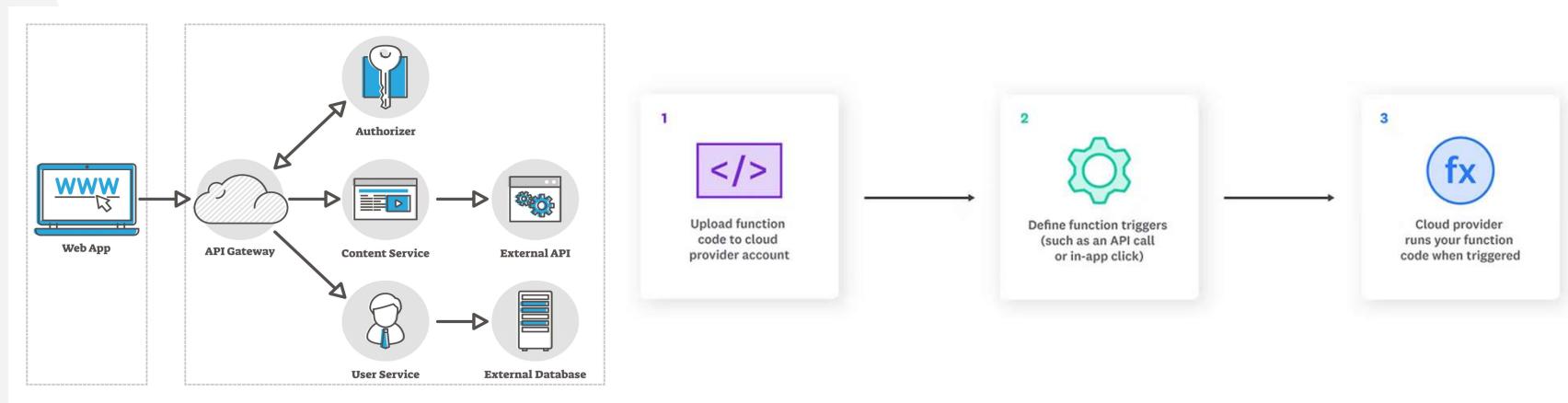
- ▶ **Monolithic** apps consist of **interdependent, indivisible units** and feature very slow development speed.
- ▶ **SOA** is broken into smaller, moderately coupled services and features slow development.
- ▶ **Microservices** are very small, **loosely coupled** independent services and feature **rapid continuous development**.



Serverless Architecture

Serverless architecture is an approach to software design that allows developers to build and run services without having to manage the underlying infrastructure.

- ▶ **Code execution is managed by a server**
- ▶ **Serverless doesn't mean "no server"**
- ▶ **Third-party cloud service like AWS/Azure takes full responsibility for these servers**



Serverless Architecture

Advantage and disadvantages are :

Advantages	Disadvantages
Easy to deploy	Security dependent on the service provider
Enhanced scalability	Privacy & Vendor lock-in : Shared resources in Cloud
Lower costs	Not for long-term tasks
Accuracy on function development	Complexity of troubleshooting

Which Architecture?

Which architecture to choose?

Monolithic



Startups,
Small Apps



Small resource
base

SOA



Enterprise apps with
complex operations

Microservices



Complex large-scale
systems



Multiple skilled teams

Serverless



Client-heavy apps



Fast-growing and
rapidly changing apps



High-latency
background tasks

MODULE I-2 : Standard Components and Platforms for Software

PLAN

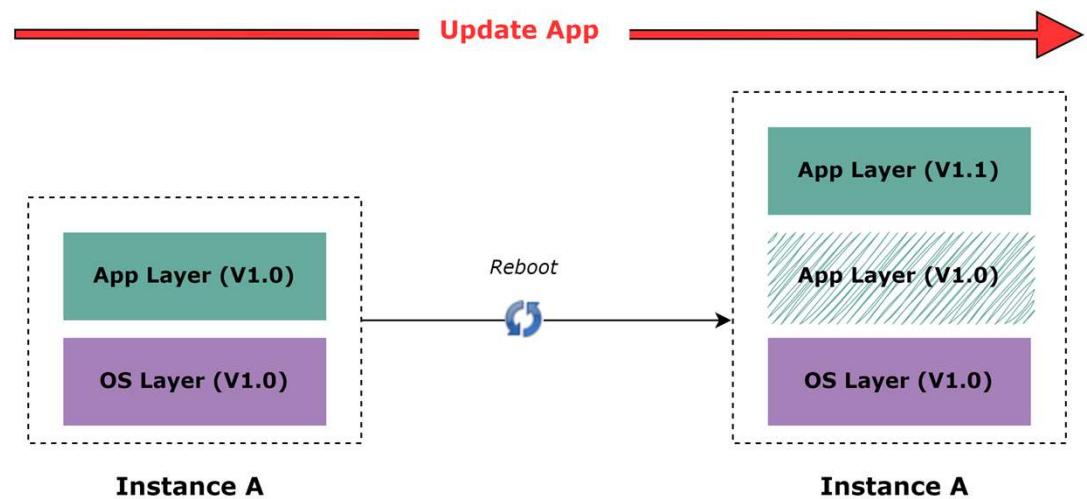
- **INFRASTRUCTURE:**
 - Mutable
 - Immutable
- **DATA STORAGE :**
 - Object
 - Block
 - File
- **DATA STRUCTURE:**
- **CAP & ACID**
- **DATABASE**
 - SQL
 - NoSQL
 - Serverless
- **MESSAGE QUEUES**
- **CLOUD COMPUTING**
 - Key features
 - Benefits
 - Service model
 - Deployment model



Mutable infrastructure

Mutable Server means the infrastructure will be continually updated, tweaked, and tuned to meet the ongoing needs of the purpose it serves.

- ▶ Ability to change
- ▶ Updating Operating System
- ▶ Updating Software



Mutable infrastructure

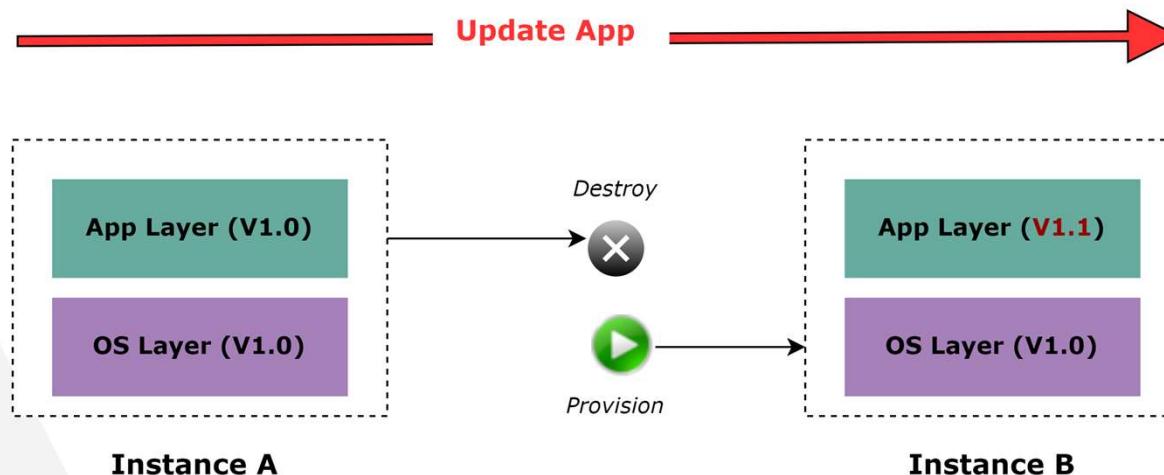
Advantage and disadvantages are :

Advantages	Disadvantages
IT team does not need to build servers from scratch every time a change is required.	Configuration drift (harder to diagnose and manage each server)
Roll out updates for individual servers, which makes the update process faster.	Indiscrete versioning
Ensure that the infrastructure used meets the specific needs of each user.	Updates can fail due to several reasons. Debugging is time consuming due to update tracking problems.

Immutable infrastructure

Immutable Server means the infrastructure cannot be modified once deployed. When changes are necessary, it is recommended to deploy afresh, add infrastructure and decommission old infrastructure.

- ▶ **No updates, security patches or configuration changes**
- ▶ **New version of the architecture is built and deployed**
- ▶ **New servers are deployed instead of updating the ones already used**



Immutable infrastructure

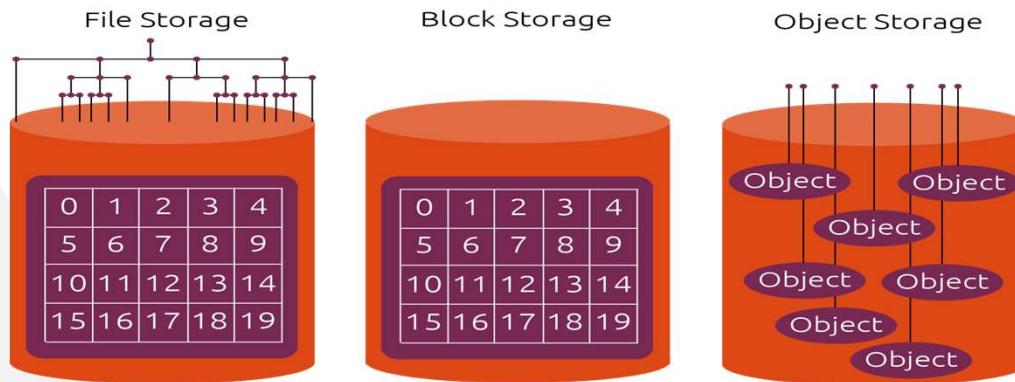
Advantage and disadvantages are :

Advantages	Disadvantages
Discrete versioning (Each server version is independent of the other.)	Impossible to modify existing servers
Easier tracking, testing different servers and rolling back	In case of problems, servers with the same configuration need a complete overhaul.
Great for interdependent environments such as cloud technologies.	Externalize data storage instead of copying it to a local disk.

Data Storage

Data storage is the retention of information using technology specifically developed to keep that data and have it as accessible, as necessary.

- ▶ **File Storage** is a hierarchical storage methodology used to organize and store data on a computer hard drive or on a network-attached storage (NAS)
- ▶ **Block Storage** is when a category of data storage is saved in huge volumes known as blocks
- ▶ **Object Storage** is a computer data storage architecture that manages data as objects



Data Storage Comparison

	Object Storage	File-based Storage	Block-based storage
TRANSACTION UNITS	Objects(files with custom metadata)	Files	Blocks
SUPPORTED TYPE OF UPDATES	No in-place update support/Updates create new object versions	Supports in-place updates	Supports in-place updates
PROTOCOLS	REST and SOAP over HTTP	SMB and NFS	SCSI, Fibre Channel, SATA
BEST SUITED FOR	Relatively static file data and as Cloud storage	Shared file data	Transactional data and frequently changing data
BIGGEST STRENGTH	Scalability and distributed access	Simplified access and management of shared files	High performance
LIMITATIONS	Doesn't provide a sharing protocol with a locking mechanism	Difficult to extend beyond the data center	Difficult to extend beyond the data center

Type of Data Structure

For the analysis of data, it is important to understand that there are three common types of data structures :

- ▶ **Semi-Structured Data** is a form of structured data that does not conform with the formal structure of data models associated with relational databases or other forms of data tables
- ▶ **Structured Data** is data that adheres to a pre-defined data model and is therefore straightforward to analyze.
- ▶ **Unstructured Data** is information that is not organized in a pre-defined manner.

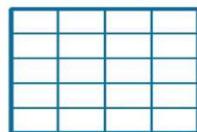


File System

C:\folder\music.m4a

sysadmin required for integrity and scale

Semi-Structured Data



Database / Structured Data

SELECT * FROM table;
INSERT INTO table;

sysadmin and DBA required for scale, integrity and performance

Structured Data



Object Storage

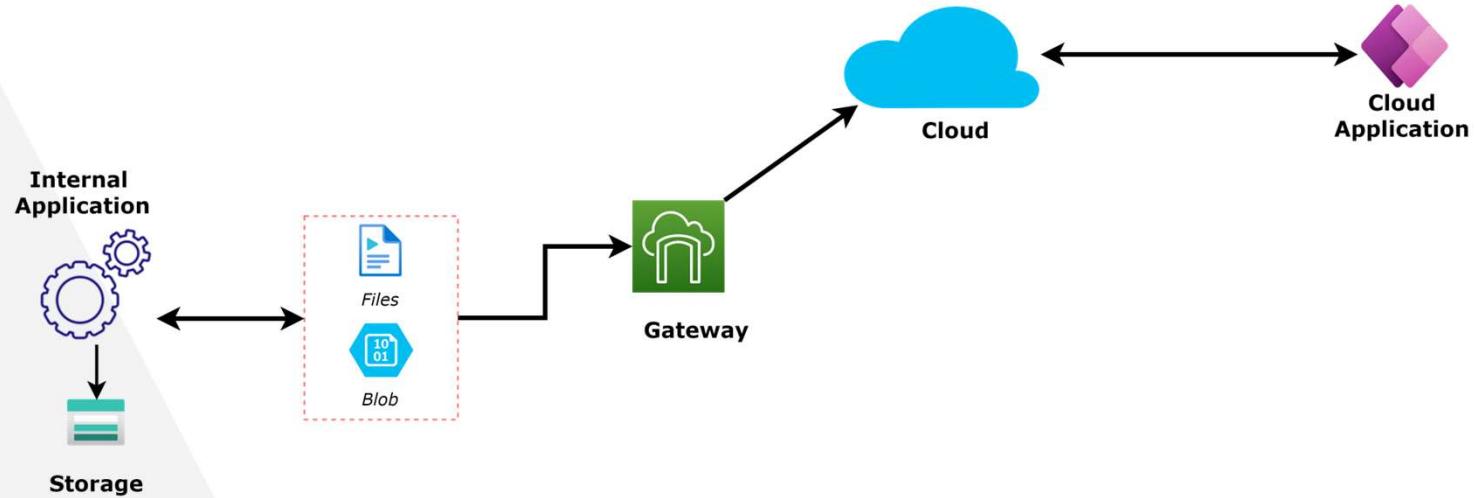
GET /object/KbglBn7qepo
PUT /object/KbglBn7qepo

sysadmin not required

Unstructured Data

Storage Use Case

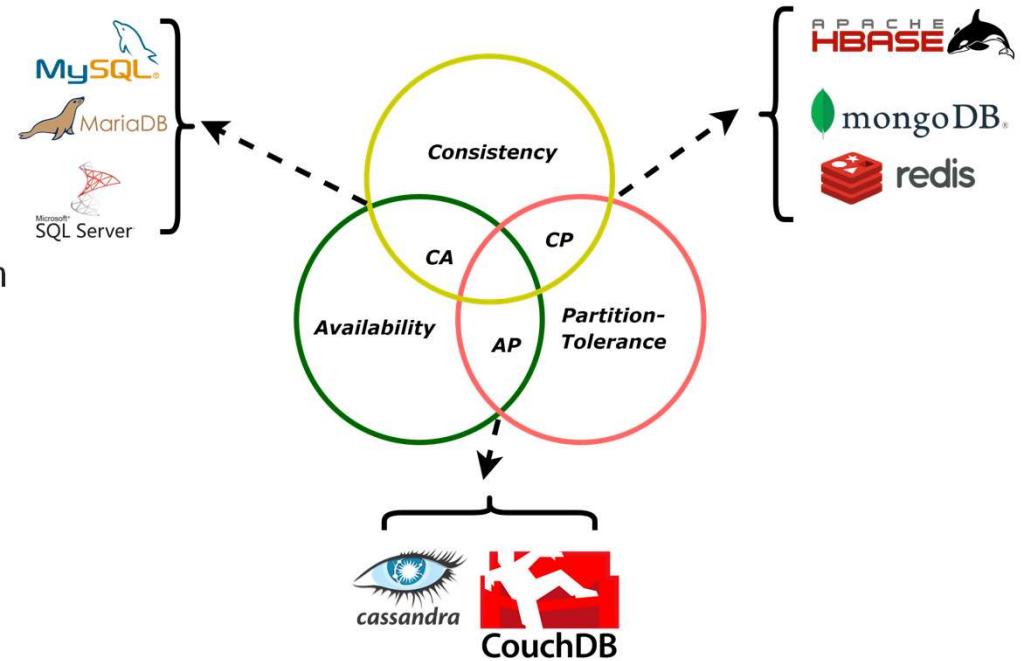
- ▶ Native cloud applications
- ▶ Big data analysis
- ▶ Internet of Things
- ▶ Storage and distribution of rich media content
- ▶ Backup and archiving



CAP & ACID

In normal operations, your data store provides all three functions. But the **CAP** theorem maintains that when a distributed database experiences a network failure, you can provide either consistency or availability.

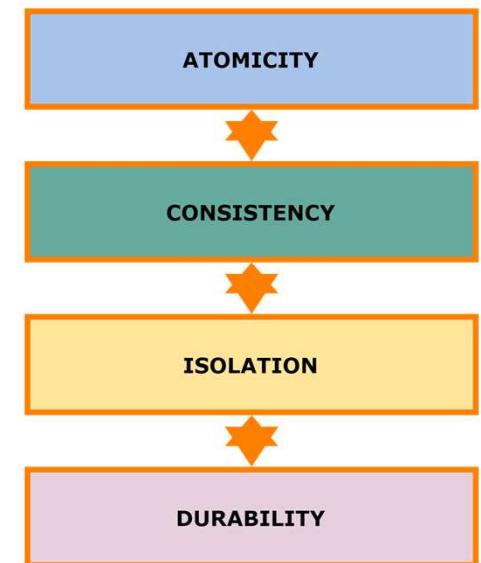
- ▶ **Consistency** : All clients see the same data at the same time
- ▶ **Availability** : The system continues to operate even in the presence of node failures
- ▶ **Partition tolerance** : The system continues to operate despite network failures



CAP & ACID

ACID describe the set of properties of database transactions that guarantee data integrity despite errors, system failures, power failures, or other issues.

- ▶ **Atomicity** : Commits finish an entire operation successfully or the database rolls back to its prior state
- ▶ **Consistency** : Any change maintains data integrity or is cancelled completely
- ▶ **Isolation** : Any read or write will not be impacted by other reads or writes of separate transactions
- ▶ **Durability** : Successful commits will survive permanently



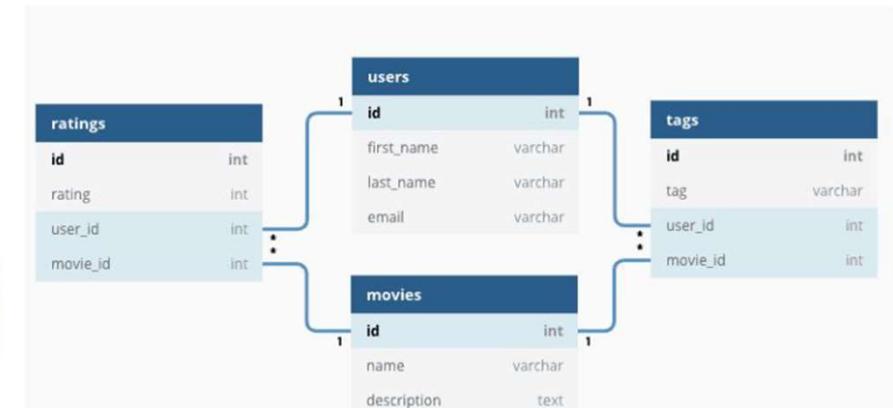
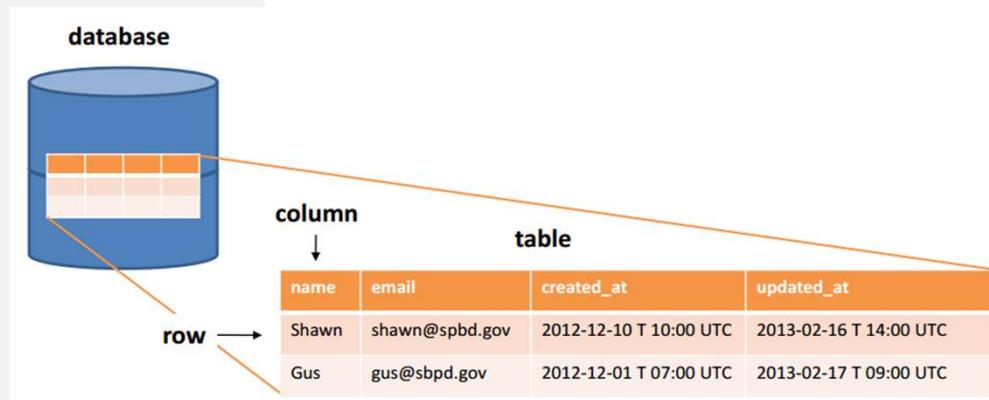
Database

There are following types of databases :

- ▶ **Relational Database** : These databases are categorized by a set of tables where data gets fit into a pre-defined category.
 - The table consists of rows and columns where the column has an entry for data for a specific category
 - Rows contains instance for that data defined according to the category.
 - The **Structured Query Language (SQL)** is the standard user and application program interface.
- ▶ **NoSQL Database** : These are used for large sets of distributed data.
 - There are some **big data** performance issues which are effectively handled by relational databases, such kind of issues are easily managed by NoSQL databases.
 - There are very efficient in analyzing large size unstructured data that may be stored at multiple virtual servers of the cloud.

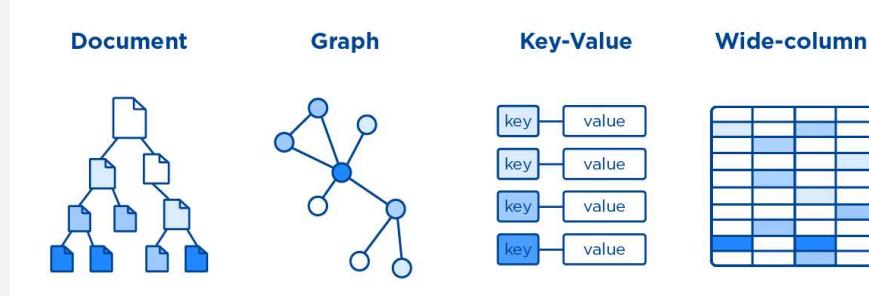
Database/Relational Database

- ▶ Based on the relational model of data
- ▶ Comply with ACID guarantees
- ▶ Relational Database systems (RDBS) use SQL
- ▶ Relational model organizes data into one or more tables
- ▶ Each row in a table has its own unique key (primary key)
- ▶ Rows in a table can be linked to rows in other tables by adding a foreign keys
- ▶ Exemple : MySQL (MariaDB), Oracle, Postgres, IBM DB, ...



Database/NoSQL Database

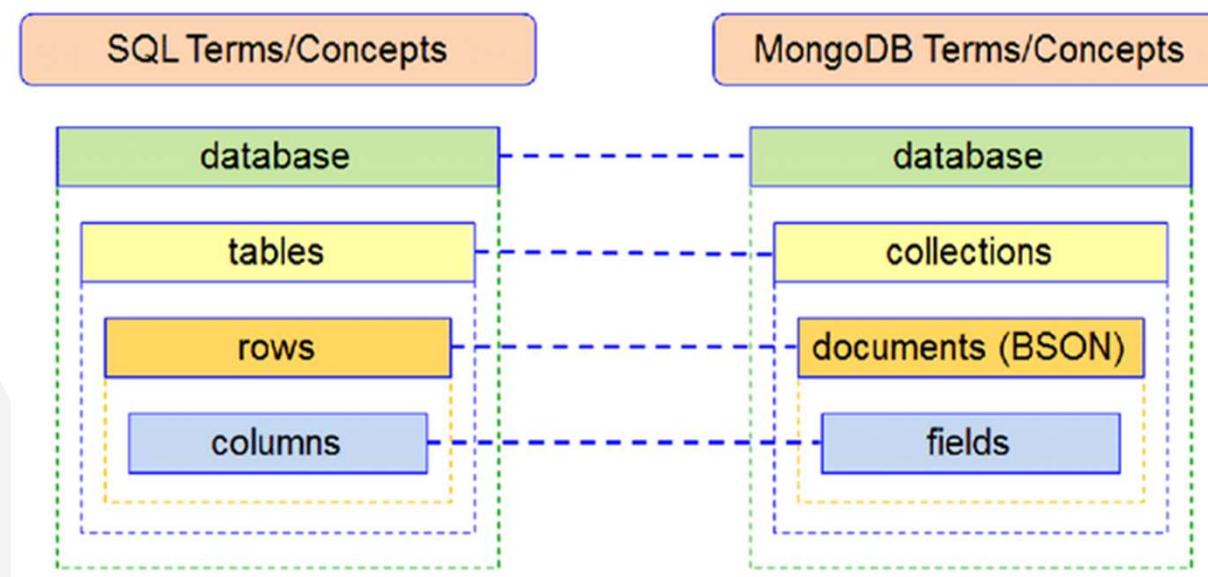
- ▶ Not only SQL
- ▶ Storing semi-structured and non-structured data
- ▶ Horizontal scalability and partition tolerance
- ▶ Sub-second response times are required for large volumes of data, compromising consistency and referential integrity.
- ▶ Finer control over availability
- ▶ Simplicity of design



Document Database	Graph Databases
Couchbase	Neo4j InfiniteGraph The Distributed Graph Database
MarkLogic	mongoDB
Wide Column Stores	Key-Value Databases
redis	accumulo
amazon DynamoDB	HYPERTABLE
riak	Apache Cassandra
AEROSPIKE	HBASE
	Amazon SimpleDB

@cloudit: <http://www.aryannava.com>

Database/Difference between SQL and NoSQL

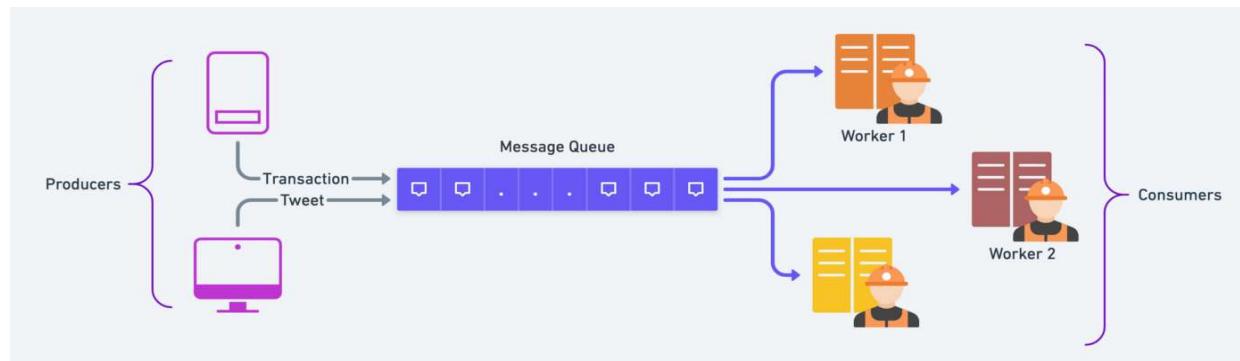


Data Storage Comparison

	Pros	Cons
SQL	Standardized schema Large user community No code required ACID compliance	Hardware Data normalization Rigidity Resource-intensive scaling
NoSQL	Continuous availability Query speed Agility Cost	No standardized language Smaller user community Inefficiency with complex queries Data retrieval inconsistency

Message Queues

Message queuing allows applications to communicate by sending messages to each other. The message queue provides temporary message storage when the destination program is busy or not connected.



- ▶ Producer creates message and send it to queue which stores the message if consumer is busy
- ▶ Consumer retrieve the message from the queue and start processing it
- ▶ Queue temporarily locks the message to prevent it from being read by another consumer
- ▶ Consumer delete the message from the queue after it completes the message processing



Cloud Computing/Definition

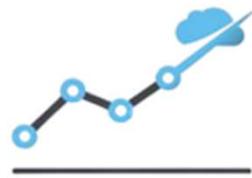
Cloud Computing "Using resources without directly owning them" is a model that allows ubiquitous, convenient, on-demand access to a shared network and a set of configurable computing resources. "NIST"

- 05 Key features
- 03 Service model
- 04 Deployment model
- Benefits



Cloud Computing/key features

- ▶ **Elasticity and rapid evolution**



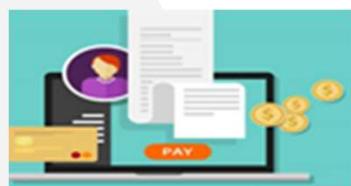
- ▶ **Resource pooling**



- ▶ **On-demand service**



- ▶ **Measurable and billable service**



- ▶ **Universal access via the network**



Cloud Computing/Benefits

- ▶ Benefit from massive economies of scale



- ▶ Increase speed and agility



- ▶ Stop spending money to manage data centers



- ▶ Shift from capital expenditures (CAPEX) to operational expenditures (OPEX)



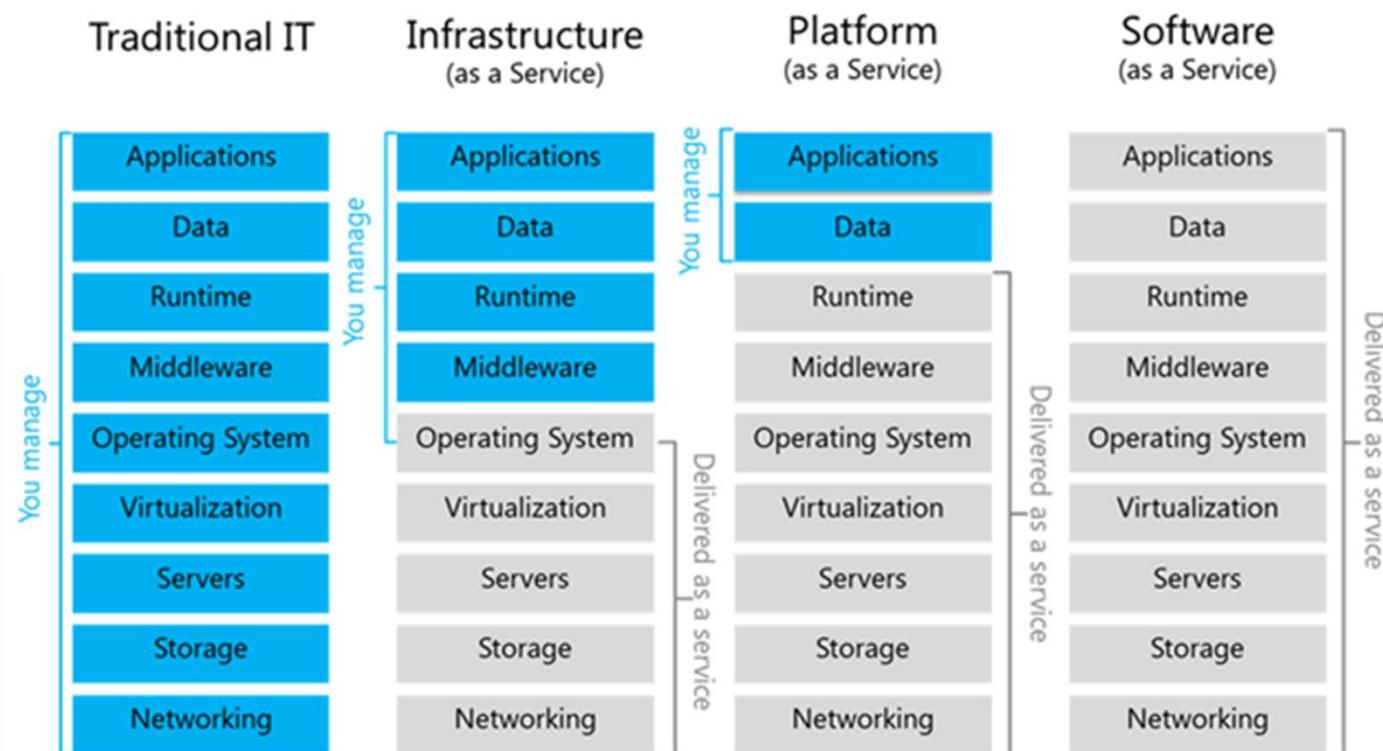
- ▶ Go global in minutes



- ▶ Stop guessing capacity



Cloud Computing/Service model



Cloud Computing/Service model

► **Infrastructure As a Service (IAAS):**

Amazon EC2

Azure VM, Digital Ocean, Linode



► **Platform As A Service (PAAS) :**

AWS ElasticBeanstalk

Azure App Services, Google App Engine



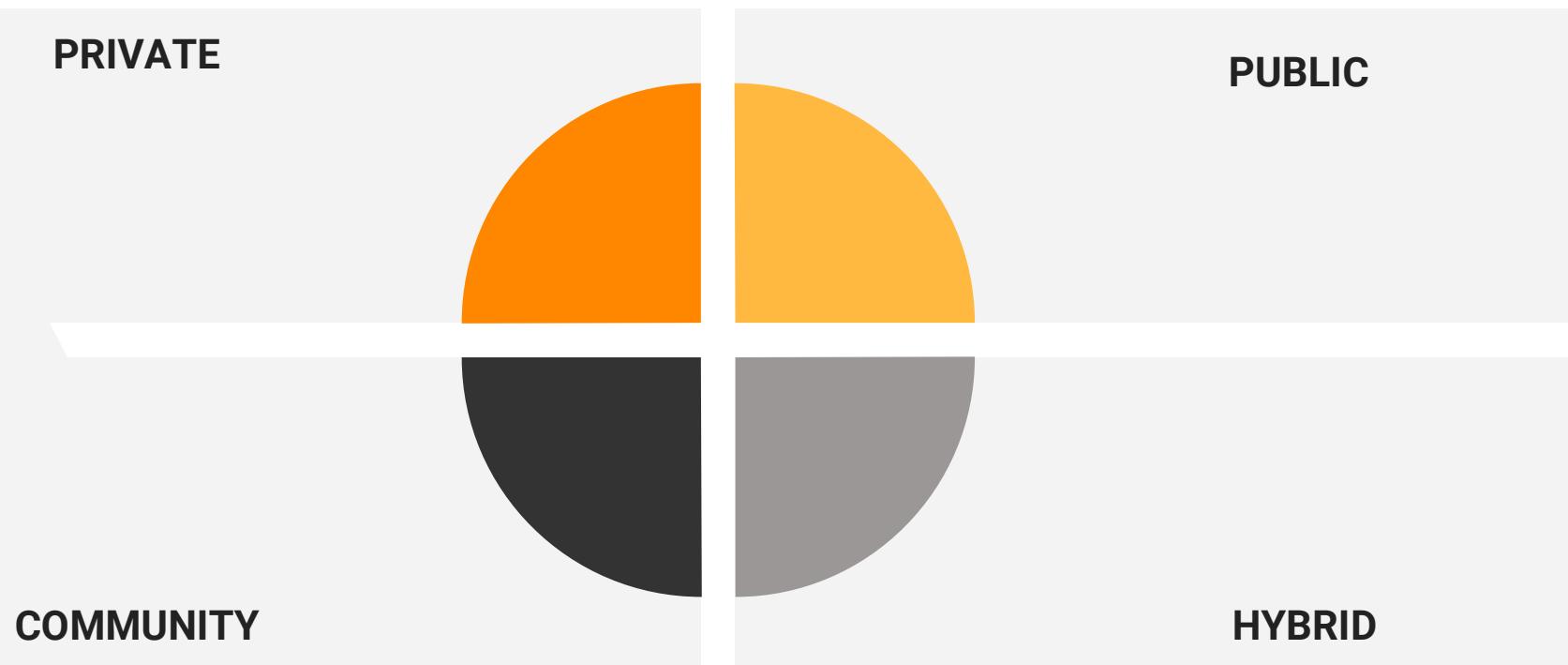
► **Software As A Service (SAAS) :**

Amazon Rekognition, Comprehend

Gmail, Dropbox, Outlook, Zoom



Cloud Computing/Deployment model



Cloud Computing/Stakeholders



Google Cloud Platform



IBM Cloud



MODULE I-3 : Source Code Management

PLAN

- ▶ **VERSION CONTROL SYSTEM:**
 - Benefits
 - Local
 - Centralized
 - Distributed
- ▶ **GIT :**
 - Installation
 - Configuration
 - Terminology
 - LifeCycle
 - Workflow
- ▶ **GIT COMMAND**
- ▶ **GIT BRANCHING**
- ▶ **GIT MERGING**
- ▶ **GIT REBASE**

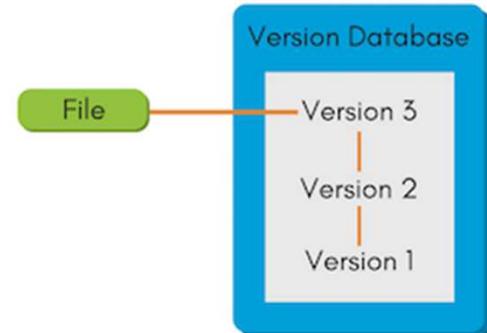


Version Control Systems

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code in a special kind of database.

Version control are sometimes referred to as :

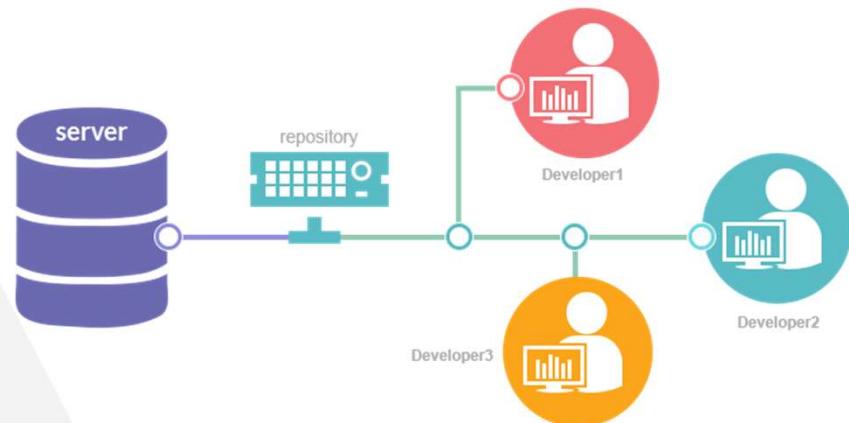
- ▶ Source code management systems (**SCMS**)
- ▶ Version Control Systems (**VCS**)
- ▶ Revision Control Systems (**RCS**)
- ▶ Code Repositories



Version Control Systems / Benefits

Benefits of the Version Control System are :

- ▶ Enhance the project development speed by providing efficient collaboration,
- ▶ Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change
- ▶ Contribute from anywhere irrespective of the different geographical locations
- ▶ Helps in recovery in case of any disaster or contingent situation
- ▶ Inform stakeholders about Who, What, Why changes have been made



Version Control Systems / Types

Three types of Version Control Systems :

► **Local Version Control Systems :**

- Contain database that kept all the changes to files under revision control.
- Keep patch sets (differences between files) in a special format on disk

► **Centralized Version Control Systems**

- Have a single "Central" copy of your project on a server
- Commit changes to this central copy
- Never have a full copy of project locally

► **Distributed Version Control Systems**

- Version control is mirrored on every developer's computer
- Allows branching and merging to be managed automatically
- Ability to work offline



Version Control Systems /Local

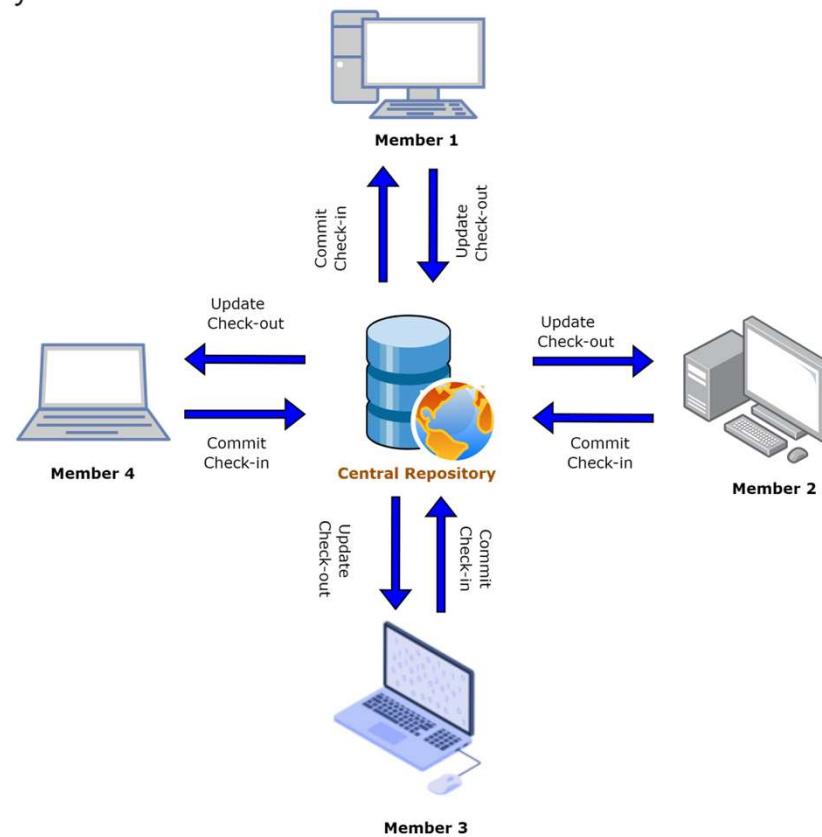
It is one of the simplest forms :

- ▶ Like a **VCS** but without a remote repository => **No remote server**
- ▶ Manage and version all the files only within your local system
- ▶ All the changes are recorded in a local database
- ▶ Every developer has their own computers and are **not sharing** anything



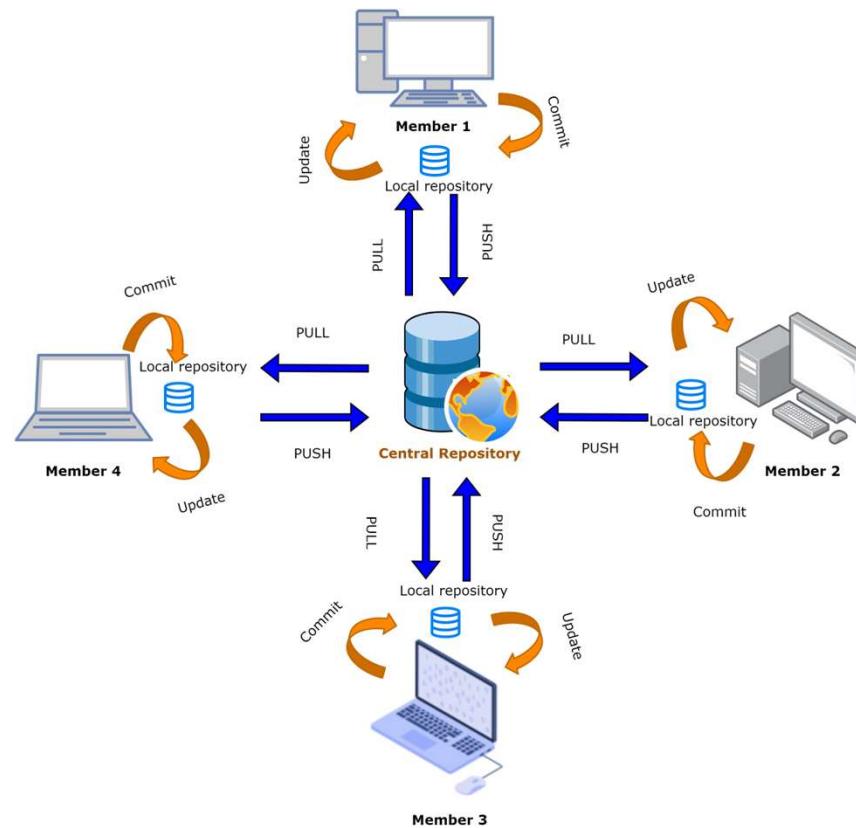
Version Control Systems /Centralized

The concept of a **Centralized system** is that it works on a Client-Server relationship. The repository is located at one place and provides access to many clients.



Version Control Systems /Distributed

In a **Distributed system** every user has a local copy of the repository in addition to the central repository on the server side.



Version Control Systems

Distributed Version Control Systems: contain multiple repositories. Each user has their own repository and working copy.

To make your changes visible to others, 4 things are required:

- ▶ You commit
- ▶ You push
- ▶ They pull
- ▶ They update



Git



- ▶ Open-source version control system
- ▶ Provides strong support for non-linear development
- ▶ Distributed repository model
- ▶ Cryptographic authentication of history
- ▶ Capable to efficiently handling small to large sized projects

Advantages	Disadvantages
Super-fast and efficient performance, cross-platform	Complex and bigger history log difficult to understand
Code changes easy and clearly tracked	Does not support keyword expansion
Easy maintainable and robust	Does not support timestamp preservation

Git installation

On Windows :

- ▶ <https://gitforwindows.org/>
- ▶ <http://babun.github.io/> (Shell Emulator)

On OS X :

- ▶ <https://sourceforge.net/projects/git-osx-installer/>

On Linux

- ▶ <your package manager> install git
(apt-get, rpm,...)

Git configuration

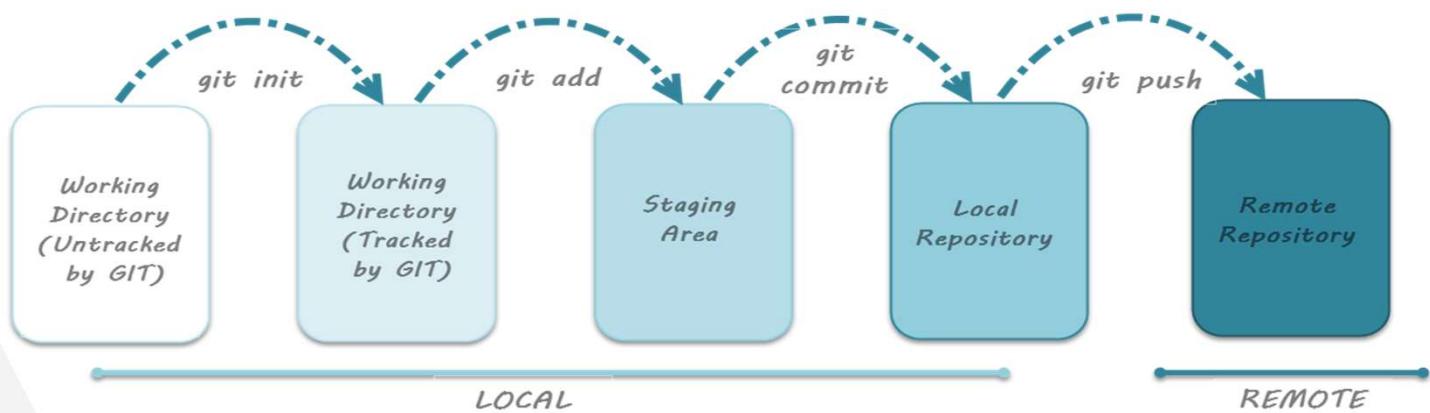
- ▶ Git provides the git config tool to set configuration variables
- ▶ Git stores all global configurations in ".gitconfig" file located in your home directory
- ▶ To set configuration values as global, add the "--global" option
- ▶ Git stores values in the "/etc/gitconfig" file that contains the configuration for every user and repository on the system
- ▶ You can use the "--system" option to apply configuration and ensure you have root rights

User name	\$ git config --global user.name " GilbertFongan"
User email	\$ git config --global user.email gilbert.fongan@yourdomain.com
Notepad++ as Default editor	\$ git config --global core.editor "notepad++.exe -multiInst -nosession"
List global configuration	\$ git config --global --list

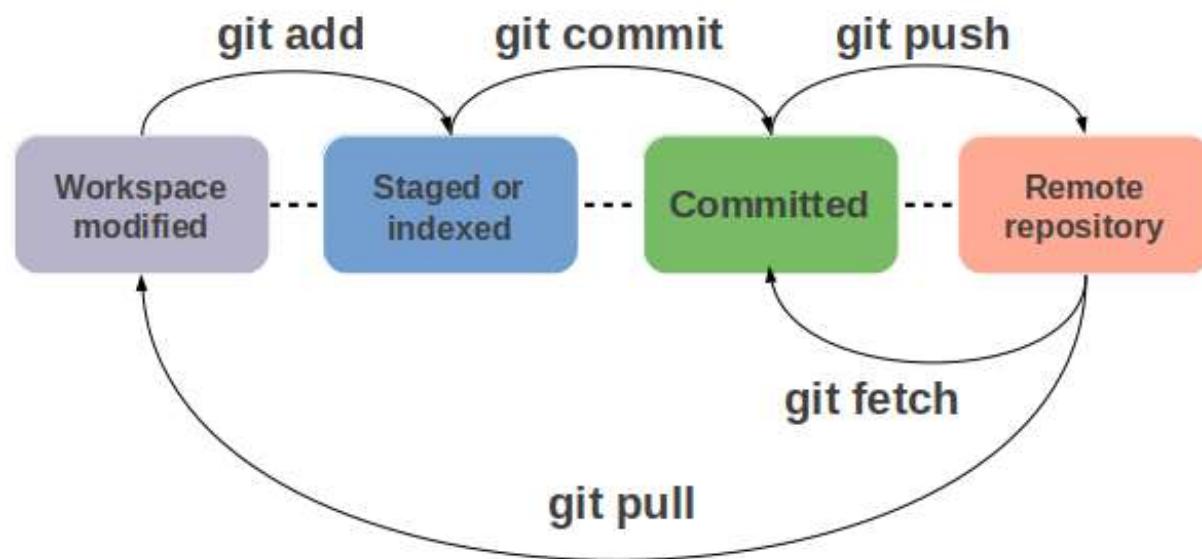
Git terminology

- **Workspace (Working directory)** : contains the files just modified
- **Staging Area (indexed)** : allows you to store selected changes to be committed
- **Local repository (committed/HEAD)** : committed code, ready to be sent to a remote server
- **Remote repository** : Remote server which contains publicly accessible code (Gitlab, GitHub, Bitbucket).

Service hosting Git repos



Git terminology

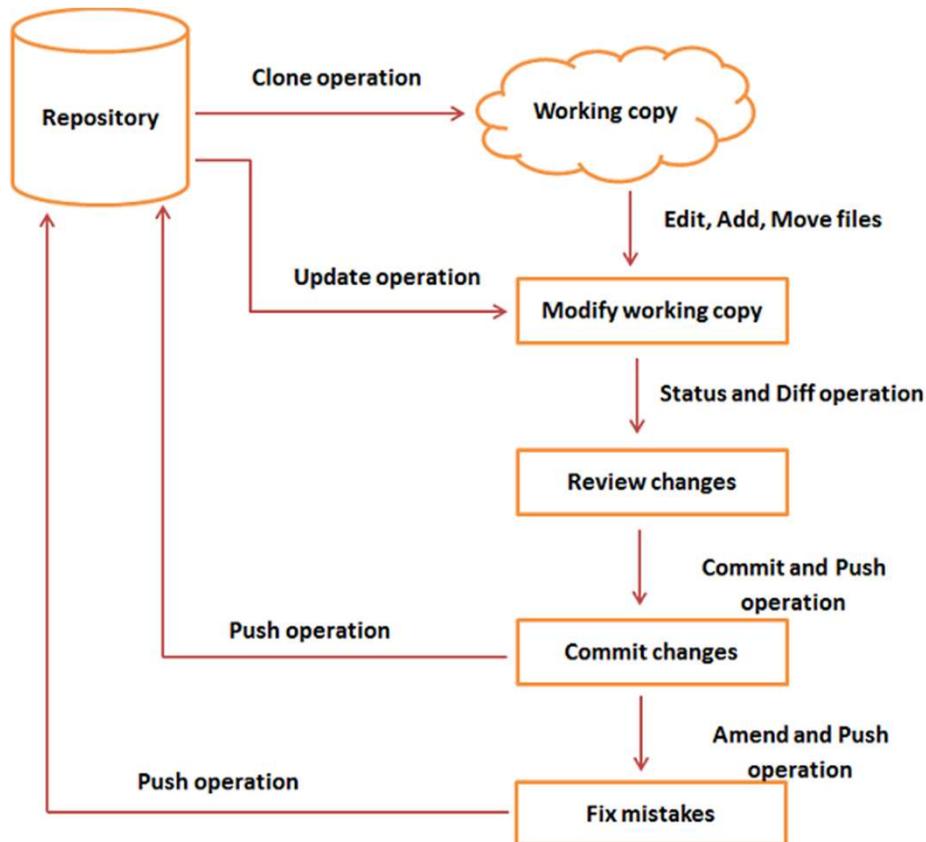


Git Life Cycle

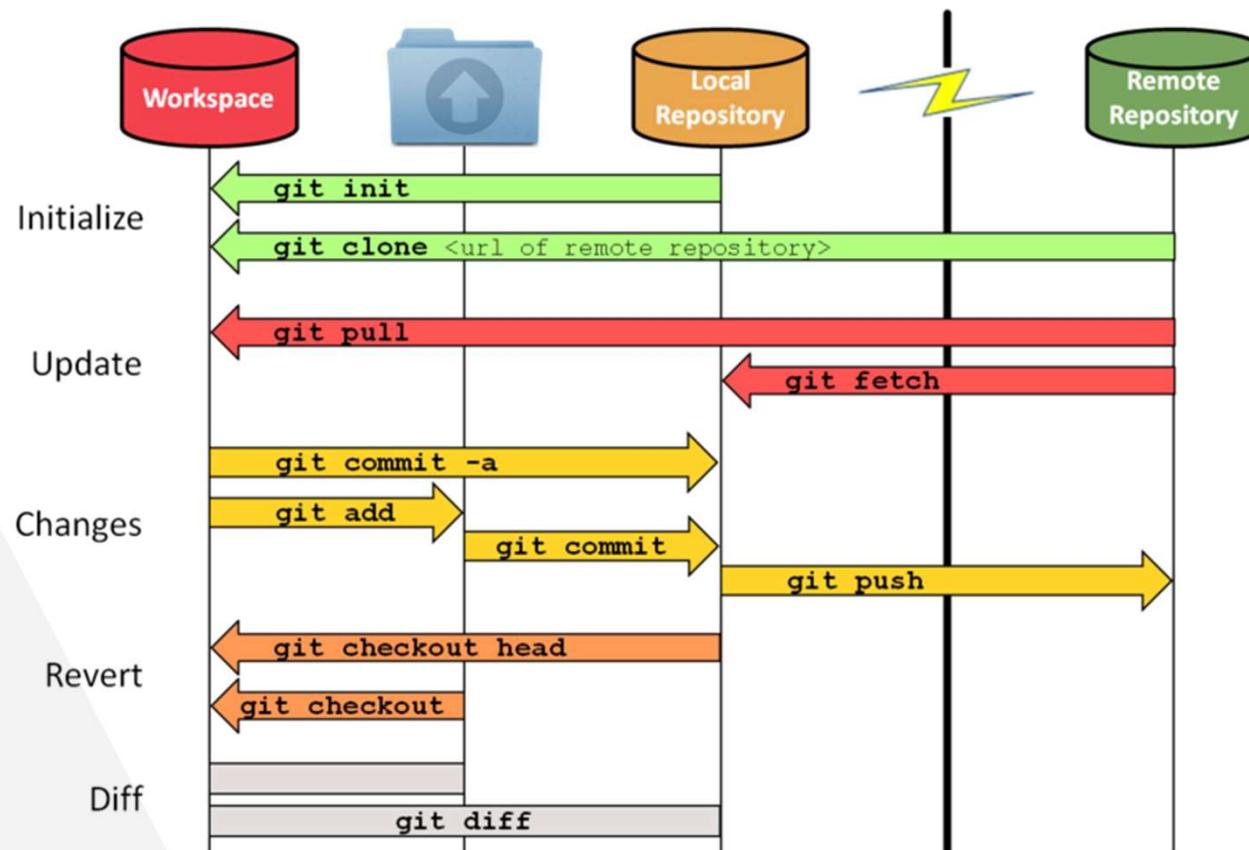
General workflow is as follows :

- ▶ Clone the Git repository as a working copy
- ▶ Modify the working copy by adding/editing files
- ▶ Update the working copy by taking other developer's changes
- ▶ Review the changes before commit
- ▶ Commit changes. If everything is fine, then you push the changes to the repository (remote)
- ▶ After committing, if you realize something is wrong , then you correct the last commit and push the changes to the repository (remote)

Git Life Cycle



Git Workflow



Git command

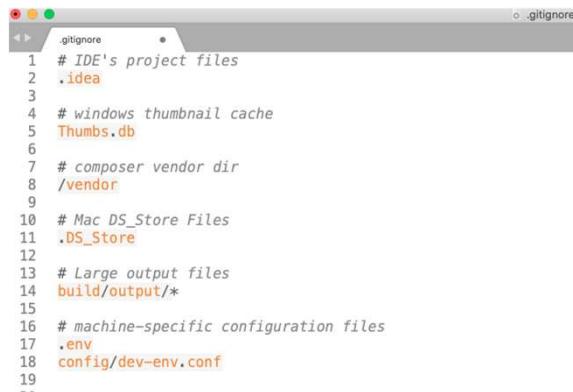
git init	Convert an existing unversioned project(workspace) to git repository or to create a new empty git repository. ".git" subdirectory will be created
git clone	Download an existing git repository to your local computer. <code>git clone -b branch_name <git url></code> : The -b argument lets you specify a specific branch to clone instead of the branch the remote HEAD is pointing to, usually the master branch.
git status	Current branch Files that have differences between <i>Workspace</i> ↔ <i>Staging area</i> (Untracked(new) files and Unstaged changes) Files that have differences between <i>Staging</i> ↔ <i>Local Git Repository</i> (Uncommitted changes)
git add	Add changes in the workspace to the staging area. <code>git add <file-name></code> or <code>git add .</code> to add all files
git commit	Add changes in the staging area to the local Git repository. <code>git commit: Staging area → Local git repository</code> <code>git commit -a: Workspace → Local git repository</code> (Untracked files are not included, only those that have been added with <code>git add</code> at some point). <code>git commit -m 'commit message'</code>
git pull	Update local git repository from the corresponding remote git repository. <code>git pull <remote> <local></code> : Local git repository ← Remote git repository
git push	Add changes in the local git repository to the remote repository. <code>git push <remote> <local></code> : Local git repository → Remote git repository
git branch	List all local branches. <code>git branch -a</code> : List all remote branches as well <code>git branch -d <branch></code> : Delete the specified branch <code>git branch <new branch></code> : Create a new branch
git checkout	Navigate between different branches. <code>git checkout <branch></code> <code>git checkout -b <new branch></code> : Create a new branch from your current branch and switch to it.
git merge	Integrate changes from multiple branches into one. <code>git merge <branch></code>

Git command

git remote	Manage connections to remote repositories. It allows you to show which remotes are currently connected, but also to add new connections or remove existing ones. <code>git remote -v</code> : List all remote connections <code>git remote add <name> <url></code> : Create a new remote connection <code>git remote rm <name></code> : Delete a connection to a remote repository <code>git remote rename <old name> <new name></code> : Rename a remote connection
git fetch	Update local git repository from the corresponding remote git repository. Git fetch does not change your workspace, it keeps the fetched content separate until it is merged. <code>git fetch <remote> <local></code> <code>git checkout <remote>/<local></code> : To view the change <code>git fetch vs git pull</code> : <code>git pull = git fetch + git merge</code>
git stash	Takes your uncommitted changes (staged and unstaged), saves them for later use
git fork	It is a copy of a repository. It allows you to freely experiment with changes without affecting the original project.
git head	HEAD is a reference to the last commit in the currently check-out branch
git revert	Revert some existing commits. Given one or more existing commits, revert the changes that the related patches introduce, and record some new commits that record them. This requires your working tree to be clean (no modifications from the HEAD commit).
git reset	Reset current HEAD to the specified state. <code>git reset HEAD~ --hard</code> to remove the last commit
git log	Display committed snapshots.
git cherry-pick	Sometimes you don't want to merge a whole branch into another, and only need to pick one or two specific commits (Cherry picking).
git diff	Show changes between commits, commit and working tree

Git command

git blame	Show what revision and author last modified each line of a file
git tags	Ability to tag specific points in a repository's history as being important (v1.0, v2.0)
git rebase	Involves moving code to a new base commit or combining a sequence of commits
git squash	To squash or regroup previous commits into one. This is a great way to group certain changes together before sharing them with others.
.gitignore	A text file which tells which files and folders to ignore in a project. A local ".gitignore" file is usually placed in the root directory of a project. You can also create a global ".gitignore" file and any entries in that file will be ignored in all of your Git repositories.



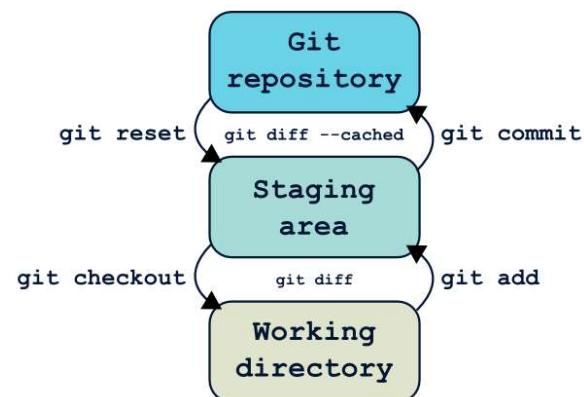
```

.ignite
1 # IDE's project files
2 .idea
3
4 # windows thumbnail cache
5 Thumbs.db
6
7 # composer vendor dir
8 /vendor
9
10 # Mac DS_Store Files
11 .DS_Store
12
13 # Large output files
14 build/output/*
15
16 # machine-specific configuration files
17 .env
18 config/dev-env.conf
19

```

Git / Recovering from mistakes

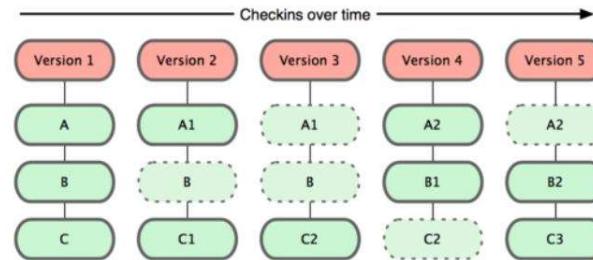
- ▶ To copy a file from the working directory to the staging area, we use ***git add***.
- ▶ To save the staging area in the git repository and create a new commit, we use ***git commit***.
- ▶ To copy a file from the Git repository to the staging area, we use ***git reset***.
- ▶ To copy a file from the staging to the working directory (thus deleting the current modifications), we use ***git checkout***.
- ▶ To view the changes between the working directory and the staging area, we use ***git diff***.
- ▶ To see the changes between the staging area and the last commit, we use ***git diff --cached***.



Git branching / Commits

1

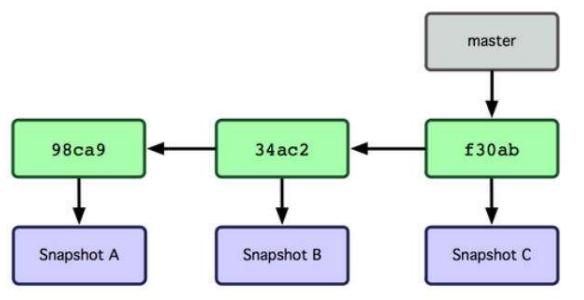
Review: Distributed - Snapshots



Files are stored by SHA-1 hash rather than filename.
Stored in git database in compressed format
Must "checkout" from database into working directory to edit
In this example, files A, B and C are *tracked*

4

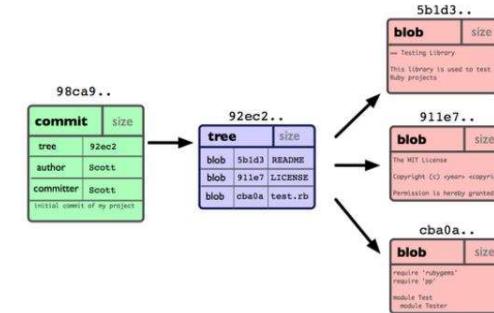
Default branch is master



master moves forward automatically when you commit

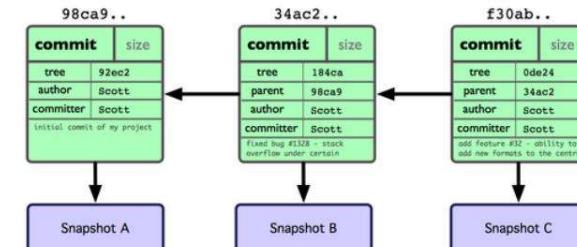
2

Example commit, 3 files



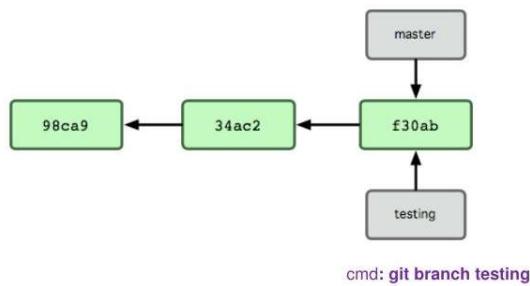
3

After 3 commits



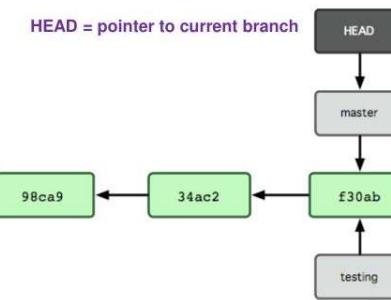
Git branching

1 Add a branch



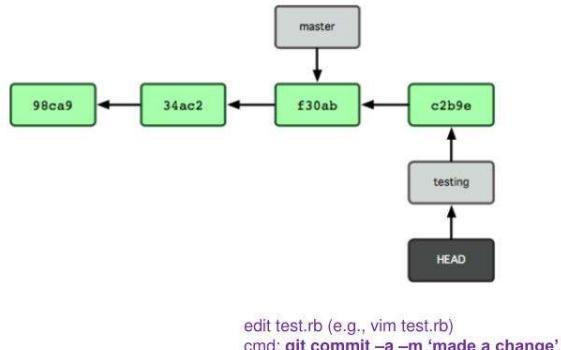
Why is creating a branch in git cheap and quick? Because it's just creating a 40-character SHA-1 checksum of the commit it points to

2 But you're still on master

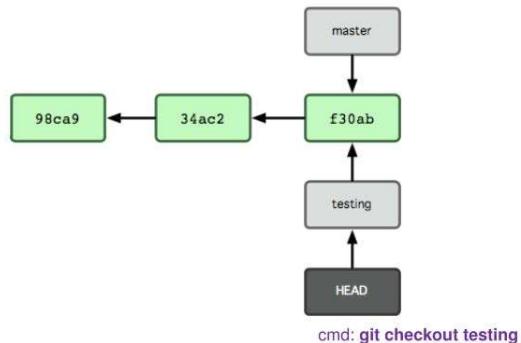


NOTE: concept of HEAD is different from CVS

4 Make changes on that branch



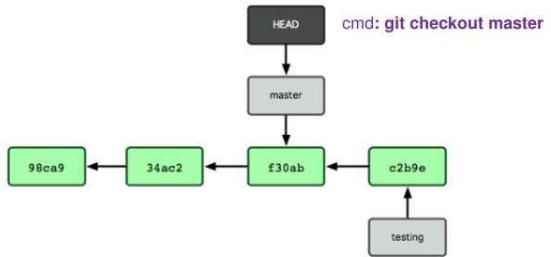
3 Use checkout to change



Git branching

1

Switch back to master

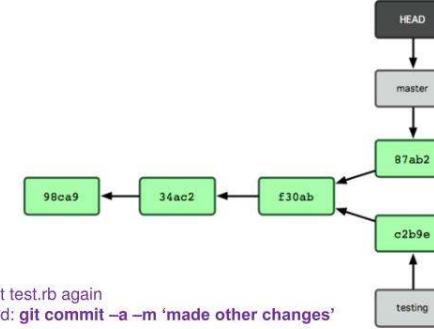


NOTE: This also reverts files in your working directory (e.g., c:\mycode) back to master.

So edit to test.rb no longer in your working directory (but you haven't lost it, as long as you committed before switching – remember it will be in your local database. But don't delete .git!)

2

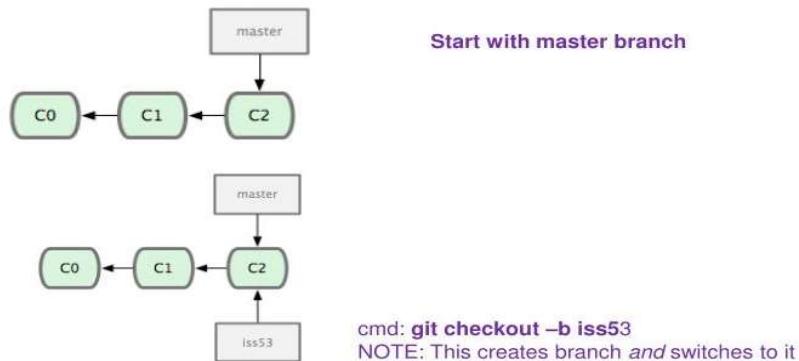
Change master branch



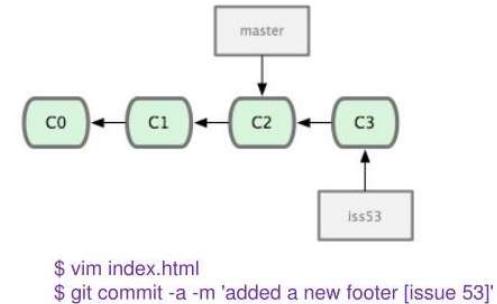
edit test.rb again
cmd: `git commit -a -m 'made other changes'`

Git basic merging

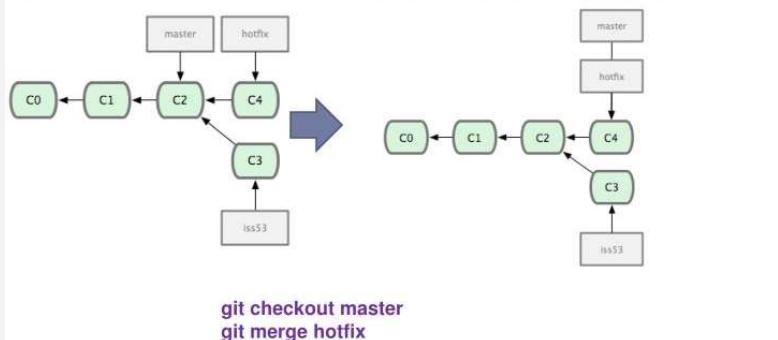
1 A branch/merge example



2 Update branch

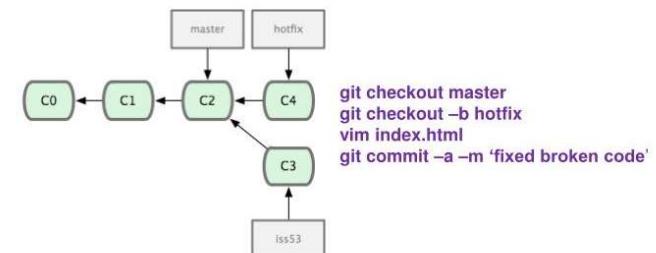


4 Now merge hotfix and master



3

Need to switch to an urgent fix

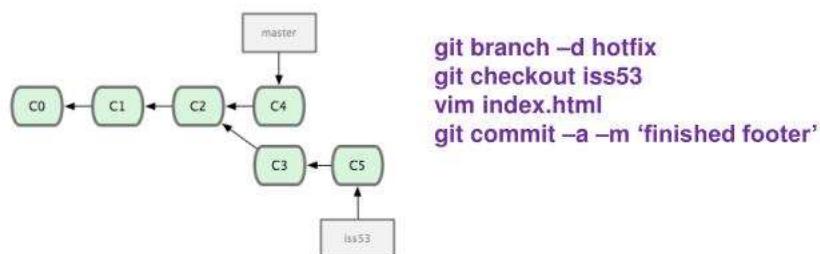


NOTE: git won't let you switch to master if you have uncommitted changes that conflict with the code you're checking out. So you should have a clean slate before you switch (stash and amend can deal with this – later topics)

Git basic merging

1

A little cleanup, then return to issue 53



```

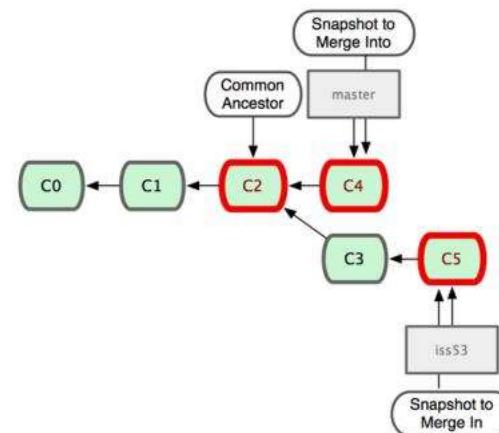
git branch -d hotfix
git checkout iss53
vim index.html
git commit -a -m 'finished footer'
  
```

Be careful with branch `-d`. OK in this example, just a duplicate pointer. May not always be the case.

Note that work done on hotfix is not part of iss53 branch.

2

Basic merge



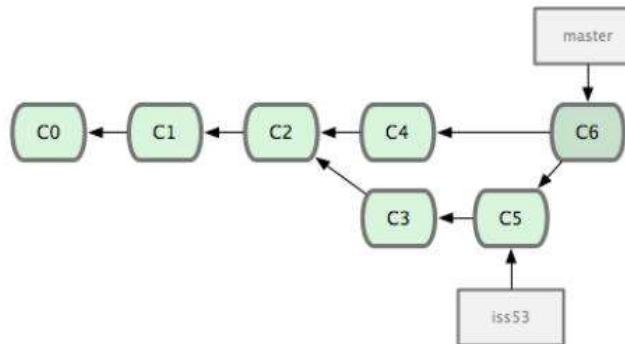
```

$ git checkout master
$ git merge iss53
Merge made by recursive.
 README | 1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
  
```

git identifies the best common ancestor to use for merge

Git basic merging

Basic Merge result



master now includes hotfix and iss53

Remove iss53 if you want: `git branch -d iss53`



Rebase and merge

Rebasing and **merging** are both designed to integrate changes from one branch into another branch but in different ways.

- ▶ **Merge** is the result of the combination of commits in feature branch
- ▶ **Rebase** add all the changes in feature branch starting from the last commit of the master branch

Rebasing a feature branch into master leads to move the base of the feature branch to master branch's ending point.

Merging takes the contents of the feature branch and integrates it with the master branch. As a result, only the master branch is changed. The feature branch history remains same.
Merging adds a new commit to your history.

Rebase and merge

What is a merge?

A process that unifies the work done in two branches



What is a fast-forward merge?

It will just shift the **master** HEAD



What is squash on merge?

It will compact feature commits into one before merging



What is a rebase?

It's a way to replay commits, one by one, on top of a branch



MODULE I-4 : Continuous Integration and Continuous Delivery

PLAN

- TRADITIONAL INTEGRATION:
- CONTINUOUS INTEGRATION:
- SOFTWARE TESTING :
 - Manuel & automatic testing
 - White Box testing
 - Black Box testing
 - Tools
- CONTINUOUS DELIVERY
- CONTINUOUS DEPLOYMENT
- CI/CD DEPLOYMENT
 - Blue & Green Deployment
 - Canary deployment
- JENKINS
 - Workflow
 - Build stages
 - Architecture Master & Slave
 - Jenkins Declarative Pipeline



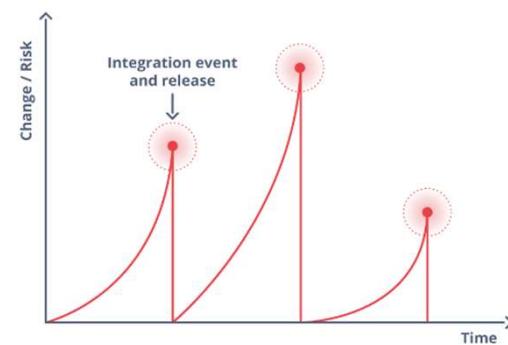
Traditional Integration

In **Traditional Integration** or/**software development cycle** :

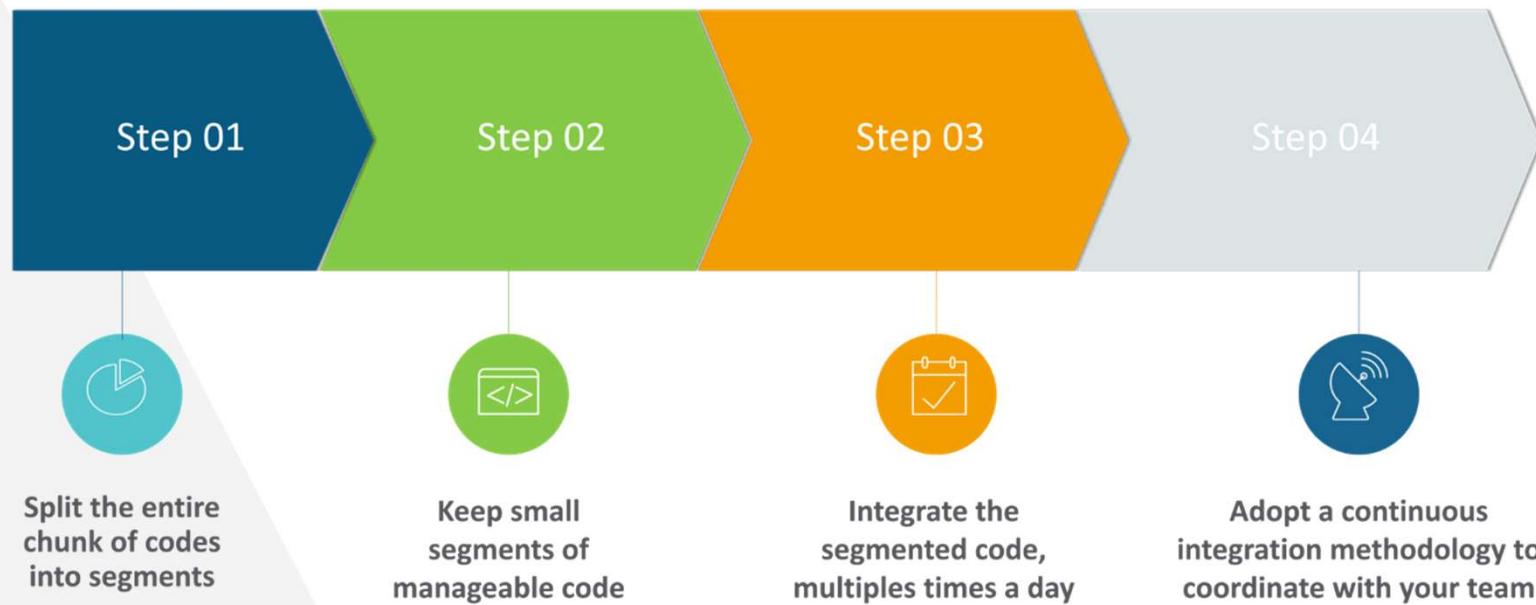
- ▶ Each developer gets a copy of the code from the **central repository**
- ▶ All developers begin at the same starting point and work on it
- ▶ Each developer makes progress by working on their own team
- ▶ Each developer add methods and functions, shaping the code to meet their needs
- ▶ Meanwhile, the other developers and teams continue working on their own tasks, solving the problems they have been assigned

The main factors that can make these problems escalate

- ▶ The size of the team working on the project
- ▶ The amount of time passed since the developer got latest version of the code from the central repository



Solution for problems faced in Traditional Integration



Benefits of CI/CD

Reduction of delivery risk

Better visibility on change

Increased efficiency and delivery options

The process is known

Opens up more avenues for review

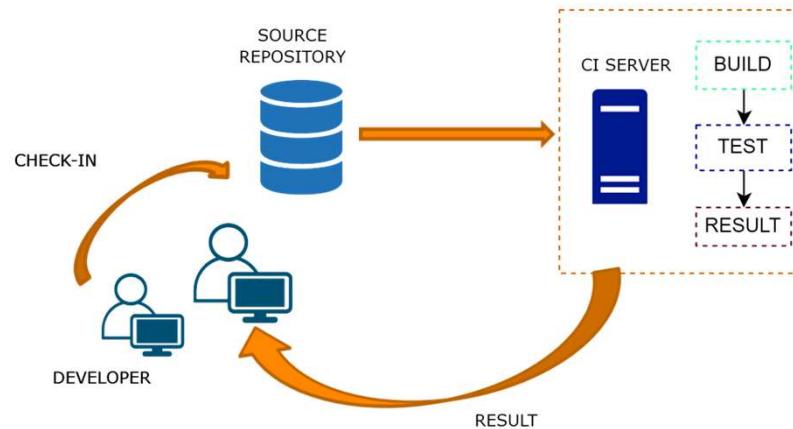
Enhanced learning from failure



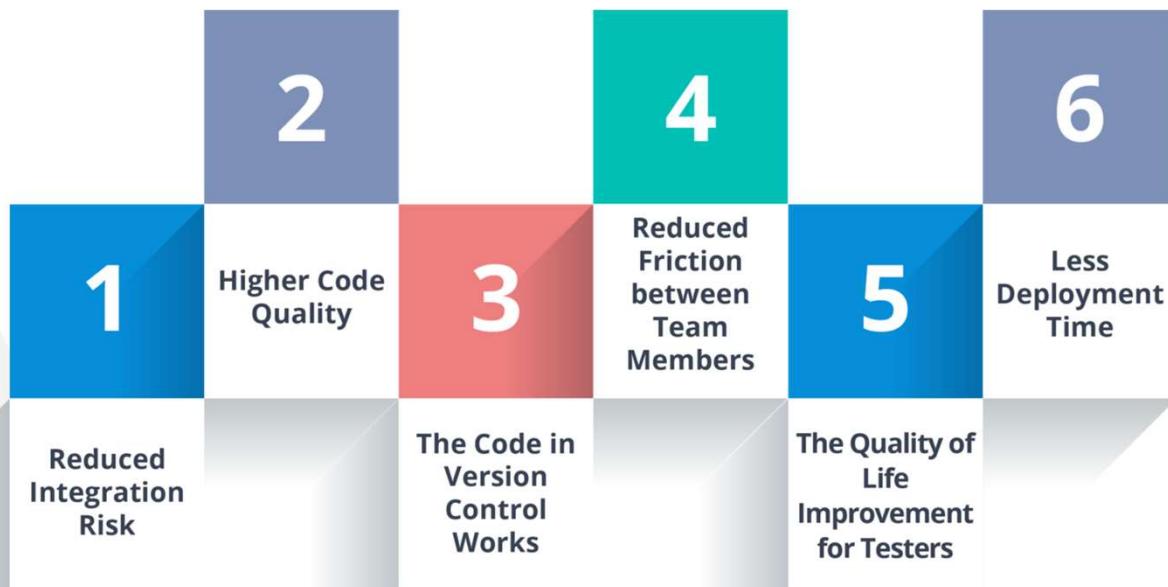
Continuous Integration

Continuous Integration :

- ▶ Software development practice
- ▶ Developers integrate code into a shared repository frequently
- ▶ Each integration is verified by an automated build and automated tests to detect integration errors as quickly as possible
- ▶ This approach leads significantly to develop cohesive software more rapidly



Benefits of Continuous Integration



Software Testing

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free. Some prefer saying **Software testing** definition as a **White Box, Black Box Testing** and **Grey Box Testing**.

Here are the benefits of using software testing :

- ▶ Cost-Effective
- ▶ Security
- ▶ Product quality
- ▶ Customer Satisfaction



Software Testing / Types

In order to better understand the concepts of Continuous Delivery and Continuous Deployment , we need to understand what is the Software Testing and what are its different types :

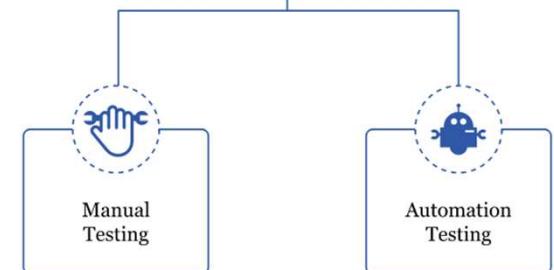
- ▶ **Manual Testing** : Testing software only by human intervention. It may include detailed step-by-step test cases for testing periods.

Advantages : Cost-Effective, Nothing can beat the human eye, User experience modification, Flexibility.

- ▶ **Automation Testing** : Testing by using automation tools.

Advantages : Scheduling, Regression testing is easy, Reusability of test scripts, Saves time.

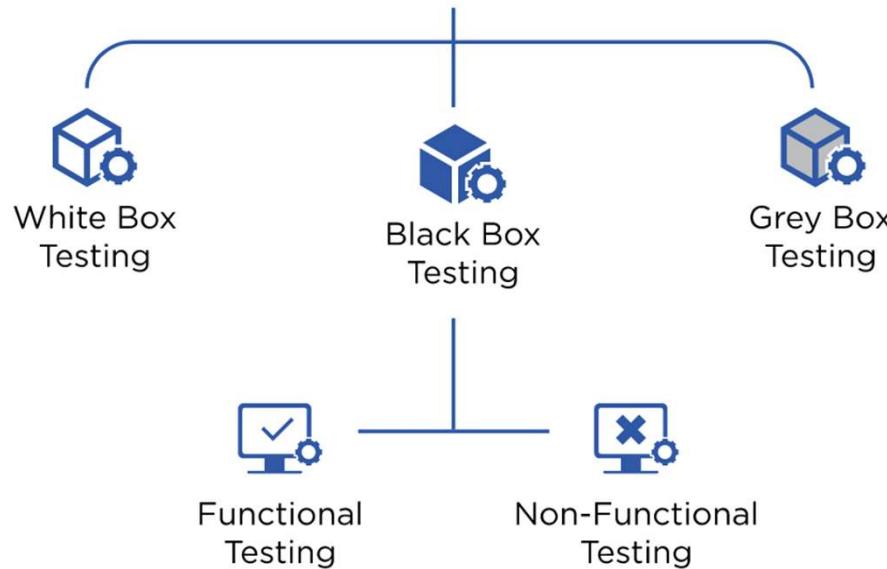
Types Of Software Testing



Software Testing / Approaches

There are mainly three approaches to Software Testing

Approaches to Software Testing



Software Testing / Approaches

Testing	Advantages	Disadvantages
White Box Testing (Clear box testing/Glass box testing)	<ul style="list-style-type: none"> - Performed at the initial stages - More in depth - Find hidden defects - Helps in code optimization 	<ul style="list-style-type: none"> - Complicated - Requires highly skilled resources - Tools may not be readily available
Black Box Testing (Behavioral testing)	<ul style="list-style-type: none"> - Exposes inconsistencies in specifications - No need to understand programming 	<ul style="list-style-type: none"> - Test may be tough to design - Many bugs can go undetected
Grey Box Testing (Mixture of white-box and black-box)	<ul style="list-style-type: none"> - Develop more intelligent tests - Clear goals while testing <p>Overall quality of the software is enhanced</p>	<ul style="list-style-type: none"> - Complicated - Hard to detect bugs

Black box testing

Black Box Testing is further classified into two categories :

- ▶ **Functional Testing** : verify that there are no gaps between developed features/functions and required features/functions.

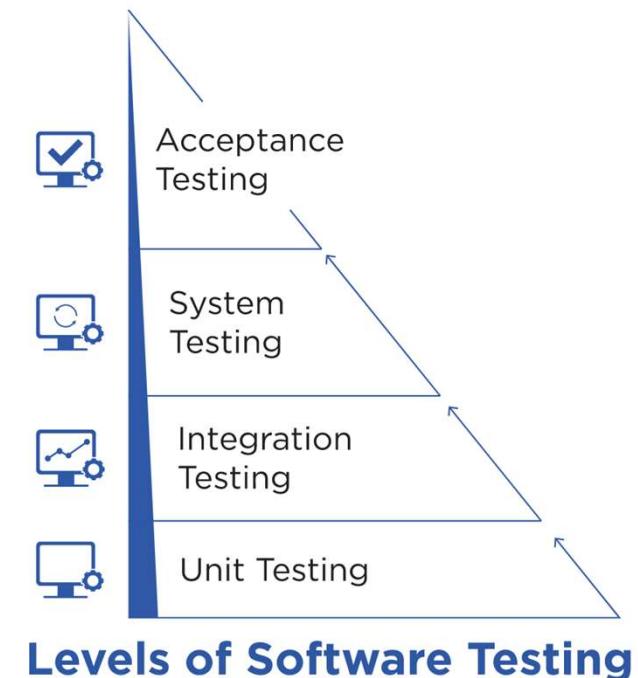
Unit Testing, Integration Testing , System Testing , Acceptance Testing.

- ▶ **Non-Functional Testing** : Focus on the non-functional parts of the software like:

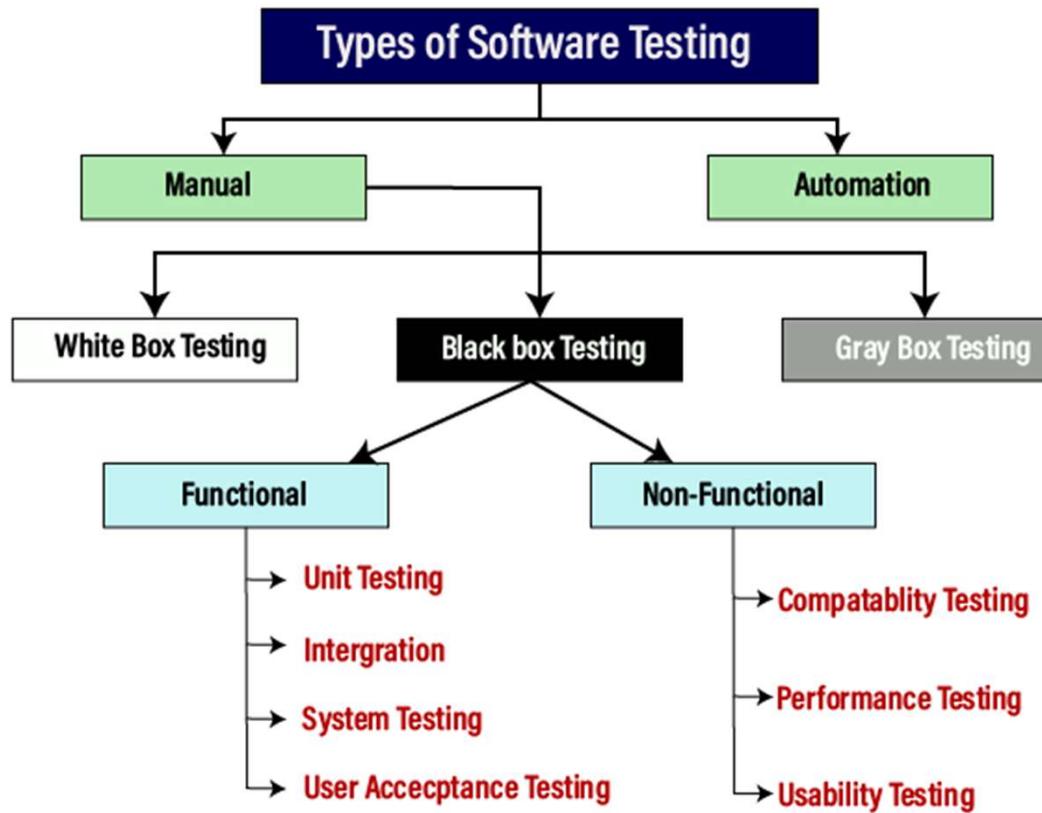
Memory leaks, Performance, Load, Scalability, Volume, Usability.

Software Testing / Levels

- ▶ **Unit Testing :** tiniest testable component of the software. The aim is to ensure the proper functioning of each unit.
- ▶ **Integration Testing :** Individual units are grouped for testing. The aim is to detect errors in the integrated unit's interaction.
- ▶ **System Testing :** The integrated software is tested wholly. The aim is to assess the conformity of the software with the established requirements and end-to-end testing
- ▶ **Acceptance Testing :** Software is assessed for acceptability. It is verified against the requirements to ensure it is adequate for delivery.



Software Testing / Resume



Software Testing / Tools

Today, many software testing tools are of great importance, especially for automation testing.



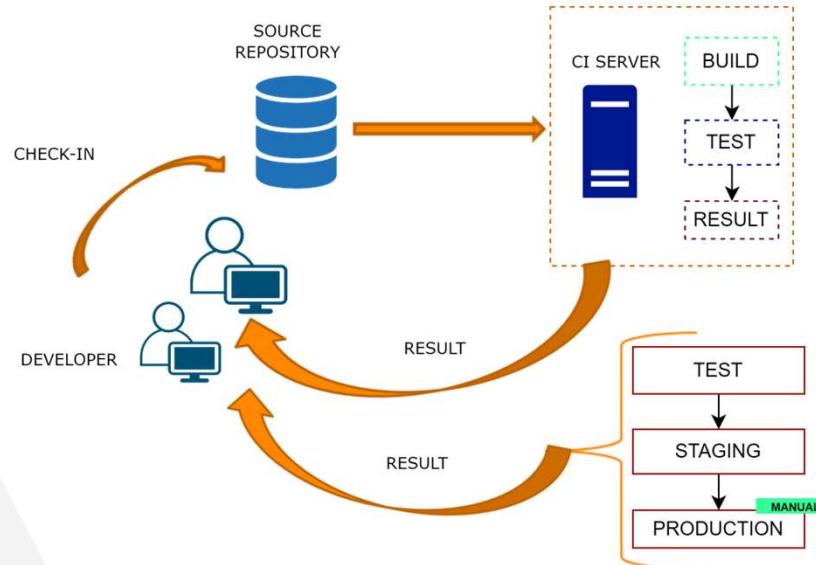
BDD
Cucumber



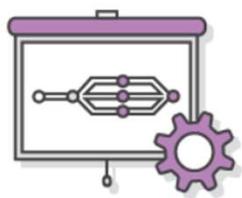
Continuous Delivery

Continuous Delivery:

- ▶ Software development practice where code changes are automatically prepared for a release to production.
- ▶ Expands upon continuous integration by deploying all code changes to a testing environment after the build stage.
- ▶ Developers will always have a deployment-ready build artifact that has passed through a standardized test process



Benefits Continuous Delivery



Automate the Software Release Process



Improve Developer Productivity



Find and Address Bugs Quicker

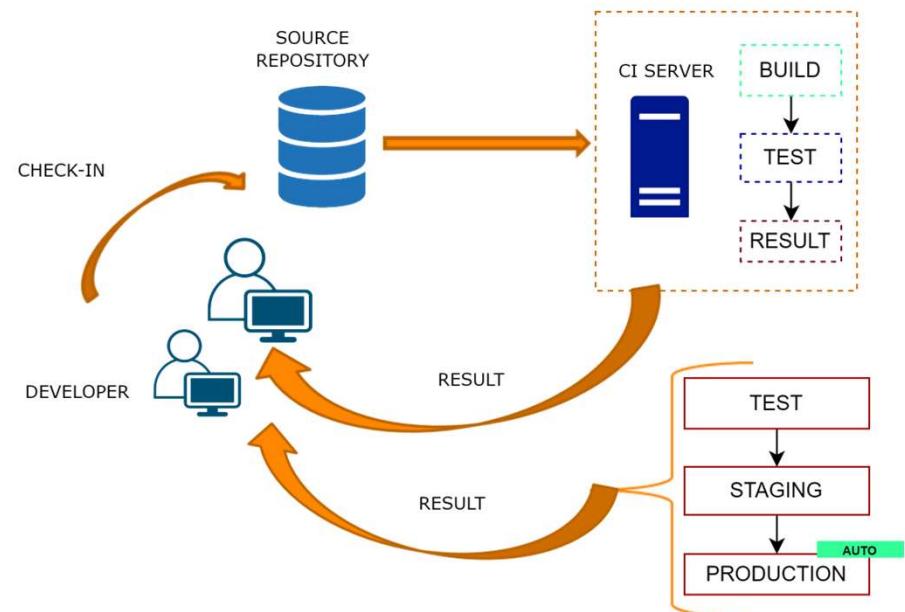


Deliver Updates Faster

Continuous Deployment

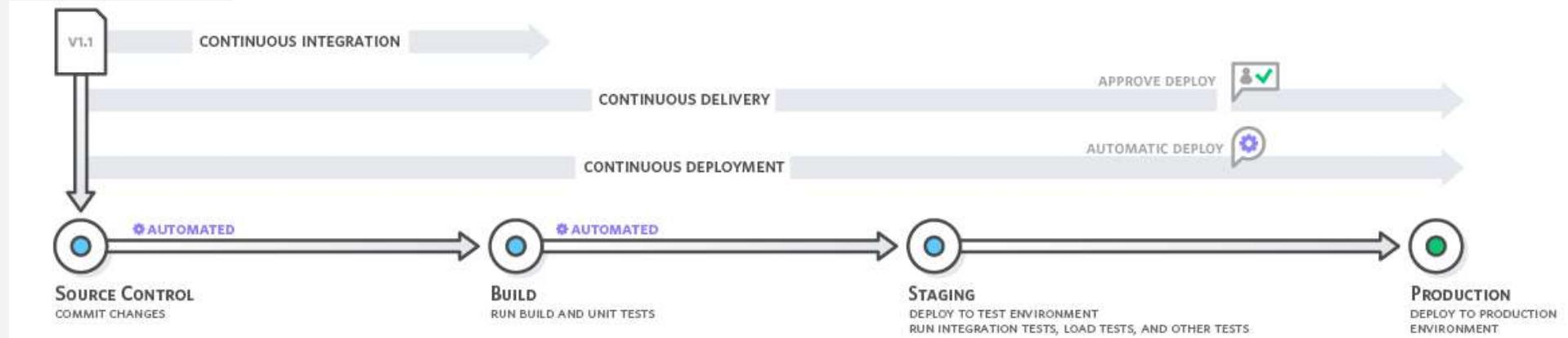
Continuous Deployment:

- ▶ It eliminates the human safeguards against unproven code in live software.
- ▶ It should only be implemented when the development and IT teams rigorously adhere to **production-ready development practices**

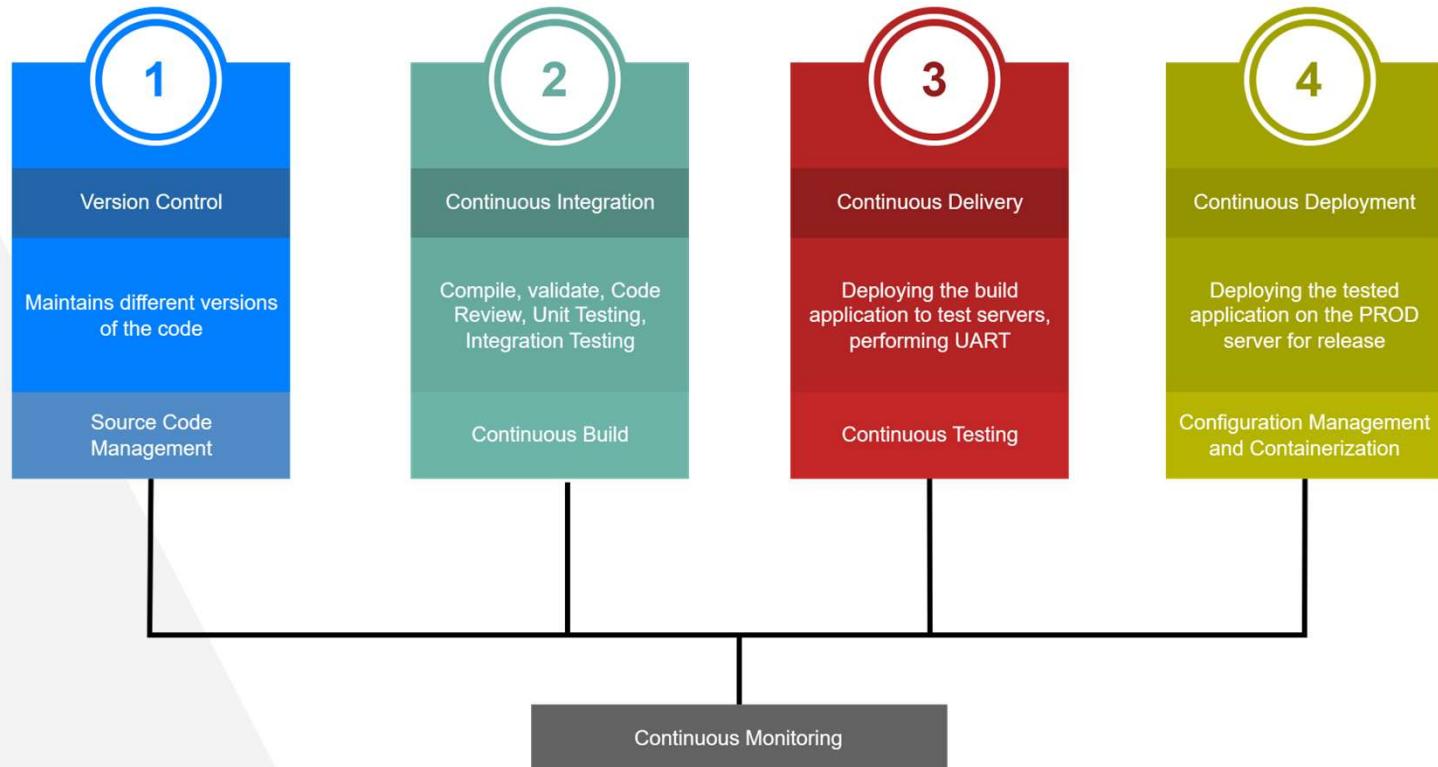


Continuous Delivery vs Continuous Deployment

- ▶ With Continuous Delivery, every code change is built, tested and then pushed to a **non-production testing** or **staging environment**
- ▶ There can be multiple, parallel test stages before a production deployment
- ▶ The difference between **Continuous Delivery** and **Continuous Deployment** is the presence of a **manual approval** to update to production.



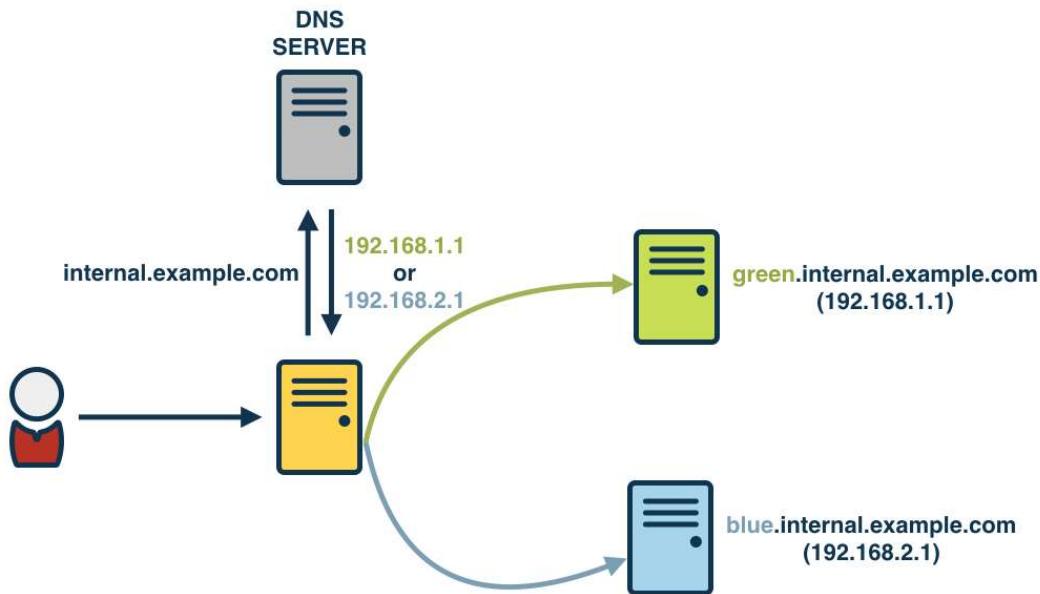
CI/CD DevOps Stages



CI/CD Deployment

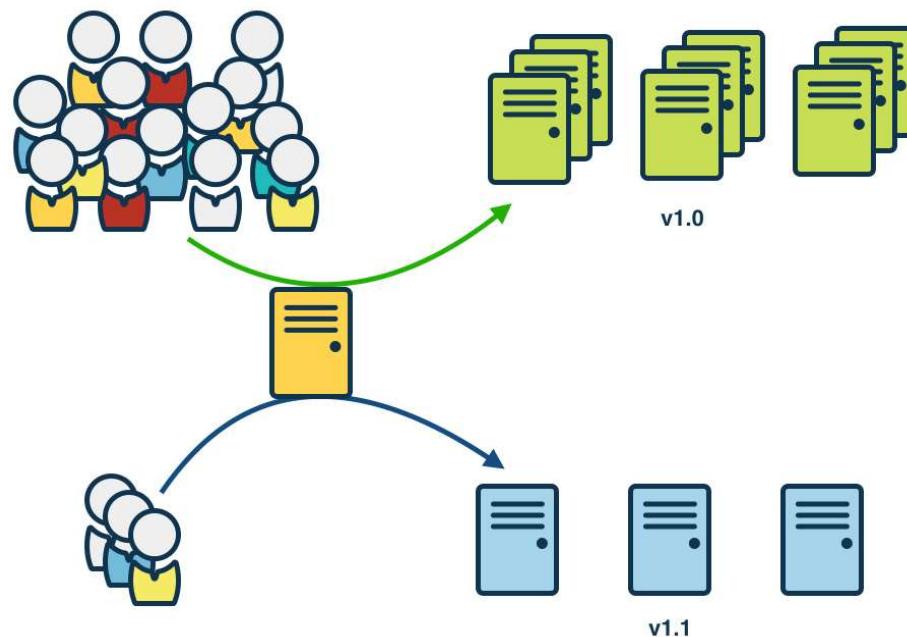
Blue / Green Deployment is a technique for deployments where the existing running deployment is left in place. A new version of the application is installed in parallel with the existing version.

When the new version is ready, cut over to the new version by changing the load balancer configuration.



CI/CD Deployment

Canary Deployment are like Blue/Green, although only a small amount of the servers are upgraded. Then, using a **cookie or similar**, a fraction of users are directed to the new version.



CI/CD Tools



AWS CodePipeline



Azure DevOps



Jenkins



GitLab



Travis CI

Jenkins

Jenkins :

- ▶ Open-source Continuous Integration Server
- ▶ Written in Java with plugins built for CI purpose
- ▶ Easy to install and use
- ▶ Multi-technology and Multi-platform
- ▶ Widely used, extensible and free.
- ▶ Used to manually, periodically, or automatically build software development projects



Why Jenkins?

Easy to install :

- ▶ Download one file -> jenkins.war
- ▶ Run one command -> java-jar jenkins.war

Easy to use :

- ▶ Create a new job-checkout and build a small project
- ▶ Checking a change-watch it build
- ▶ Create/fix a test – Watch it build and run/checkin and watch it pass



Jenkins

Multi-technology :

- ▶ Build C, Java, C#, Python, Perl, SQL
- ▶ Test with JUnit, NUnit, MSTest

Great extensibility :

- ▶ Support different VCS
- ▶ Code quality metrics , Build notifiers and UI customization

Jenkins Interface

Offers many types of projects :

- ▶ Free style
- ▶ Building a Maven Project
- ▶ Pipeline and multibranch pipeline (most used for Git projects)

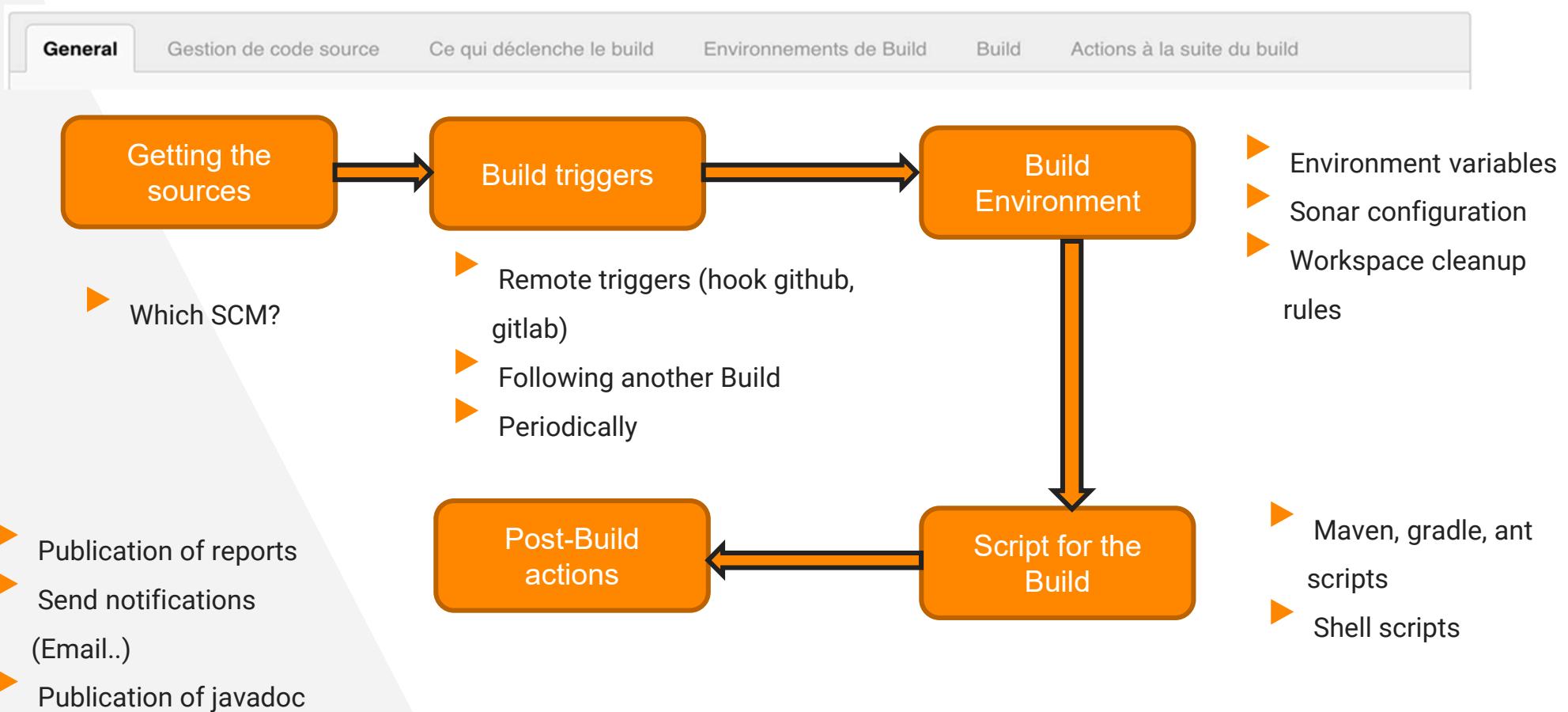
Ability to clone an existing project.

The screenshot shows the Jenkins 'New Item' creation dialog. At the top, there is a text input field labeled 'Saisissez un nom' (Enter a name) with a red error message below it: 'Ce champ ne peut pas être vide. Veuillez saisir un nom valide et appuyer sur OK.' (This field cannot be empty. Please enter a valid name and press OK.). Below the input field, there are five project type options, each with an icon and a brief description:

- Construire un projet free-style**: Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.
- Construire un projet maven**: Construit un projet avec maven. Jenkins utilise directement vos fichiers POM et diminue radicalement l'effort de configuration. Cette fonctionnalité est encore en bêta mais elle est disponible afin d'obtenir vos retours.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Construire un projet multi-configuration**: Adapté aux projets qui nécessitent un grand nombre de configurations différentes, comme des environnements de test multiples, des binaires spécifiques à une plateforme, etc.
- Tâche externe**: Ce type de tâche permet de suivre l'exécution d'un process lancé en dehors de Jenkins, et ce, même sur une machine distante. Cela vous permet d'utiliser Jenkins comme tableau de bord de vos systèmes automatisés existant.

At the bottom of the dialog, there are two buttons: 'OK' and 'Annuler' (Cancel).

Jenkins / Building stages



Jenkins / User interface

The screenshot shows the Jenkins web interface. On the left, there's a sidebar with links: New Item, People, Build History, Manage Jenkins, and Credentials. Below these are two collapsed sections: Build Queue and Build Executor Status, each showing 'No builds in the queue.' and '1 idle' or '2 idle' executors respectively. The main area is titled 'Configuration Panel'. It features a search bar at the top right with 'ENABLE AUTO REFRESH' and 'add description' buttons. A table below is labeled 'Job Table' with columns: S, W, Name (sorted by Name), Last Success, Last Failure, and Last Duration. One job entry is shown: build helloworld, with a grey icon, a yellow sun icon, and the status 'N/A' for all metrics. At the bottom of the page, there's a footer with a link to localize the page, a timestamp 'Page generated: Aug 14, 2015 1:25:52 PM', and links for 'REST API' and 'Jenkins ver. 1.617'.

Configuration Panel

Header

Job Table

Build Queue & Executor Status Panel

Legend: RSS for all, RSS for failures, RSS for just latest builds

S	W	Name ↓	Last Success	Last Failure	Last Duration
		build helloworld	N/A	N/A	N/A

Icon: S M L

No builds in the queue.

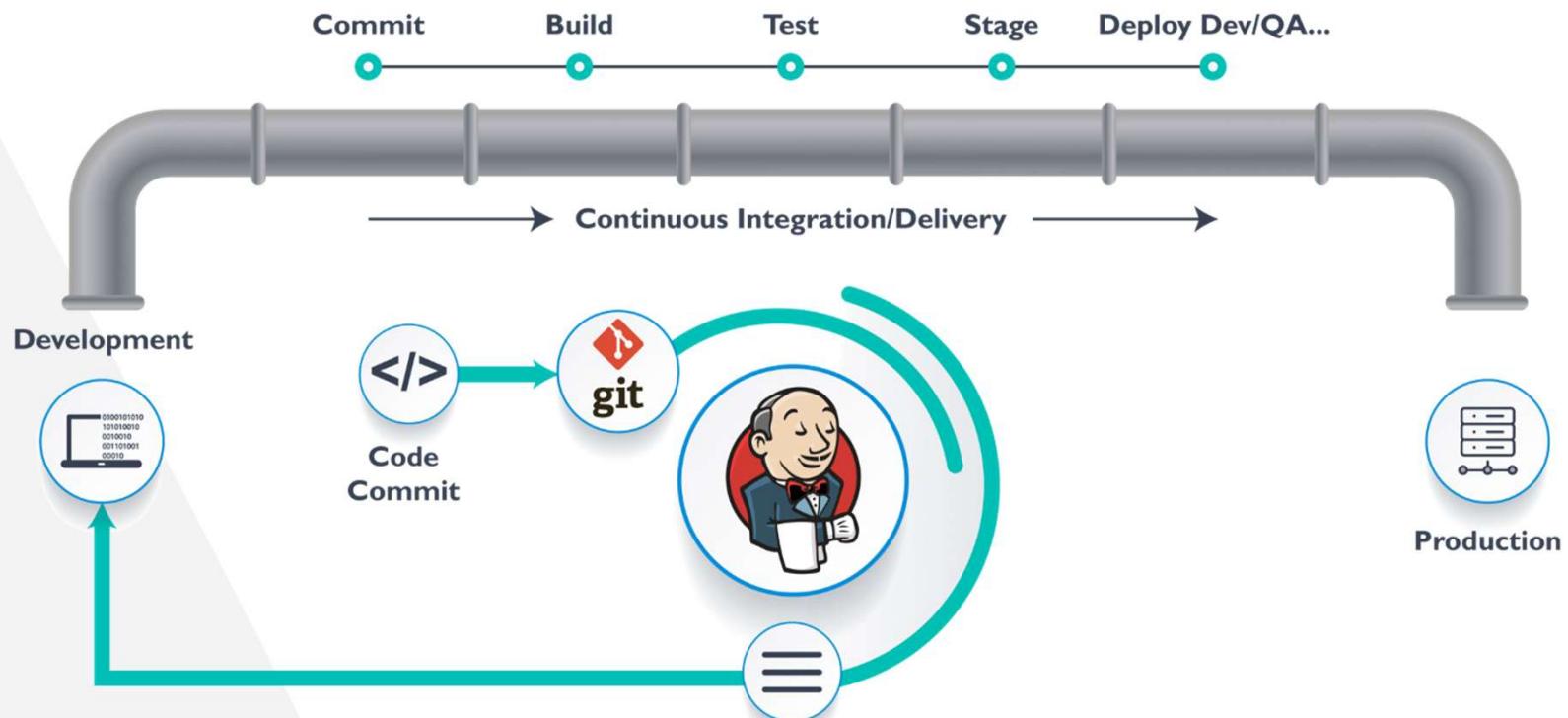
Build Executor Status

1 idle
2 idle

Help us localize this page

Page generated: Aug 14, 2015 1:25:52 PM REST API Jenkins ver. 1.617

Jenkins workflow



Jenkins / Getting the sources

Gestion de code source

Aucune
 Git

Repositories

Repository URL: `https://github.com/nicolas59/epsi-jee-eboutique.git`

Credentials: - aucun -

Branches to build

Branch Specifier (blank for 'any'): `*/master`

Add Branch

Navigateur de la base de code: (Auto)

Additional Behaviours:

Subversion

On peut ajouter des behaviours

► Different types of sources
(Git, Subversion, CVS)

Possibility to add behaviors

Jenkins / Building stage

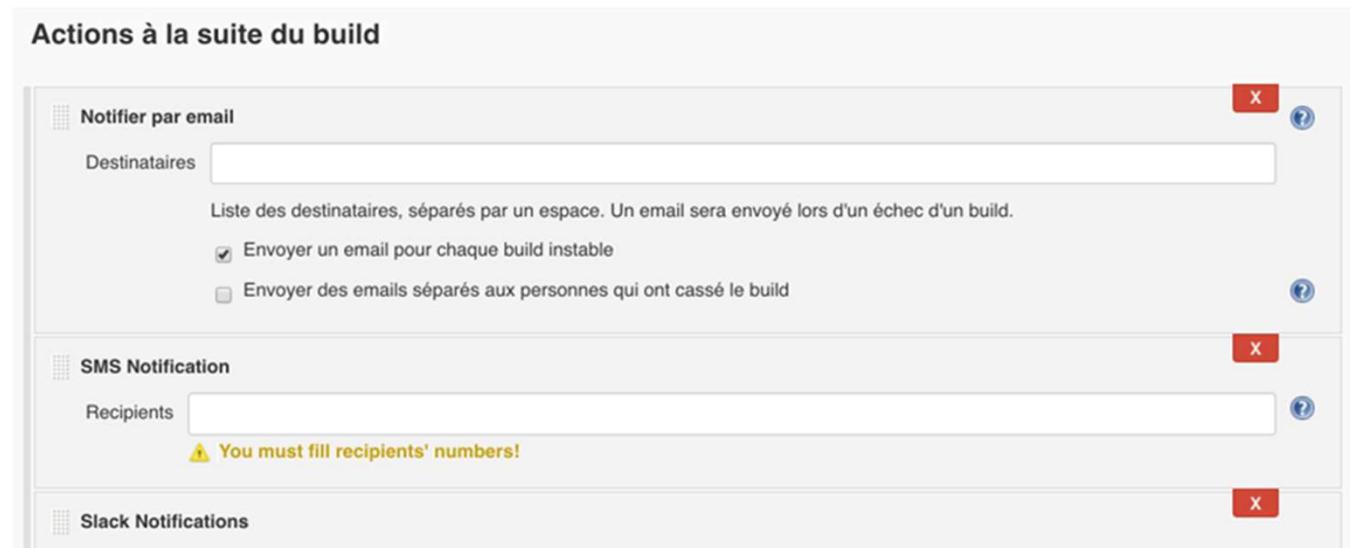
- ▶ Many build tools for several languages
- ▶ Java : Maven, Gradle, Ant
- ▶ .Net : MsBuild
- ▶ iOS
- ▶ Shell Scripts



Jenkins / Post-Build stage

Multiple notifications mechanisms

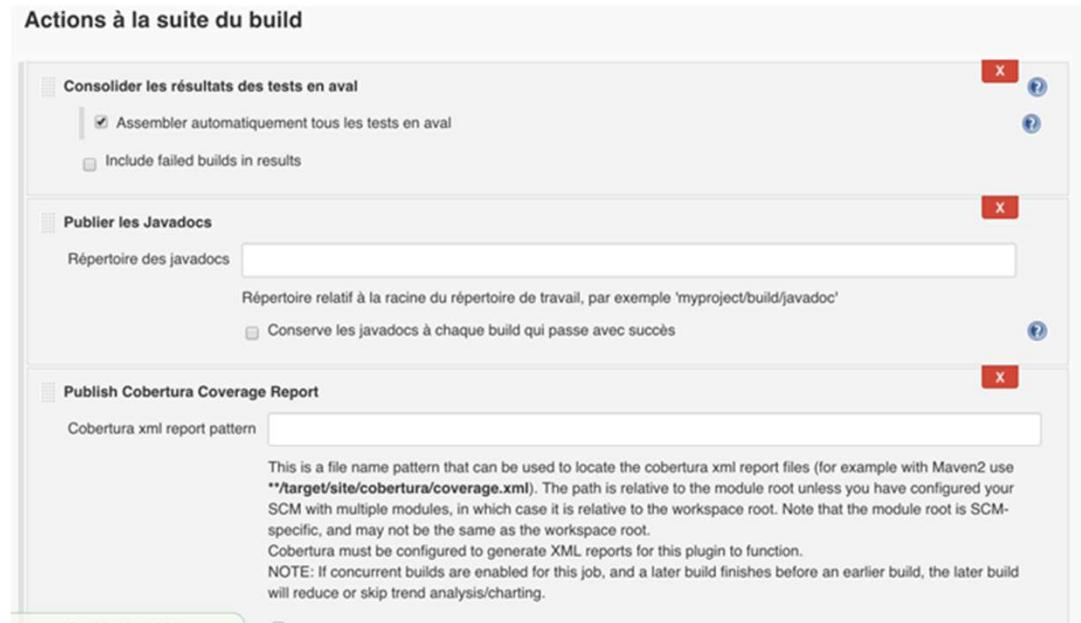
- ▶ Slack
- ▶ SMS
- ▶ Email



Jenkins / Post-Build stage

Many types of publishable reports

- ▶ **Reports on statistical analysis of the code** (Checkstyle, PMD, Findbug, ...)
- ▶ **Unit test execution and coverage report** (Junit, Cobertura, TestNG, JaCoCo...)
- ▶ **JavaDoc publication**



Jenkins /Build result

Jenkins > tp2-persistance > rafraîchissement automatique

[Retour au tableau de bord](#) [État](#) [Modifications](#) [Répertoire de travail](#) [Lancer un build avec des paramètres](#) [Supprimer Projet](#) [Configurer](#) [Rename](#) [Coverage Trend](#)

Projet tp2-persistance

[Espace de travail](#) [Changements récents](#) [Derniers résultats des tests \(aucune erreur\)](#)

[Ajouter une description](#) [Désactiver le projet](#)

Résultats des tests

Build	Count
#21	10
#22	10
#23	10
#24	10
#25	10

(montrer les échecs seulement) agrandir

Code Coverage Trend

Build	Coverage
#21	~185
#25	~200

enlarge

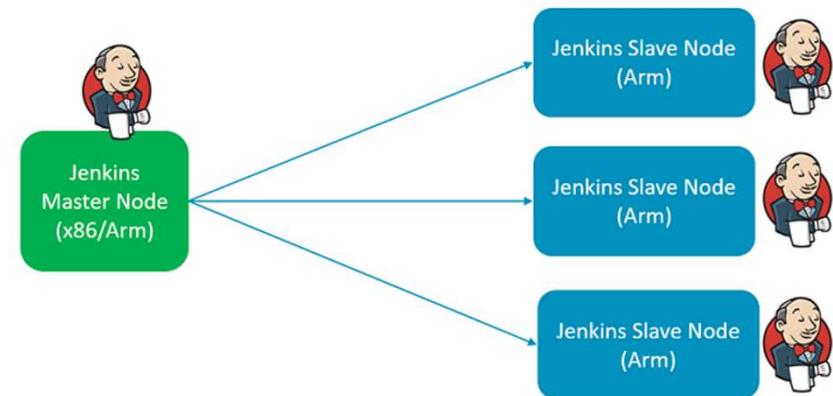
Liens permanents

- Dernier build (#25). il y a 5 mn 19 s
- Dernier build stable (#25). il y a 5 mn 19 s
- Dernier build avec succès (#25). il y a 5 mn 19 s
- Dernier build en échec (#24). il y a 6 mn 41 s
- Dernier build non réussi (#24). il y a 6 mn 41 s
- Last completed build (#25). il y a 5 mn 19 s

[RSS des builds](#) [RSS des échecs](#)

Jenkins / Architecture

- ▶ Jenkins supports **Master-Slave** architecture
- ▶ Jenkins can run the same test case on **different environments in parallel** using Jenkins Distributed Builds.
- ▶ Known as Jenkins **Distributed Builds**.
- ▶ Which in turn helps to achieve the desired results **quickly**.
- ▶ All the **job results** are collected and combined on the **Master node for monitoring**.



Jenkins / Architecture

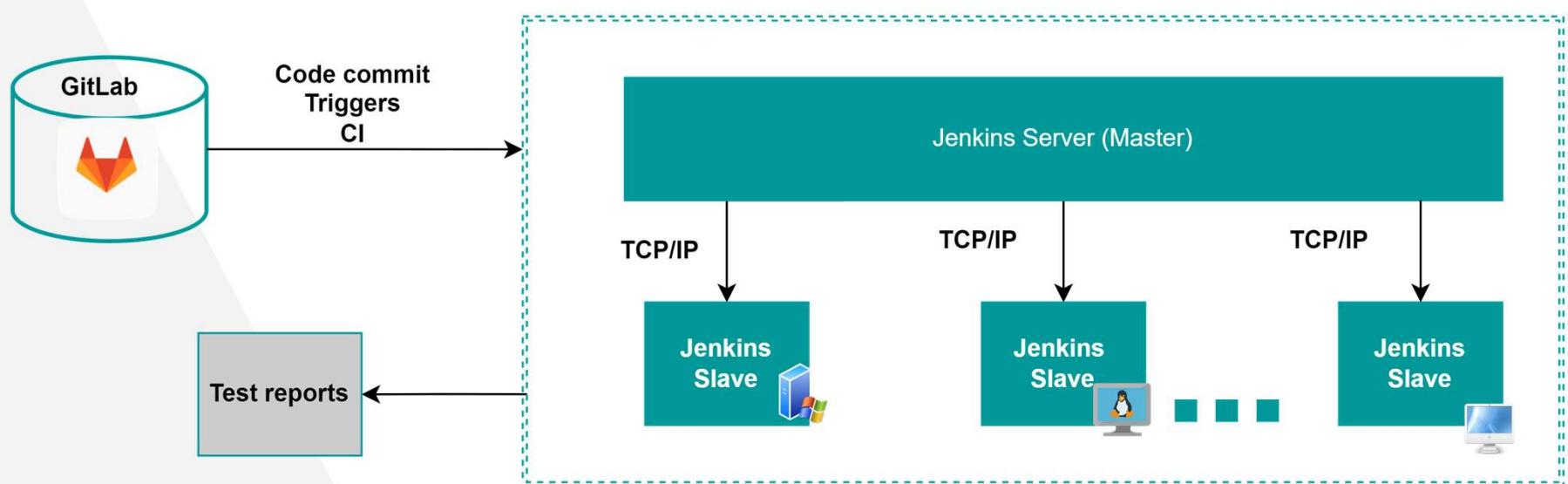
Jenkins Master :

- Scheduling and execute build jobs directly
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking the online and offline as required)
- Recording and presenting the build results.

Jenkins Slave :

- It hears request from the Master Instance
- Slaves can run a variety of Operating Systems.
- The job of a slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- A project can be configured to always run on a particular Slave machine/type or simply let Jenkins pick the next available Slave

Jenkins / Architecture example



Jenkins / Setup Master and Slaves

Go to the Manage Jenkins section and scroll down to the section of Manage Nodes



Jenkins / Setup Master and Slaves

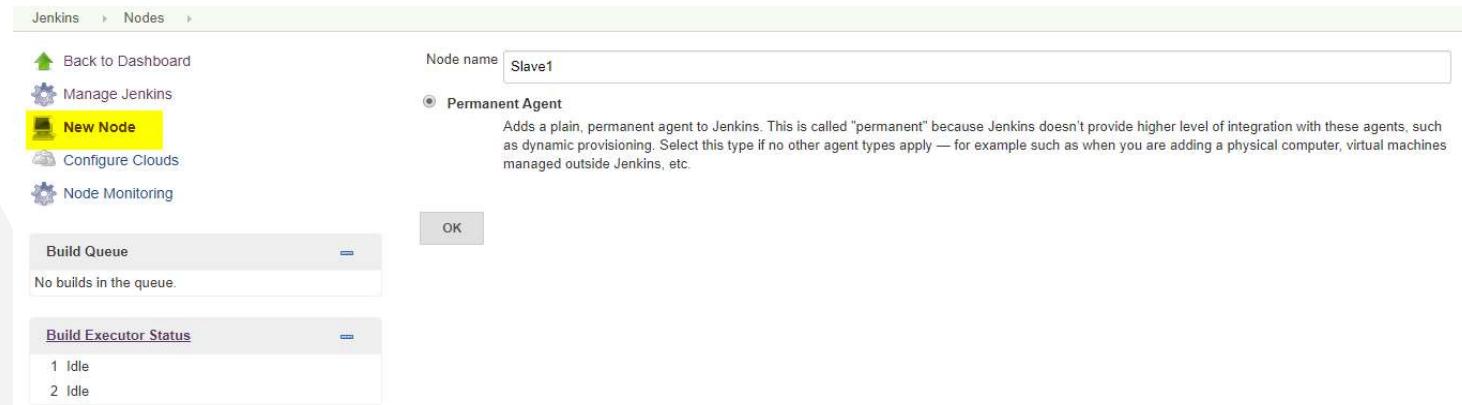
On New Node



The screenshot shows the Jenkins interface for managing nodes. On the left, there's a sidebar with links: Back to Dashboard, Manage Jenkins, New Node (which is highlighted with a yellow box), Configure Clouds, and Node Monitoring. Below this are two collapsed sections: Build Queue (No builds in the queue) and Build Executor Status (1 Idle, 2 Idle). The main content area has a header with a Jenkins logo, the word 'Jenkins', a search bar, a user icon for 'Arvind Phulare', and log out and enable auto refresh buttons. Below the header is a table titled 'Nodes' with one row. The table columns are S, Name (sorted by name), Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The single row shows a node named 'master' running on Windows 10 (amd64), with its status as 'In sync'. The table includes a 'Refresh status' button at the bottom right.

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Windows 10 (amd64)	In sync	306.05 GB	7.89 GB	306.05 GB	0ms
	Data obtained	5 min 14 sec	5 min 14 sec	5 min 14 sec	5 min 14 sec	5 min 14 sec	5 min 14 sec

Give a name for the Node, Choose the Permanent Agent option and click on OK



This screenshot shows the 'New Node' configuration dialog. On the left is a sidebar with the same set of links as the previous screenshot. The main area has a 'Node name' field containing 'Slave1' and a radio button group for 'Agent Type' where 'Permanent Agent' is selected. A descriptive text explains that this adds a plain, permanent agent to Jenkins. At the bottom right is an 'OK' button.

Jenkins / Setup Master and Slaves

Configure Node

of executors: 2

Remote root directory: C:\Users\Arvind Phulare\Desktop\slave_ws

Labels: slave1

Usage: Use this node as much as possible

Launch method: Launch agent by connecting it to the master

Disable WorkDir

Custom WorkDir path: C:\Users\Arvind Phulare\Desktop\slave_ws

Internal data directory: remoting

Fail if workspace is missing

Availability: Keep this agent online as much as possible

Node Properties

Disable deferred wipeout on this node

Save

Jenkins

Nodes

Back to Dashboard | Manage Jenkins | New Node | Configure Clouds | Node Monitoring

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
1	master	Windows 10 (amd64)	In sync	305.77 GB	4.77 GB	305.77 GB	0ms
2	Slave1	Windows 10 (amd64)	In sync	305.77 GB	4.77 GB	305.77 GB	119ms

Data obtained: 8.6 sec, 8.5 sec, 8.5 sec, 7.6 sec, 8.5 sec, 8.5 sec

Build Queue: No builds in the queue.

Build Executor Status

- master: 1 Idle, 2 Idle
- Slave1: 1 Idle, 2 Idle

Jenkins Declarative Pipeline

Declarative Pipeline is a relatively recent addition to Jenkins Pipeline, which features a more simplified and customized syntax in addition to the Pipeline subsystems. Declarative "Section" blocks for common configuration areas like :

- ▶ Stages
- ▶ Tools
- ▶ Post-build actions
- ▶ Notifications
- ▶ Environment
- ▶ Build agent
- ▶ All wrapped up in a pipeline { } step,
with syntactic and semantic validation available.

```
pipeline {
    agent { label 'node-1' }
    stages {
        stage('Source') {
            steps {
                git 'https://github.com/digitalvarys/jenkins-tutorials.git'
            }
        }
        stage('Compile') {
            tools {
                gradle 'gradle4'
            }
            steps {
                sh 'gradle clean compileJava test'
            }
        }
    }
}
```

Jenkins Declarative Pipeline

The **Stages** section contains one or more stage **blocks**.

- ▶ Stages block look the same as the new block-scoped stage step
- ▶ Think of each stage block as like an individual Build Step in a Freestyle job
- ▶ There must be a stages section present in your pipeline block

Example

```
stages {  
    stage("build") {  
        timeout(time: 5, units: 'MINUTES') {  
            sh './run-some-script.sh'  
        }  
    }  
    stage("deploy") {  
        sh "./deploy-something.sh"  
    }  
}
```

Jenkins Declarative Pipeline

The **Agent** determines where your build runs.

Current possible settings :

- ▶ Agent label : ":" - Run on any Node
- ▶ agent docker : "ubuntu" - Run on any Node within a Docker container of the "Ubuntu" image.
- ▶ Agent docker : "ubuntu", label:"foo" - Run on a node with the label "foo" within a Docker container of the "Ubuntu" image
- ▶ Agent none - Don't run on a Node at all, manage Node blocks yourself within your stages.

There must be an agent section in your pipeline block.

Jenkins Declarative Pipeline

The **Tools** section .

- ▶ Allows you to define tools to auto-install and add to the PATH
- ▶ Doesn't work with agent docker:'ubuntu'.
- ▶ This will be ignored if agent none is specified
- ▶ The tools section takes a block of tool name/tool version pairs, where the tool version is what you've configured on this Master

Example

```
tools {  
    maven "Maven 3.3.9"  
    jdk "Oracle JDK 8u40"  
}
```

Jenkins Declarative Pipeline

The **Environment** section .

- ▶ Block of key=value pairs that will be added to the environment when the build runs in.

Example

```
environment {  
    FOO = "bar"  
    BAZ = "faz"  
}
```

Jenkins Declarative Pipeline

The **Notifications** and **Post Build** section .

- ▶ Post Build and notifications both contain blocks with one or more build condition keys and related step blocks.
- ▶ The steps for a particular build condition will be invoked if that build condition is met.
- ▶ **Post Build** checks its conditions and executes them, if satisfied, **after all stages have completed**, in the same Node/Docker container as the stages.
- ▶ **Notifications** checks its conditions and executes them, if satisfied, **after Post Build**, but doesn't run on a Node at all.

Name	Satisfied When...
success	The build is successful
failure	The build has failed
unstable	The build is unstable
changed	The build's status is different than the previous build
always	Always true

Jenkins Declarative Pipeline

The **Notifications** and **Post Build** examples.

```
notifications {  
    success { hipchatSend 'Build passed' }  
    failure {  
        hipchatSend 'Build failed' mail to:'me@example.com',  
        subject:'Build failed',  
        body:'Fix me please!'  
    }  
}  
-----  
postBuild {  
    always { archive "target/**/*" junit 'path/to/*.xml' }  
    failure {  
        sh './cleanup-failure.sh'  
    }  
}
```

Jenkins Declarative Pipeline

```
pipeline {
    tools {
        maven 'Maven 3.3.9'
        jdk 'oracle JDK 8u40'
    }
    // run on any executer
    agent label:''
    stages {
        stage('build') {
            sh 'mvn clean install -Dmaven.test.failure.ignore=true'
        }
    }
    postBuild {
        always {
            archive 'target/**/*'
            junit 'target/surefire-reports/*.xml'
        }
    }
    notification {
        success {
            mail(to:'gilbert.toussido@gmail.com', subject:"SUCCESS: ${currentBuild.fullDisplayName}", body:"Huh, we're success." )
        }
        failure {
            mail(to:'gilbert.toussido@gmail.com', subject:"FAILURE: ${currentBuild.fullDisplayName}", body:"Huh, we're failure." )
        }
        unstable {
            mail(to:'gilbert.toussido@gmail.com', subject:"UNSTABLE: ${currentBuild.fullDisplayName}", body:"Huh, we're unstable." )
        }
    }
}
```

Jenkins Declarative Pipeline with slave

```
pipeline {  
    agent none  
    stages {  
        stage('distribute') {  
            parallel (  
                'windows': {  
                    node('windows') {  
                        bat 'print from windows'  
                    }  
                },  
                'mac': {  
                    node('osx') {  
                        sh 'print from mac'  
                    }  
                },  
                'linux': {  
                    node('linux') {  
                        sh 'print from linux'  
                    }  
                }  
            )  
        }  
    }  
}
```

PART II. Machine Deployment

MODULE II-1 : Virtual Machine deployment

PLAN

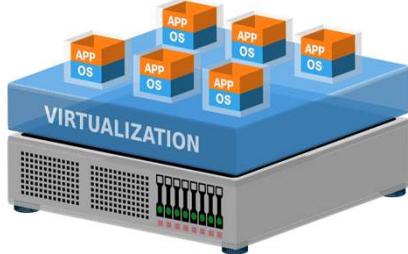
- ▶ **VIRTUAL MACHINE:**
- ▶ **VAGRANT :**
 - Features
 - Architecture
 - Workflow
- ▶ **VAGRANTFILE:**
 - Configure
 - Options
 - Providers
 - Provisioners
 - Boxes
- ▶ **VAGRANT COMMAND**



Virtual Machine

Virtual Machine (VM) is a software implementation of a machine (computer) that executes programs like a physical machine.

- ▶ A Virtual Machine provides an interface identical to the underlying bare hardware.
- ▶ The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.



Types of Virtual Machine

There are two types :

► **System Virtual Machine-Hardware Virtual Machine :**

Provides a complete system platform environment which supports the execution of a complete operating system (OS).

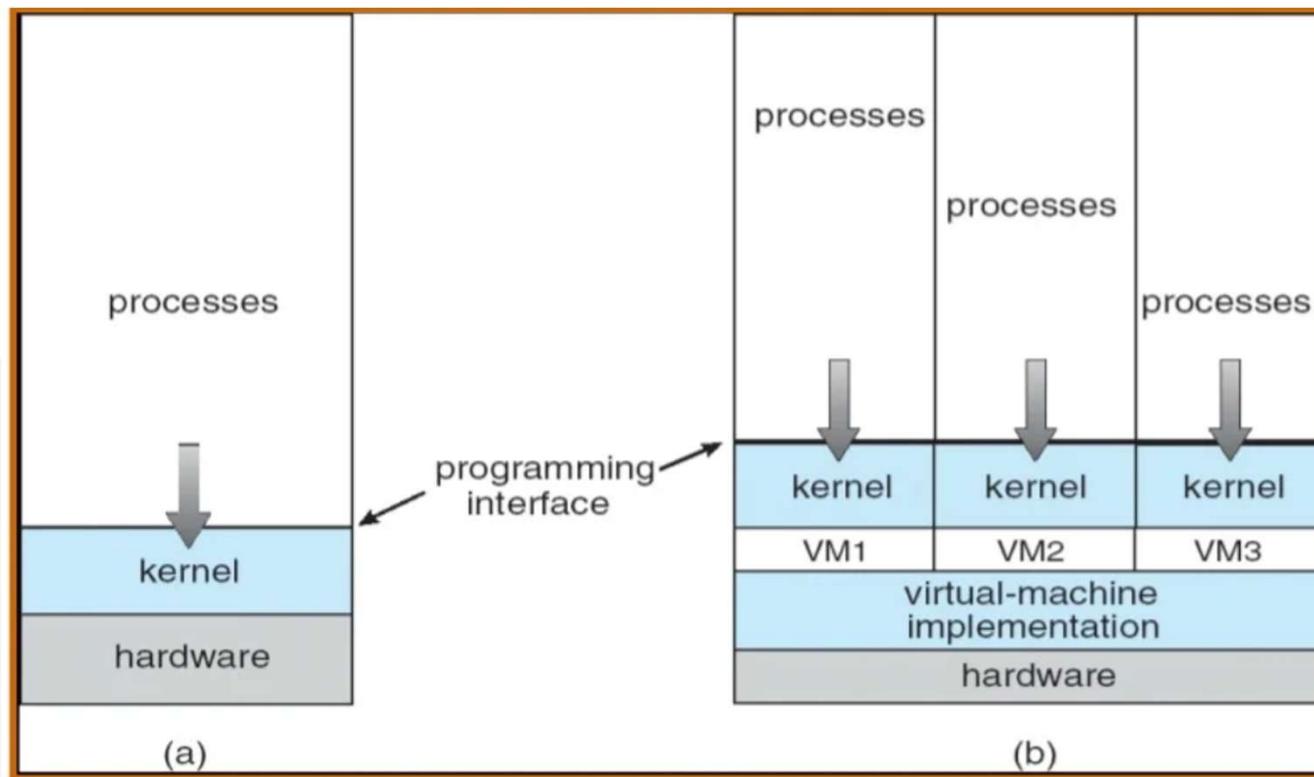


► **Process Virtual Machine-Application Virtual Machine**

Provides a platform-independent programming environment that abstracts away details of the underlying hardware or operating system from software or application runtime.



Virtual Machine Architecture



Virtual Machine

Advantage and disadvantages are :

Advantages	Disadvantages
Familiar Interfaces	Difficulty in direct access to hardware
Isolation of OS and Reduction of cost	RAM degradation with the creation of new VMs
High Availability and Scalability	Disk Space degradation with the creation of new VMs
Backup with Fast Recovery	Less efficient than physical Machine

Vagrant for VM Deployment

Vagrant is a wrapper around Virtual Machines :

- ▶ Open-source tool for building and distributing development environments.
- ▶ Developed by **Mitchell Hashimoto** (Founder of **HashiCorp**)
- ▶ First stable version 1.0 and was released in March 2012 (the current **2.3.0**)
- ▶ Create standard environments using provisioning scripts
- ▶ Allow working on the same base configuration with remote access to boxes
- ▶ Help test and debug remotely
- ▶ Development environments managed can run on **Local Virtualized Platform** (Virtualbox, VMWare) , **Cloud** (AWS, Azure, Openstack), and **Containers** (Docker)
- ▶ It provides a simple and easy way to use **Command-line client** (Manage these environments) and **interpreter** (Vagrantfiles : text-based definitions of what each environment looks like)



Vagrant features

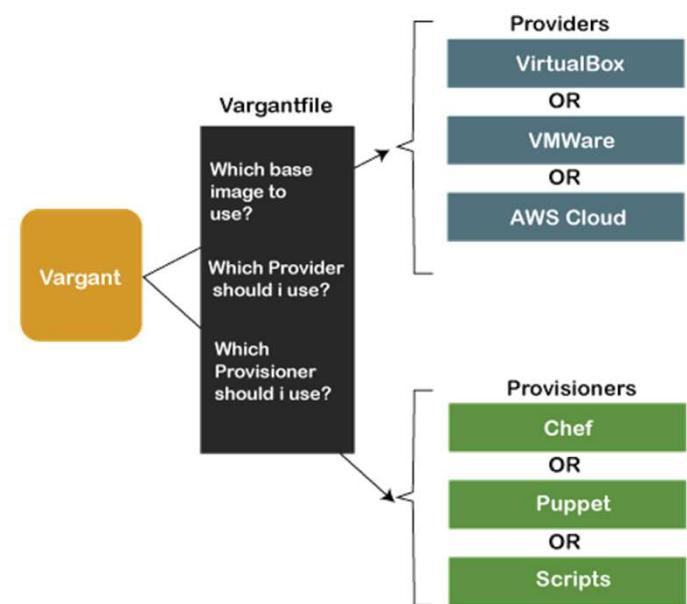
Vagrant uses **Providers** and **Provisioners** as building blocks to manage development environments.

- ▶ **Provisioners** are tools that allow users to customize the configuration of virtual environments.

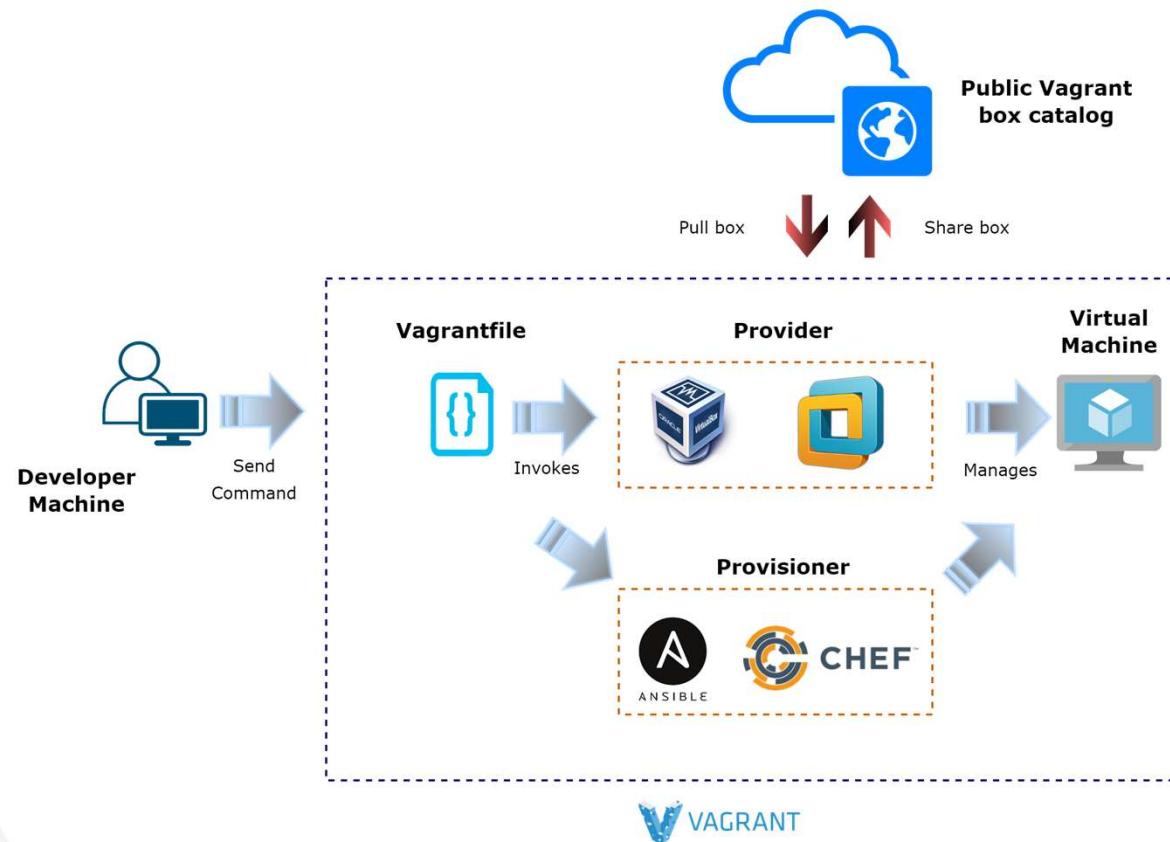
Example : File, Shell, Chef and Puppet

- ▶ **Providers** are the services that Vagrant uses to set up and create virtual environments.

Example : VirtualBox, VMWare, Hyper-V, Docker, AWS...

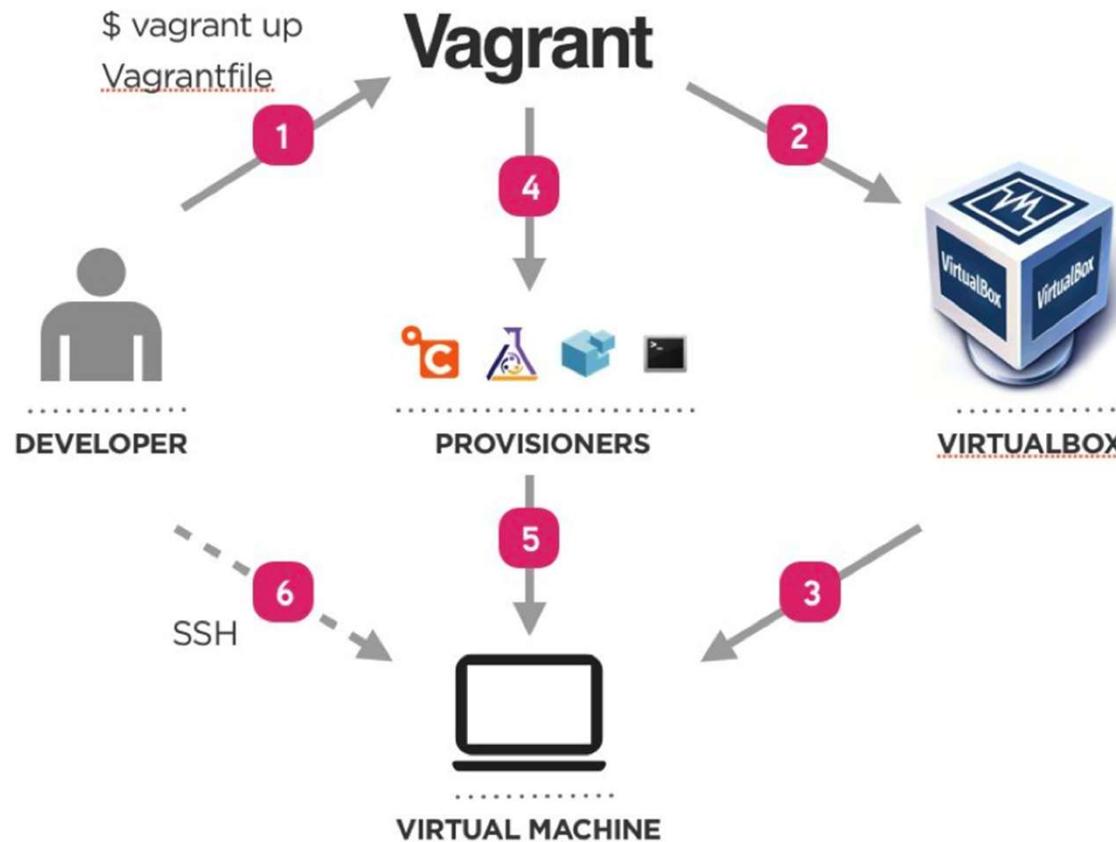


Vagrant Architecture



Vagrant Workflow

- ▶ (1) Developer Create *Vagrantfile* and run the VM
- ▶ (2) Vagrant connects to the VirtualBox provider to set up virtual environment.
- ▶ (3) VirtualBox start the VM
- ▶ (4) Vagrant executes the provisioners
- ▶ (5) Provisioners install tools on VM startup
- ▶ (6) Developer can access the VM through SSH connection



Vagrantfile

Vagrantfile is a Ruby file that instructs Vagrant to create, depending on how it is executed, new Vagrant machines or boxes.

Vagrant Box is considered as an image, a template from which we will deploy our future virtual machines.

- ▶ *Vagrant Box* is a compiled *Vagrantfile* describing a type of Vagrant machines. A new Vagrant machines can be created from a Vagrant Box
- ▶ *Vagrantfile* can directly create one or more Vagrant machines

```
# Simple Vagrantfile example

Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.hostname = "node1"
  config.vm.network "private_network", ip: "192.168.33.10"
  config.vm.synced_folder "../data", "/vagrant_data"
  config.vm.provider "virtualbox" do |vb|
    vb.customize ["modifyvm", :id, "--memory", 1024 * 4]
  config.vm.provision :shell, path: "bootstrap.sh"
end
```

Vagrantfile

- ▶ `Vagrant.configure("2")`: returns the Vagrant configuration object for the new box. Config alias is used to refer to this object. The version 2 of Vagrant API is used
- ▶ `Vm.box` : is the base box that we are going to use. The schema for box names is the maintainer account in Vagrant Cloud followed by the box name.
- ▶ `Vm.hostname` : sets the hostname of the box
- ▶ `Vm.network` : Configures network
- ▶ `vm.synced_folder` : to configures the synced folders between the host and the guest
- ▶ `Vm.provider` : Configures settings specific to a provider. Allows overriding options for the Virtual Machine provider. For example: memory, CPU, ...
- ▶ `Vm.provision` : to specify the name of the file that is going to be executed at the machine creation

Vagrantfile Configure Network

The parameter `Vm.network`:

- ▶ **Port forwarding** : Forward all requests from a service running on *port 80* of the Vagrant Virtual Machine to *port 8080* of the host machine.

```
config.vm.network "forward_port", guest:80,host:8080
```

- ▶ **Static IP** : Assign a private IP address to the Virtual Machine

```
config.vm.network "private_network", ip: "192.168.33.10"
```

- ▶ By default, networks are private (only accessible from the host machine)
- ▶ Use the flag "*public_network*" to make the guest network accessible from the LAN (Loacl Area Network)

```
config.vm.network "public_network", ip: "10.0.0.1"
```

Vagrantfile Options

Vagrantfile Options

- ❑ config.ssh.username
- ❑ config.ssh.host
- ❑ config.ssh.port
- ❑ config.ssh.private_key_path
- ❑ config.vagrant.host
- ❑ config.vm.box
- ❑ config.vm.box_url
- ❑ config.vm.customize
- ❑ config.vm.define
- ❑ config.vm.forward_port
- ❑ config.vm.guest
- ❑ config.vm.host_name
- ❑ config.vm.network
- ❑ config.vm.provision
- ❑ config.vm.provider
- ❑ config.vm.share_folder

```

1  # -*- mode: ruby -*-
2  # vi: set ft=ruby :
3
4  VAGRANTFILE_API_VERSION = "2"
5
6  Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
7    config.vm.box = "geerlingguy/centos7"
8    config.vm.hostname = "nodejs.test"
9    config.vm.network :private_network, ip: "192.168.55.55"
10   config.ssh.insert_key = false
11   config.vm.synced_folder ".", "/vagrant", disabled: true
12
13  config.vm.provider :virtualbox do |v|
14    v.memory = 512
15  end
16
17  # Ansible provisioner.
18  config.vm.provision :ansible do |ansible|
19    ansible.playbook = "provisioning/playbook.yml"
20  end
21 end

```

Vagrant provisoners

- ▶ Allows initial configuration of the VM to easily set up your VM with everything it needs to run your software
- ▶ Important part of making VM creation repeatable
- ▶ Scripts made for provisioning can typically be used to set up production machines quickly as well
- ▶ Some available provisioners :



```
# Shell provisioning example

Vagrant.configure("2") do |config|
  #...other configuration

  config.vm.provision :shell" do |s|
    s.inline = "echo hello Fongan"
    s.path   = "scripts/bootstrap.sh"
  end
end
```

```
# File provisioning example

Vagrant.configure("2") do |config|
  #...other configuration

  config.vm.provision :file" do |f|
    f.source= ".gitconfig"
    f.destination  = "~//.gitconfig"
  end
end
```

Vagrant Boxes

Box is the base image used to create a virtual environment with Vagrant

A box is a compressed file containing the following :

- ▶ **Vagrantfile** : the information from this will be merged into your Vagrantfile that is created when you run `vagrant init boxname` in a folder.
- ▶ **Box-disk.vmdk** : The Virtual Machine image.
- ▶ **Box.ovf** : Defines the virtual hardware for the box
- ▶ **Metadata.json** : Inform Vagrant about the provider the box work with.

<code>vagrant box list</code>	See a list of all installed boxes on your computer
<code>vagrant box add <name><url></code>	Download a box image to your computer
<code>vagrant box outdated</code>	Check for updates vagrant box update
<code>vagrant boxes remove <name></code>	Deletes a box from the machine
<code>Vagrant package</code>	Packages a running VirtualBox environment in a reusable box

Vagrant command (1/3)

Creating a VM : Vagrant init : Initialize vagrant with Vagrantfile and ./vagrant directory

vagrant init - m	Create a minimal Vagrantfile (no comments or helpers)
vagrant init - f	Create a new Vagrantfile, overwriting the one at the current path
vagrant init -box-version	Create a Vagrantfile, locking the box to a version constraint
Vagrant init <boxpath>	Initialize Vagrant with a specific box. To find a box, go to the public Vagrant box catalog. For example, <i>vagrant init ubuntu/trusty64</i>

Starting a VM

vagrant up	Starts vagrant environment (also provisions only on the FIRST vagrant up command)
vagrant resume	Resume a suspended machine (vagrant up works just fine for this as well)
vagrant provision	Forces re-provisioning of the vagrant machine
Vagrant reload	Restarts vagrant machine, loads new Vagrantfile configuration
Vagrant reload --provision	Restart the virtual machine and force provisioning

Vagrant command (2/3)

Getting into a VM

<code>vagrant ssh</code>	Connects to machine via SSH
<code>vagrant ssh <boxname></code>	If you give your box a name in your Vagrantfile, you can ssh into it with. Boxname works from any directory

Stopping a VM

<code>vagrant halt</code>	Stops the Vagrant machine
<code>vagrant suspend</code>	Suspends a Virtual Machine (remembers state)

Saving Progress

<code>Vagrant snapshot save [options][vm-name] <name></code>	Allows us to save the VM so that we can roll back at a later time.
--	--

Vagrant command (3/3)

Other tips

vagrant -v	Get vagrant version
vagrant status	Outputs status of the Vagrant machine
vagrant global-status	Outputs status of all vagrant machines
vagrant global-status --prune	Outputs status of all vagrant machines, but prunes invalid entries
Vagrant provision --debug	Use the debug flag to increase the verbosity of the output
Vagrant push	Vagrant can be configured to deploy code on remote central registry
Vagrant up --provision tee provision.log	Runs vagrant up, forces provisioning and logs all output to a file

MODULE II-2 :

Cloud deployment

PLAN

► **CLOUD DEPLOYMENT MODEL:**

- Private
- Public
- Hybrid
- Community

► **CLOUD SERVICE MODEL:**

- Cloud Foundry
- Openshift
- Openstack

► **CLOUD INIT:**

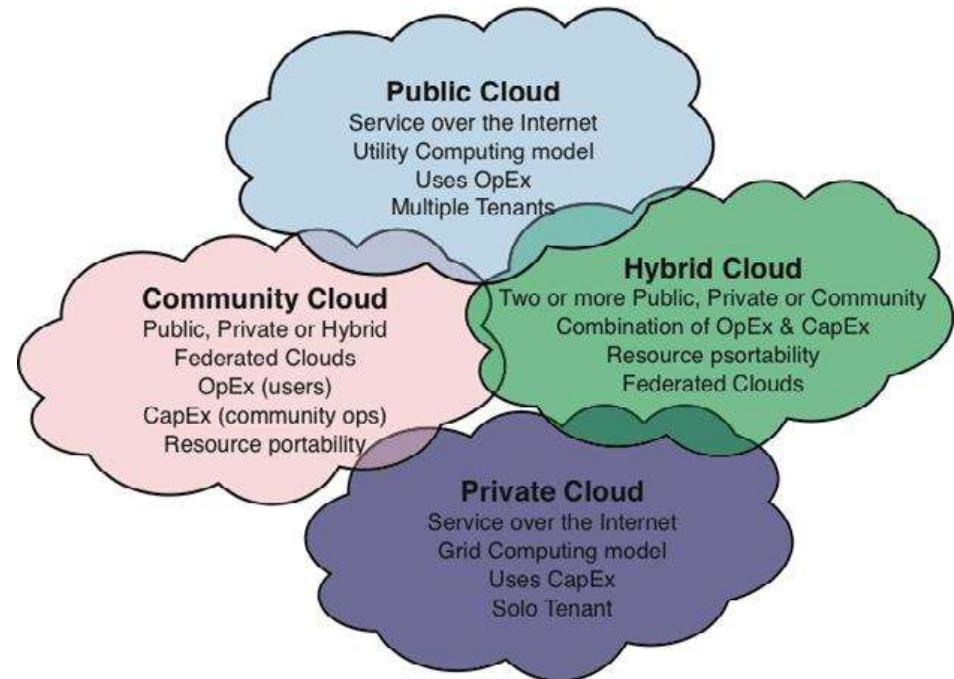
- Syntax
- Example



Cloud Deployment model

Cloud can be classified in terms of who owns and manages the cloud. Types of Cloud (Deployment Model)

- ▶ Public Cloud,
- ▶ Private Cloud,
- ▶ Hybrid Cloud,
- ▶ Community Cloud,

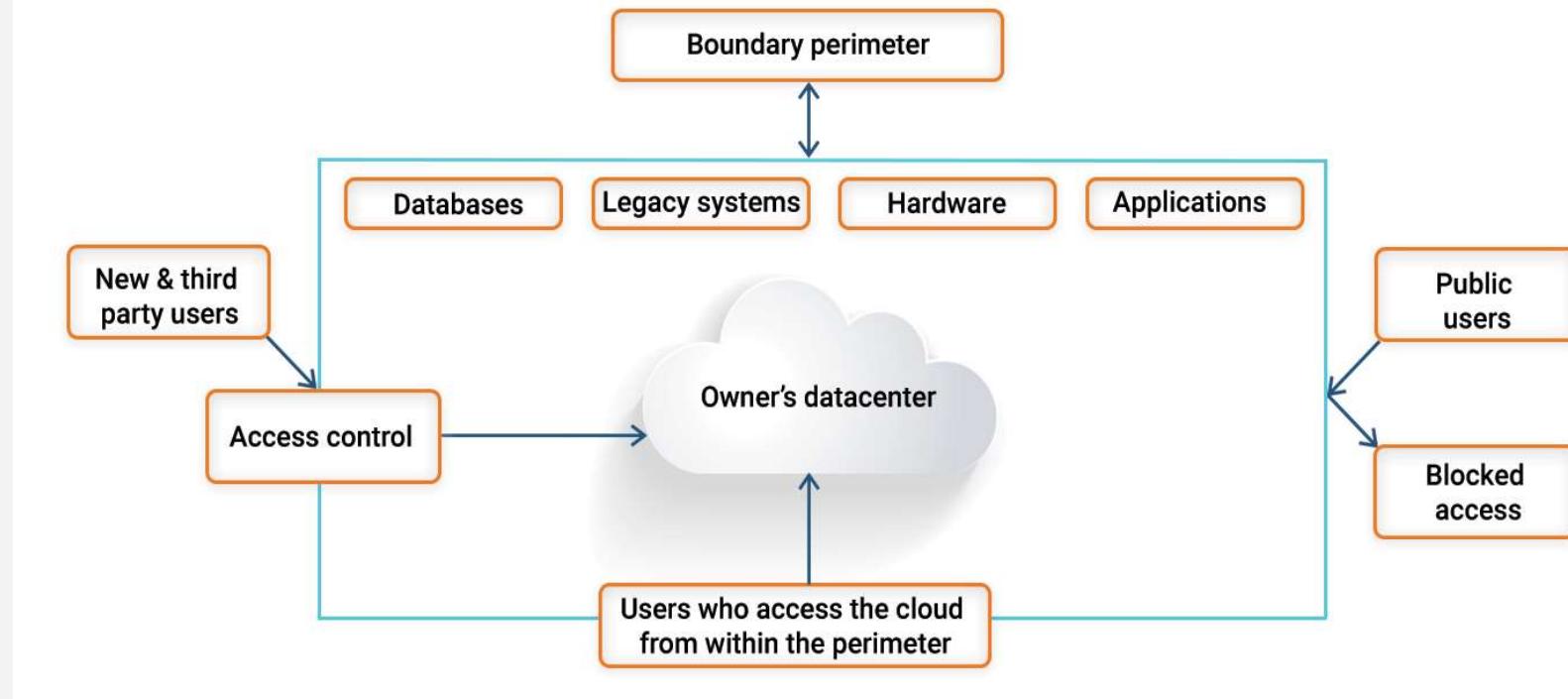


Private Cloud

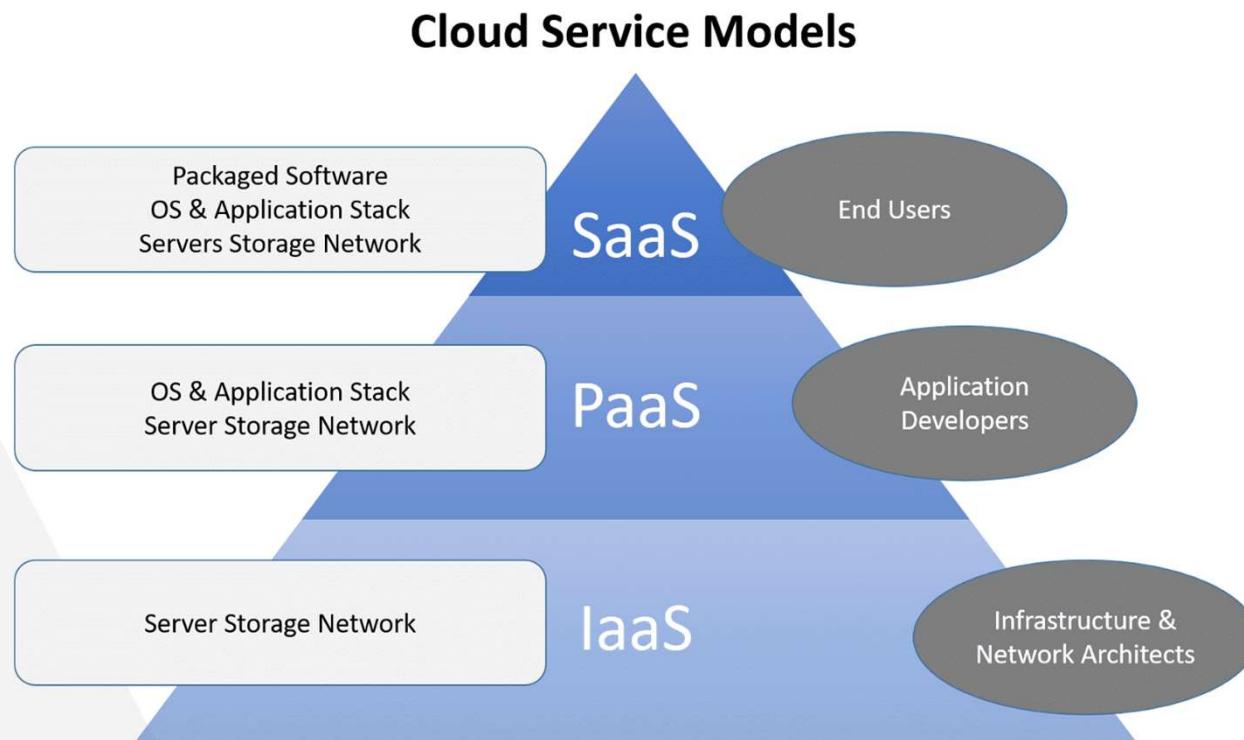
- ▶ A **private Cloud** or **internal Cloud** is used when the Cloud infrastructure, proprietary network or data center, is operated solely for a business or organization, and serves customers within the business fire-wall.
- ▶ Most of the **private Cloud** are large company or government departments who prefer to keep their data in a more controlled and secure environment.
- ▶ The difference between a **private Cloud** and **public Cloud** is that in a private Cloud-based service, data and processes are managed within the organization without the restrictions network bandwidth, security exposures and legal

Private Cloud

PRIVATE CLOUD



Cloud Service Model



PaaS & IaaS Platform

Cloud Foundry : provides a highly efficient, modern model for cloud native application delivery on top of Kubernetes.

- ▶ Application and services centric lifecycle API
- ▶ Container-based architecture
- ▶ External dependencies are considered services



CLOUD FOUNDRY

Openshift : cloud-based Kubernetes platform that helps developers build applications.

- ▶ Managing applications written in different languages
- ▶ Uses a hyper-visor to abstract the layer from the underlying hardware



OPENSHIFT

Openstack :

- ▶ Virtual servers and other resources are made available to customers
- ▶ Interrelated components that control diverse, multi-vendor hardware pools of processing, storage and networking resources throughout a data center



Cloud Deployment

Cloud-init :

- ▶ Industry standard multi-distribution method for cross-platform Cloud instance initialization
- ▶ Supported across all major public cloud providers.
- ▶ Customize a new server installation during its deployment using *data* supplied in configuration files
- ▶ Cloud init's behavior can be configured via user-data.
- ▶ User-data can be given by the user at **instance launch time** via "**--user-data**" or "**--user-data-file**" argument to a run instances command within CCloud platform's CLI tool.
- ▶ Modular and highly configurable
- ▶ Supported user data formats :

Shell scripts(starts with **#!**), Cloud config files (starts with **#cloud-config**).



Cloud-init Modules

Cloud-init has modules for handling :

- ▶ Disk configuration
- ▶ Command execution
- ▶ Creating users and groups
- ▶ Package management
- ▶ Writing content files
- ▶ Bootstrapping Chef/Puppet/Ansible
- ▶ Additional modules can be written in Python if desired

Some of the things it configures are :

- ▶ Setting a default local
- ▶ Setting hostname
- ▶ Generate SSH private keys
- ▶ Adding SSH keys to user's `.ssh/authorized_keys` to log in
- ▶ Setting up ephemeral mount points

Cloud Config Syntax

- ▶ Run 'apt-get upgrade' on first boot

```
#cloud-config
apt_upgrade : true
```

- ▶ Enable *byobu* by default for all system users

```
#cloud-config
byobu_by_default : system
```

- ▶ Import ssh keys for launchpad user 'smoser' and add his ppa

```
#cloud-config
ssh_import_id : [smoser]
apt_sources :
  - source : "ppa:smoser/ppa"
```

- ▶ Run a few commands on first boot

```
#cloud-config
runcmd:
  - [ wget, http://slashdot.org, - 0, /tmp/index.html]
  - [ sh, -xc, "echo $(data) ':hello world!'" ]
```

Cloud-init Example

- ▶ Configuration of instance through "user-data" provided to cloud-init
- ▶ The most popular formats for scripts user-data is the cloud-config.
- ▶ Example of YAML file "cloud-init.yaml"

```
#cloud-config

package_update: true
packages:
  - apt-transport-https
  - ca-certificates
  - curl
  - gnupg-agent
  - software-properties-common

runcmd:
  - curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
  - add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
  - apt-get update -y
  - apt-get install -y docker-ce docker-ce-cli containerd.io
  - systemctl start docker
  - systemctl enable docker

final_message: "The system is finally up, after $UPTIME seconds"
```

- ▶ File compatible with Ubuntu instance. Necessary to adapt this file if you are using another operating system

Cloud-init Example

- ▶ Some explanations :

- **package_update** : Update of the apt database on first boot.
- **packages** : The list of packages to install
- **runcmd** : Contains a list of commands to be executed
- **final_message** : This message will be displayed at the end of the first start (Find it in the logof Cloud-init)

- ▶ Use cloud-init configuration file with multipass to validate that it works

```
$ multipass launch -n my-testinit --cloud-init cloud-config.yaml
```

- ▶ Access to Docker on new machine with the following command :

```
$ multipass exec my-testinit --sudo docker ps
```

- ▶ Check the cloud-init log :

```
$ multipass exec my-testinit --sudo cat /var/log/cloud-init-output.log
```

Cloud-init Example on Azure VM

The screenshot shows the 'Advanced' tab of the Azure VM creation interface. A red box labeled '1' highlights the 'Advanced' tab itself. Another red box labeled '2' highlights the 'Custom data and cloud init' section. In this section, there is a code editor containing the following YAML configuration:

```
package_upgrade: true
packages:
- nginx
```

Below the code editor, a tooltip states: "Custom data on the selected image will be processed by cloud-init. Learn more about custom data for VMs".

At the bottom, there are buttons for "Review + create", "< Previous", and "Next : Tags >".

MODULE II-3 : System Image Creation

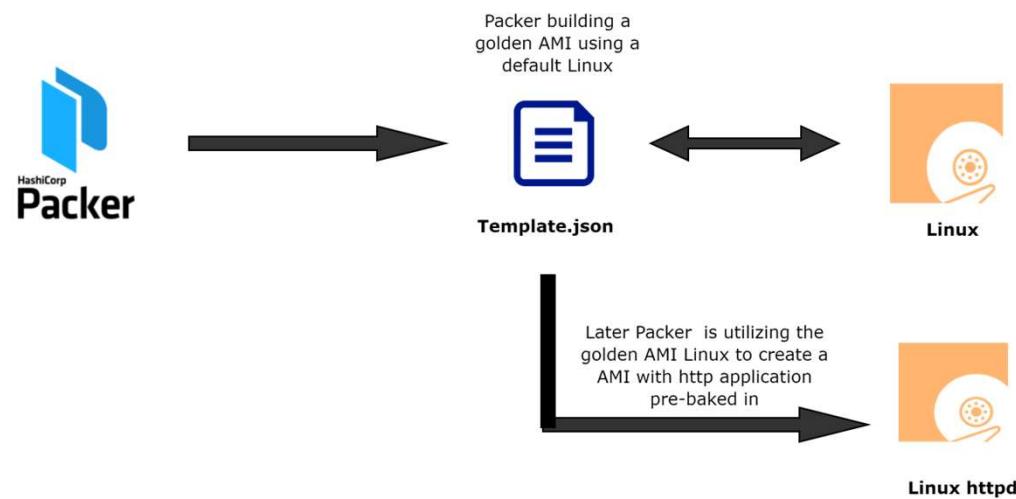
PLAN

- **PACKER :**
 - Advantages
 - Use cases
- **PACKER INSTALLATION**
- **PACKER WORKFLOW**
- **PACKER BUILD**
- **PACKER PROVISION**
- **PACKER POST PROCESS**
- **USE CASE ON AWS**
- **PACKER COMMAND**



Packer

- ▶ **Packer** is an open-source tool for creating identical machine images for multiple platforms from a single source configuration
- ▶ Packer is lightweight, runs on every major operating system
- ▶ Packer does not replace configuration management like Chef/Puppet when building images, on the contrary it uses them to install software onto the image.



Advantages of using Packer

Super-fast infrastructure deployment

- ▶ Packer images allow you to launch completely **provisioned and configured machines** in seconds, rather than several minutes or hours.
- ▶ **Machines** can also be launched in seconds without waiting for a typically much longer provisioning time

Multi-provider portability

- ▶ Packer creates identical images for **multiple platforms** : Run development in desktop virtualization solutions like VMWare/VirtualBox , staging/QA in a private Cloud like Openstack and production in AWS/Azure.

Improved stability

- ▶ Packer installs and configures all the software for a machine at the time the image is built
- ▶ If there are bugs in these scripts, they'll be caught early, rather than several minutes after a machine is launched.

Use Cases Packer

The following are use cases of Packer :

- ▶ **Continuous Delivery**

Packer is lightweight, portable and command-line driven. This makes it the perfect tool to put in the middle of your Continuous delivery pipeline.

- ▶ **Dev/Prod Parity**

Packer helps keep development, staging and production as similar as possible

- ▶ **Appliance/Demo Creation**

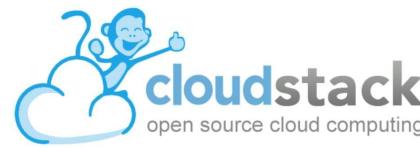
Packer is perfect for creating appliances and disposable product demos. As your software changes, you can automatically create appliances with the software pre-installed.

Supported Platforms

Supported platform are :



Amazon EC2



Google
Compute
Engine



You can add support to any platform by extending Packer using plugins

Packer Installation

- ▶ **Ubuntu/Debian:** HashiCorp officially maintains and signs packages

```
$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
$ sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
$ sudo apt-get update && sudo apt-get install packer
```

- ▶ **Mac OS :** with Homebrew

```
$ brew tap hashicorp/tap
$ brew install hashicorp/tap/packer
```

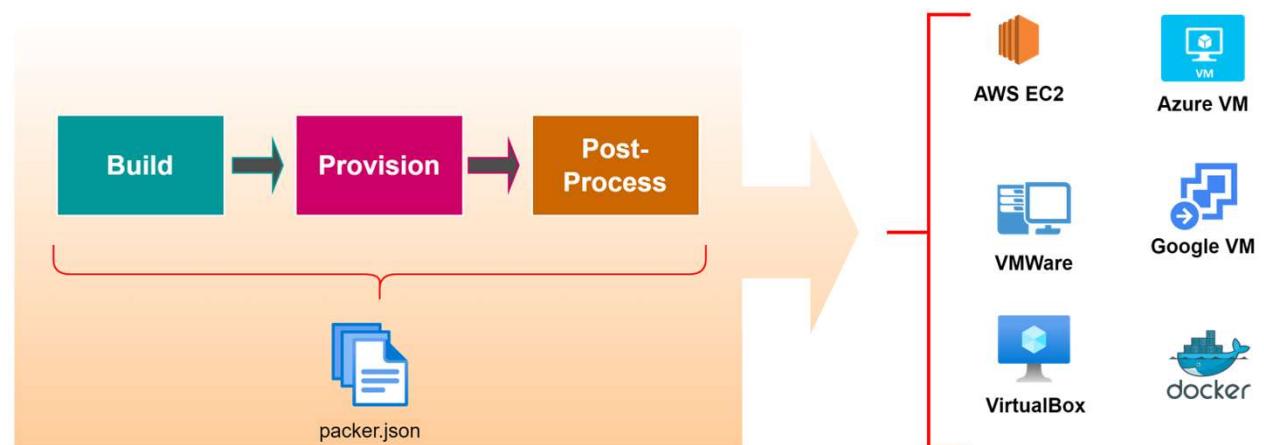
- ▶ **Windows :**

```
$ brew tap hashicorp/tap
$ brew install hashicorp/tap/packer
```

Packer Workflow

The main terminology of Packer are :

- ▶ **Templates** : JSON files containing the build information
- ▶ **Builders** : Platform specific building configuration
- ▶ **Provisioners** : Tool that install software after the initial OS install
- ▶ **Post-processors** : Actions to happen after the image has been built



Packer Build

```
packer.json
{
  "variables": {
    "aws_access_key": "{{env 'AWS_ACCESS_KEY'}}",
    "aws_secret_key": "{{env 'AWS_SECRET_KEY'}}"
  },
  "builders": [
    {
      "type": "amazon-ebs",
      "access_key": "{{user 'aws_access_key'}}",
      "secret_key": "{{user 'aws_secret_key'}}",
      "region": "us-east-1",
      "source_ami": "ami-fce3c696",
      "instance_type": "t2.micro",
      "ssh_username": "admin",
      "ami_name": "yourApp {{timestamp}}"
    }
  ]
}

$ packer validate packer.json : Validate command
```

Packer can create multiple images for multiple platforms in parallel, all configured from a single template [8].

Packer Build Command with vars

- ▶ Via Command line using **-var flag**

```
> packer build  
-var 'aws_access_key=id'  
-var 'aws_secret_key=Secret'  
Packer.json
```

- ▶ Via File with **-var-file flag**

```
variables.json  
{  
  "aws_access_key": "accessKey",  
  "aws_secret_key": "secretKey"  
}
```

```
$ packer build -var-file=variables.json packer.json
```

Var-file flag can be specified multiple times and variables from multiple files will be read and applied. Combining the **-var** and **-var-file** flags together also works how you'd expect. Flags set later in the command override flags set earlier.

Packer Provision

```
packer.json
{
  "variables": [...],
  "builders": [...],
  "provisioners": [
    {
      "type": "shell",
      "inline": [
        "sleep 30", --waiting for SSH to be available
        "sudo apt-get update",
        "sudo apt-get install -y redis-server"
      ]
    },
    {
      "type": "shell",
      "script": "./scripts/install-java.sh",
    }
  ]
}
```

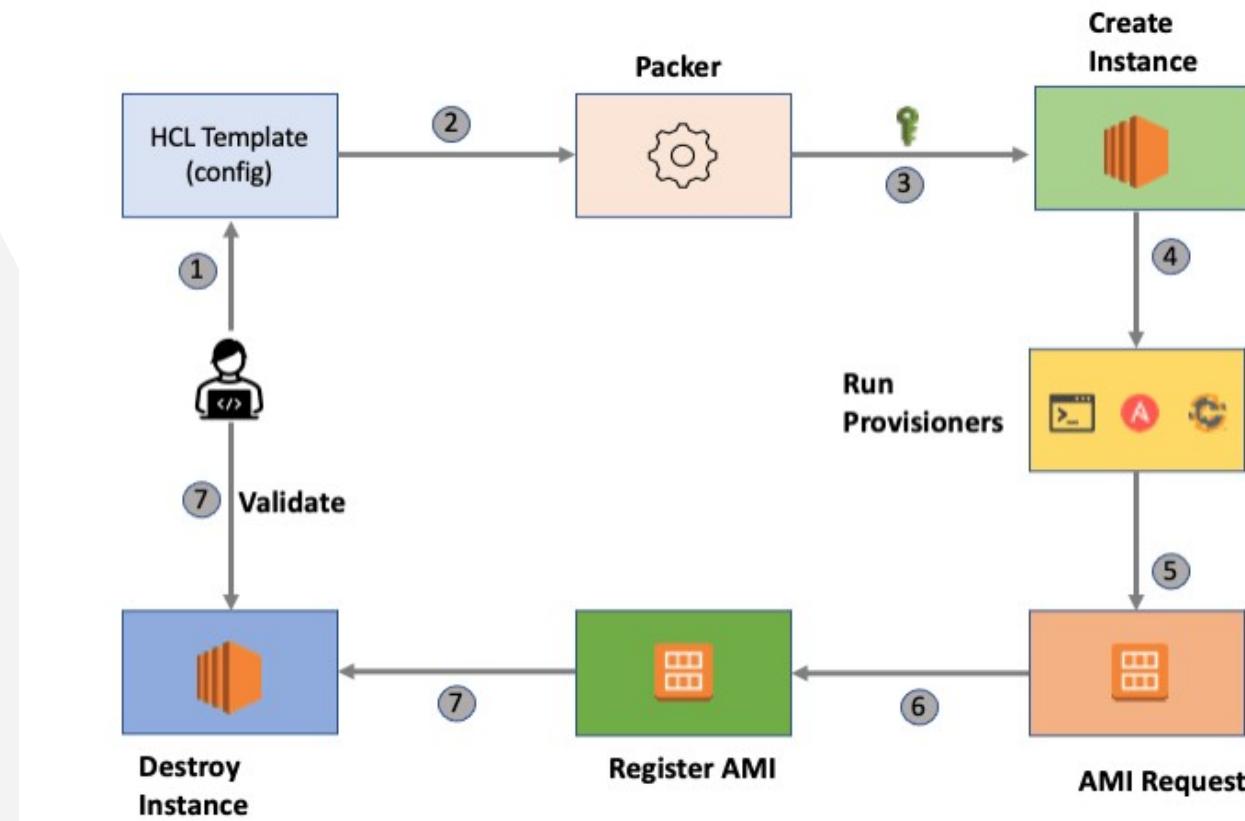
Others – Remote shell, File uploads, Ansible (local&remote), Chef, Puppet, Salt, PowerShell etc.

Packer Post Process

```
packer.json
{
  "variables": [...],
  "builders": [...],
  "provisioners": [...],
  "post-processors": [
    {
      "type": "compress",
      "format": "tar.gz"
    }
  ]
}
```

Others – Amazon Import, CheckSum, Docker Push/Tag/Save, Google Compute Export, Vagrant, vSphere.

Packer/Use case on AWS



Packer command

CLI

packer init	Install required plugins
Packer plugins required	List plugins that will be installed by "packer init"
Packer plugins installed	List installed plugins
Packer build	Build image from template.Takes a template and runs all the builds within it in order to generate a set of artifacts. Use <code>--force</code> to forces a builder to run
Packer fmt	Format HCL2 configuration files to a canonical format and style
Packer validate	Check that a template is valid
Packer inspect	See components of a template
Packer hcl2_upgrade	Convert JSON to HCL2

PART III. Container Management

...

MODULE III-1 : Container Usage

PLAN

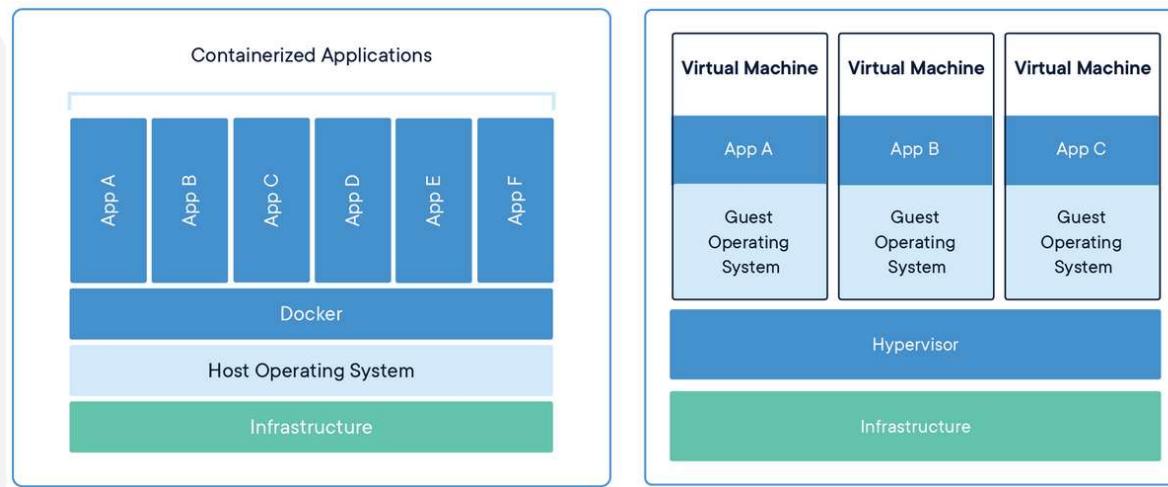
- ▶ VIRTUALIZATION
- ▶ DOCKER
 - Functionality
 - Benefits
 - Architecture
 - Engine
- ▶ CONTAINER RUNTIME
- ▶ DOCKER IMAGES
 - Image
 - Registries
 - Naming and tagging
 - Layers
 - Commands
- ▶ CONTAINERIZATION OF APP :
 - Dockerfile
 - Instructions
 - Example
 - Command
- ▶ MULTI-STAGE BUILD



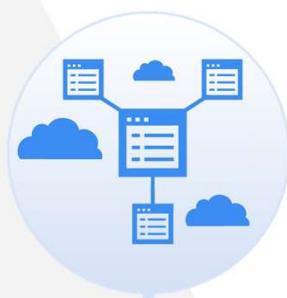
Virtualization

In Computing, **Virtualization** refers to the act of making a virtual version of one thing, together with virtual hardware platforms, storage devices, and electronic network resources.

Containerization is a form of virtualization where applications run in isolated user spaces, called containers, while using the same shared operating system (OS).



Advantages of Containerization over Virtualization



**Increased
Portability**



**Improved
Scalability**



**Simple and Fast
Deployment**



**Enhanced
Productivity**



**Improved
Security**

Docker

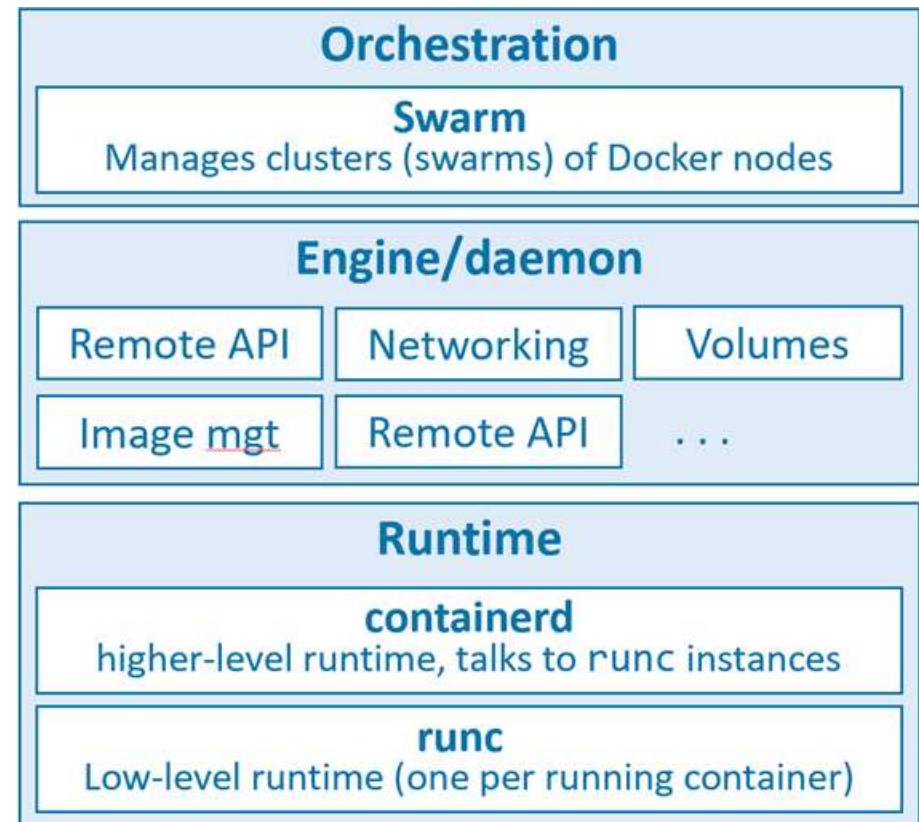
- ▶ Docker is a software that runs on Linux and Windows environments
- ▶ It creates, manage and orchestrates **Containers**.
- ▶ The Docker project is open-source and the upstream lives in the moby/moby repo on GitHub
- ▶ Docker, Inc. Is the overall maintainer of the open-source project and offers commercial versions of Docker with support contracts.
- ▶ There are two main editions of Docker : **Enterprise Edition (EE)** and **Community Edition (CE)**
- ▶ Docker version numbers follow the YY.MM-xx versioning scheme "19.03.12(25 juin 2020)"
- ▶ Tool that is designed to benefit both developers and IT operators, making it a part of many DevOps toolchains.



Docker Technologies

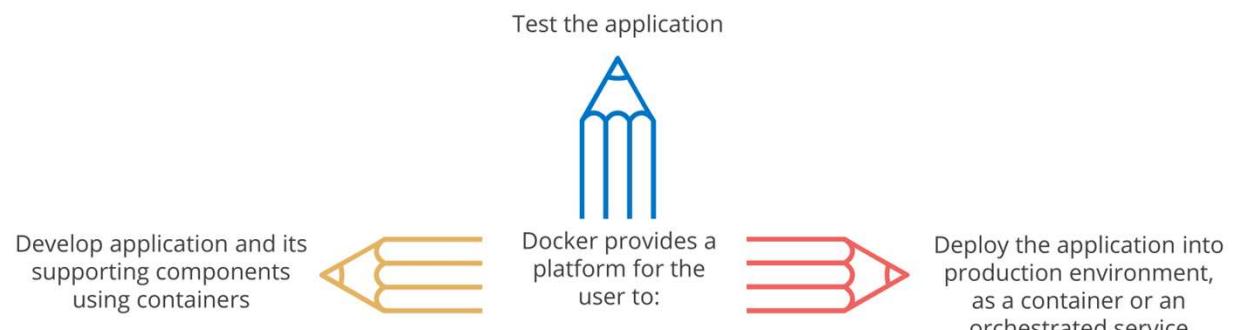
Docker technologies include at least three things to be aware :

- ▶ The runtime
- ▶ The daemon or Engine
- ▶ The orchestrator

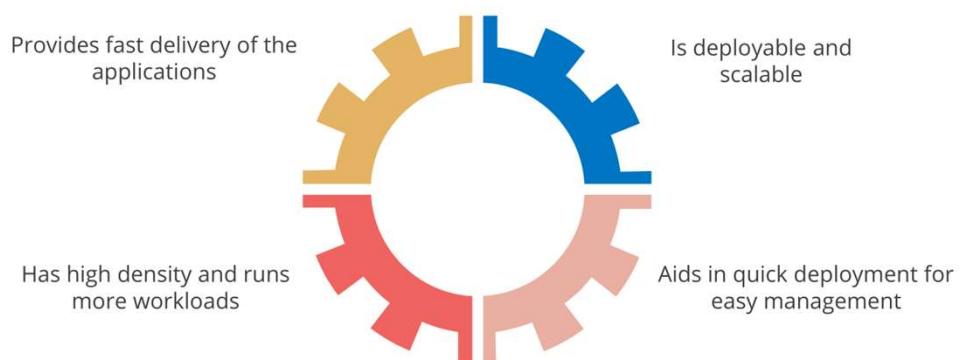


Docker Functionalities & Properties

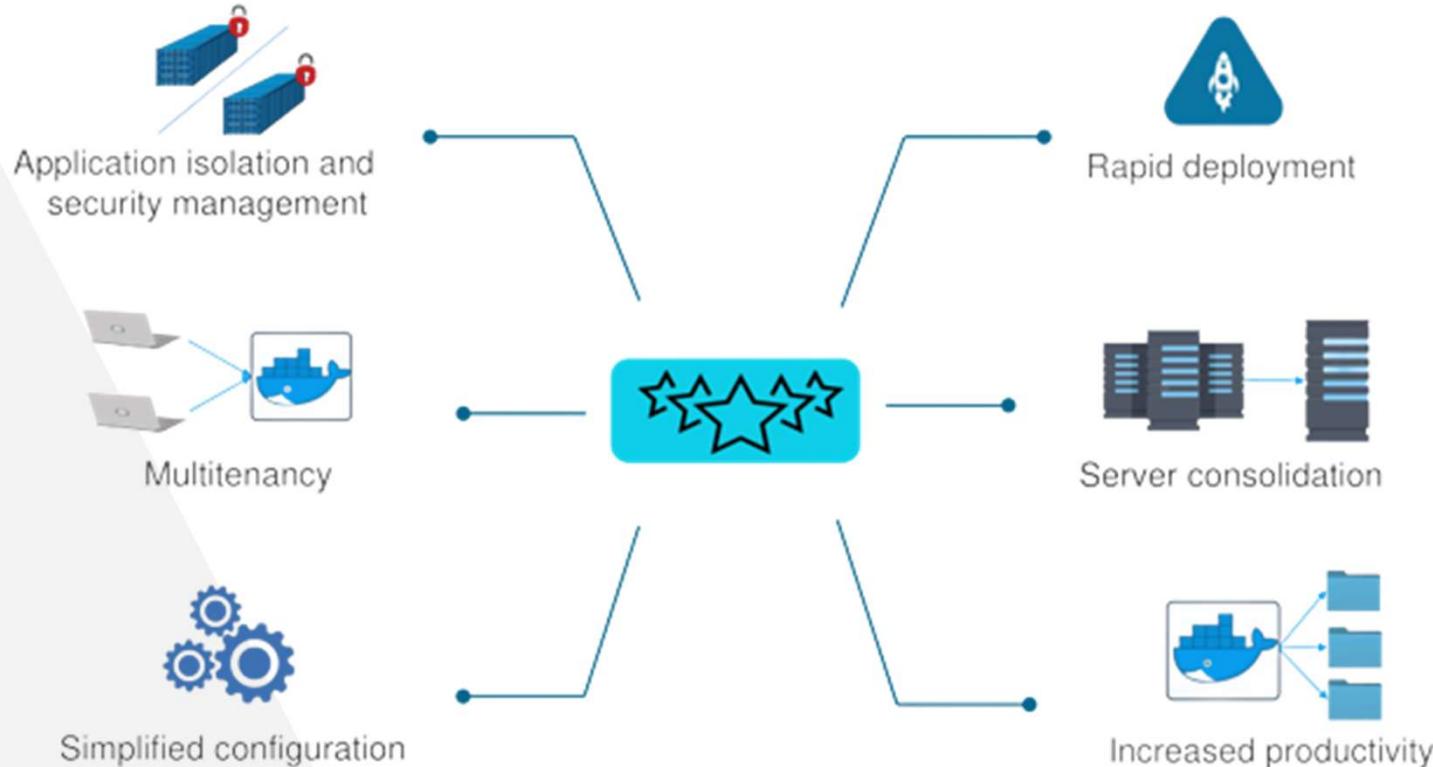
► Docker Functionalities



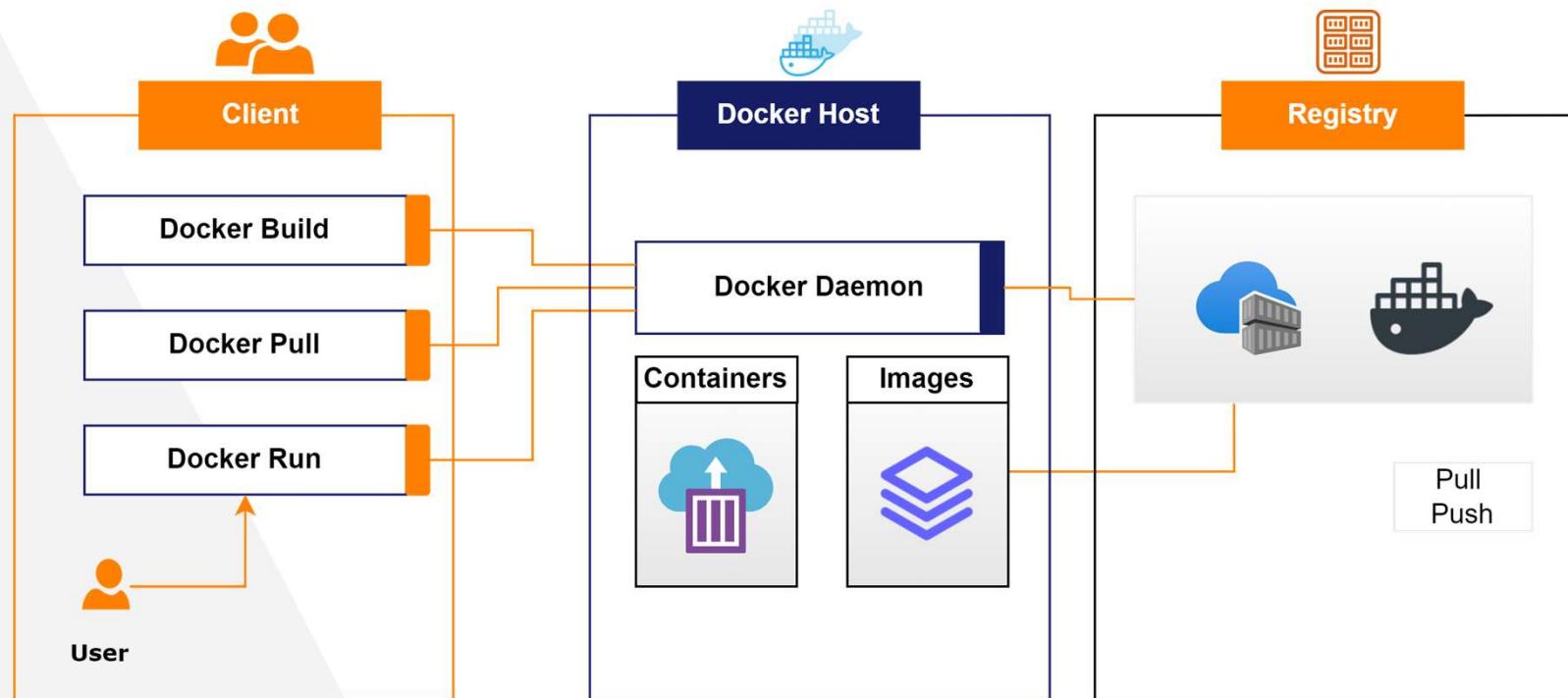
► Docker Properties



Benefits of Docker

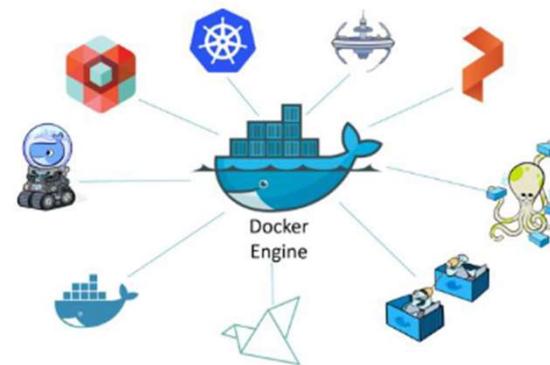
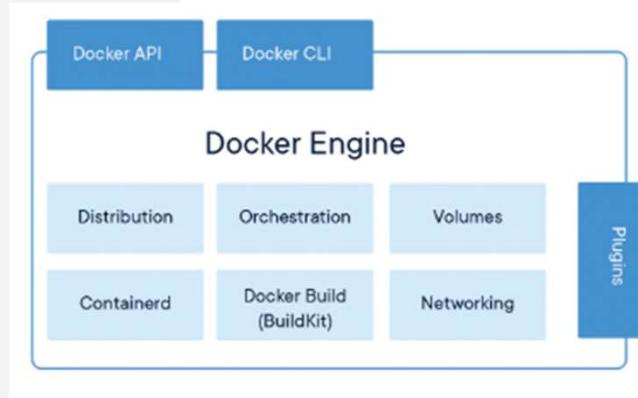


Docker Architecture



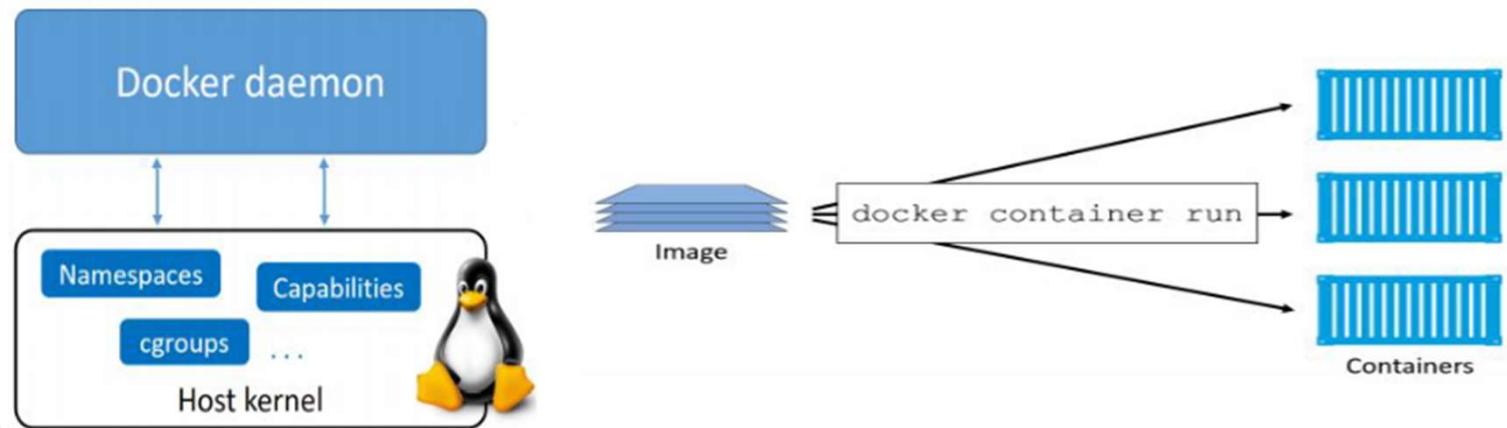
Docker Engine/Docker Daemon

- ▶ **Docker Engine** is the infrastructure plumbing software that runs and orchestrates containers (VMWare admin -> like ESXi).
- ▶ Docker Engine is modular in design with many swappable components. Where possible, these are based on open-standards outlined by the Open Container Initiative (OCI).
- ▶ Docker Engine is made from many specialized tools APIs, execution driver, runtime, shims etc.
- ▶ All other Docker, Inc. And 3rd party products plug into the Docker Engine and build around it.



Containers

- ▶ **Container** is the runtime instance of an image. In the same way that we can start a VM from Virtual Machine template.
- ▶ Run until the App they are executing exits and share the OS/kernel with the host they're running on.



Containers Commands

Docker container run -it ubuntu /bin/bash	Start an Ubuntu container in the foreground, and tell it to run the Bash shell
[Ctrl + PQ]	Detach your shell from the terminal of a container and leave the container running (UP) in the background
Docker container ls	Lists all containers in the running (UP) state. -a flag you will also see containers in the stopped (Exited) state
Docker container exec -it <container-name or container-id> bash	Let's you run a new process inside of a running container. This command will start a new Bash Shell inside of a running container and connect to it.
Docker container stop <container-name or container-id>	Stop a running container and put it in the Exited(0) state
Docker container start <container-name or container-id>	Restart a stopped (Exited) container
Docker container rm <container-name or container-id>	Delete a stopped container
Docker container inspect <container-name or container-id>	Show detailed configuration and runtime information about a container

Docker Images

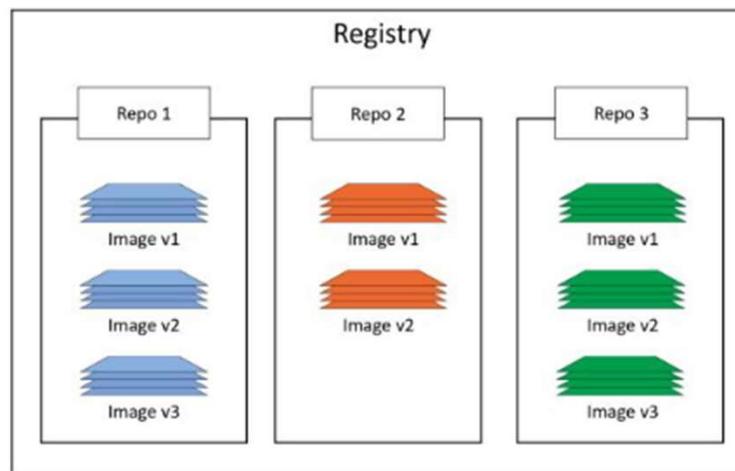
- ▶ **Docker images** is like VM templates (Admin)
- ▶ Docker images is like classes (Developer)
- ▶ Docker Images are considered build-time constructs, whereas containers are run-time constructs



- ▶ You start by pulling images from an **image registry** (**Docker Hub**)
- ▶ The **pull operation** downloads the image to your **local Docker host** where you can use it to start one or more Docker containers.
- ▶ **Images** are made up of multiple layers that get stacked on top of each other and represented as a single object.
- ▶ Inside of the image is a cut-down operating (OS) and all of the files and dependencies required to run an application
- ▶ Containers are intended to be fast and lightweight; images tend to be small.

Docker Image Registries

- ▶ Docker images are stored in **image registries**. The most common registry is **Docker Hub** (<https://hub.docker.com>)
- ▶ The Docker client is opinionated and defaults to using Docker Hub.
- ▶ Image registries contain multiple image repositories



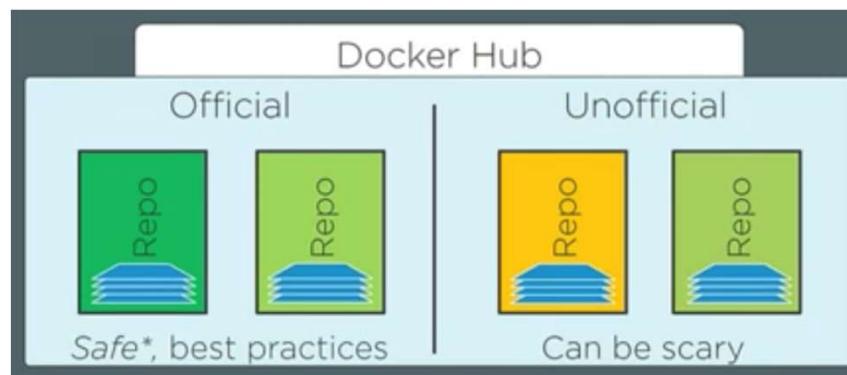
Docker Image Registries

Docker Hub also has the concept of **official repositories** and **unofficial repositories**

- ▶ **Official repositories** : contain images that have been vetted by Docker. Inc.

Examples : nginx (https://hub.docker.com/_/nginx/), mongoDB (https://hub.docker.com/_/mongo/)

- ▶ **Unofficial repositories** : you should not expect them to be safe, well-documented or built according to best practices.



Docker Image naming and tagging

- ▶ Images from official repositories are as simple as giving the repository name and tag separated by a colon(:). **Docker image pull <repository>:tag**
- ▶ If you do not specify the image tag, Docker will assume you are referring to the image tagged as **latest**
- ▶ Image is tagged as a latest does **not guarantee** it is the most recent image in a repository.
- ▶ A single image can have as many tags as you want

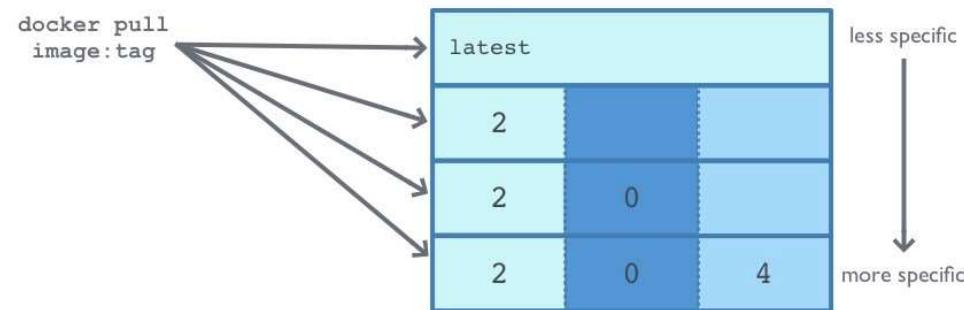


Image and layers (1/2)

- ▶ A docker image is just a bunch of loosely-connected read-only layers
- ▶ Docker takes care of stacking these layers and representing them as a single unified object

To see the layers of an image, you can inspect the image with the docker image inspect command

```
$ docker image inspect gilbertfongan/newflaskapp
[{"Id": "sha256:12d37cae3628e78ba13b96c0fde2a1fedd59f6afb1ff10bb4880f971199927c2",
"RepoTags": [
    "gilbertfongan/newflaskapp:latest"
],
<snip>
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:4942a1abcbfa1c325b1d7ed93d3cf6020f555be706672308a4a4a6b6d631d2e7",
        "sha256:5395b1dd90da45239b11b2b05354e0e1429aa312cb4a998e517810e831cbeedf",
        "sha256:b0fee53fe4bff191ae47e70eeeec4615a54d7b3a0f0e655604993bba83f8f72fb",
        "sha256:1bbae6f1e7fc51dba8a83df090afa3723cdee50c95490b0c09cb7f0060d0d215"
    ]
},
"Metadata": {
    "LastTagTime": "0001-01-01T00:00:00Z"
}
}]
```

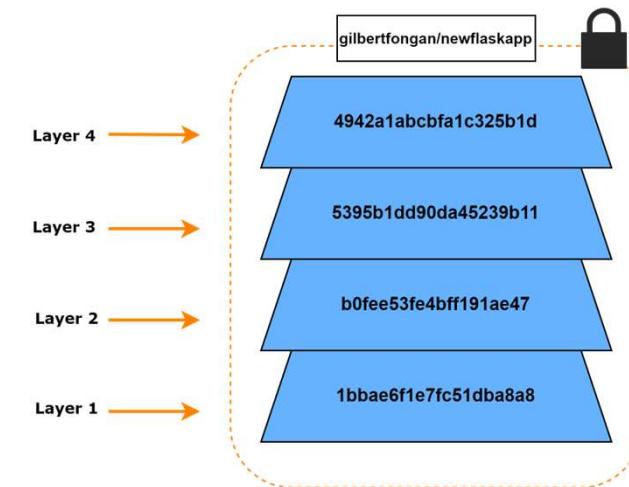


Image and layers (2/2)

- ▶ All Docker images start with a base layer, and as changes are made and new content is added, new layers are added on top.
- ▶ Multiple images can, and do, share layers. This leads to efficiencies in space and performance.

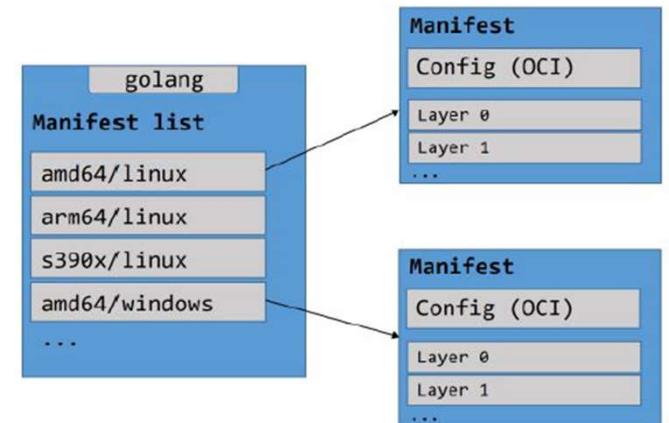
These lines tell us that Docker is smart enough to recognize when it's being asked to pull an image layer that it already has a copy of/

```
root@node1:/home/vagrant# docker pull -a gilbertfongan/newflaskapp
latest: Pulling from gilbertfongan/newflaskapp
Digest: sha256:851d8a8308443e2c22a70ff42ed1ddf69516b9166ee8b71313f7f2f17a38204d
Status: Image is up to date for gilbertfongan/newflaskapp
docker.io/gilbertfongan/newflaskapp
```

```
root@node1:/home/vagrant# docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
nginx              latest   5d58c024174d  4 days ago   142MB
gilbertfongan/newflaskapp  latest   12d37cae3628  13 months ago  438MB
```

Multi-architecture images

- ▶ A single **image (repository:tag)** can have an image for Linux on x64, Linux on PowerPC, Windows x64, ARM etc.
- ▶ To make this happen, the Registry API supports two important constructs:
 - **Manifest lists** : a list of architectures supported by a particular image tag. Each supported architecture then has its own “manifest detailing the layers it’s composed from.
 - **Manifests** : containing image config and layer data

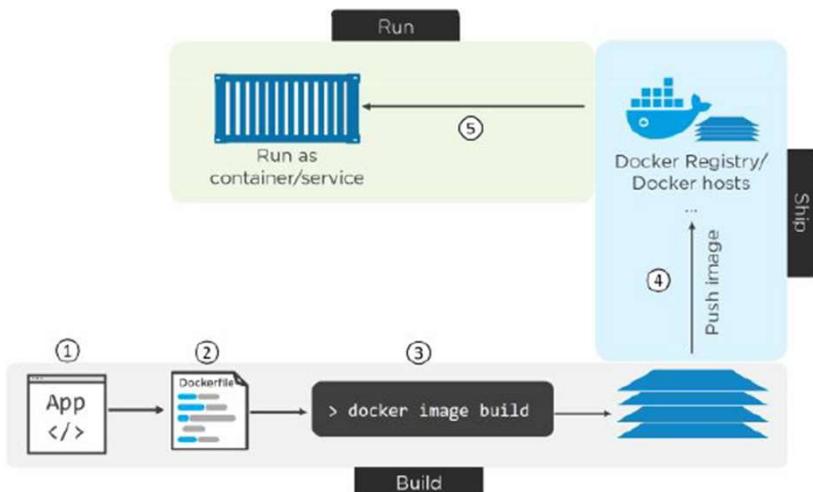


Containers image Commands

Docker image pull <code><image_name>: <image_tag></code>	The command to download images. By default, images will be pulled from repositories on Docker Hub
Docker image ls	Lists all of the images stored in your Docker host's local cache
Docker image inspect <code><image_name>: <image_tag></code>	Gives all the details of an image layer data and metadata
Docker image rm <code><image_name>: <image_tag></code>	Delete an image
Docker rmi <code><image_name>: <image_tag></code>	Delete an image. It's impossible to delete an image associated with a container in the runnin(Up) or stopped (Excited) states

Containerizing an App

- ▶ The process of taking an application and configuring it to run as a container is called “**Containerizing**”, Sometimes we call it “**Dockerizing**”.
 - ▶ Containers are all about apps. They’re about making apps simple to build, ship, and run.
 - ▶ The process of **containerizing** an app looks like this :
1. Start with your application **code**.
 2. Create a **Dockerfile** that describes your app, its dependencies, and how to run it
 3. Feed this Dockerfile into the Docker image build command.
 4. Sit back while Docker builds your application into a Docker image and push it to the registry.
 5. Run and execute the container



Dockerfile

- ▶ **Dockerfile** is the blueprint that **describes the application** and tells Docker **how to build** it into an image
- ▶ The directory containing the application is referred to as the **build context**
- ▶ It's a **common practice** to keep your Dockerfile in the root directory of the build context
- ▶ Dockerfile starts with a capital "D" and is all one word "**Dockerfile**"
- ▶ It can help bridge the gap between development and operations
- ▶ Should be **treated as code**, and checked into a source control system
- ▶ If an instruction is adding **new content** such as files and programs to the image, it will **create a new layer**. If it is **adding instructions** on how to build the image and run the application, it will **create metadata**.

Dockerfile Instructions

INSTRUCTION	DESCRIPTION
FROM	First instruction in Dockerfile and it identifies the image to inherit from
MAINTAINER	Provides visibility and credit to the author of the image
RUN	Executes a Linux command for configuring and installing
ENTRYPOINT	The final script or application used to bootstrap the container, making it an executable application
CMD	Provide default arguments to the ENTRYPOINT using a JSON array format
LABEL	Name/value metadata about the image
ENV	Sets environment variables
COPY	Copies file into the container
ADD	Alternative to copy
WORKDIR	Sets working directory for RUN, CMD, ENTRYPOINT, COPY, and/or ADD instructions
EXPOSE	Ports the container will listen on
VOLUME	Creates a mount point
USER	User to run RUN, CMD, and/or ENTRYPOINT instructions

Dockerfile Example

- ▶ Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image

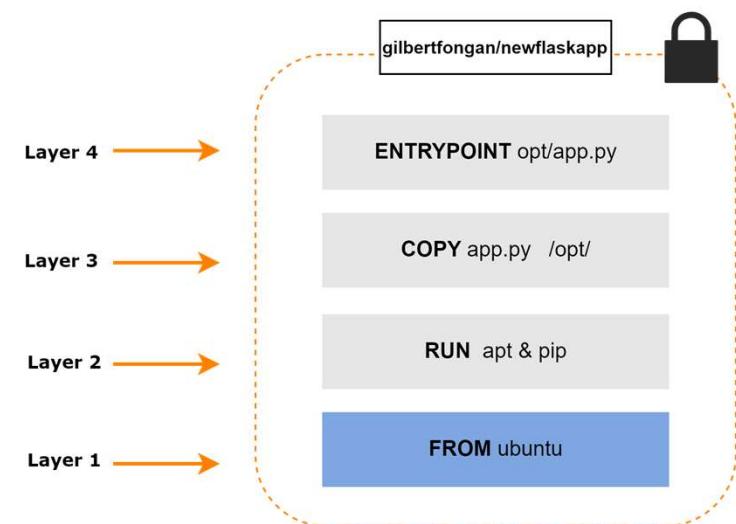
```
$ cat Dockerfile

FROM ubuntu

RUN apt-get update && apt-get install -y python3 pip
&& pip install flask

COPY app.py /opt/

ENTRYPOINT FLASK_APP=/opt/app.py flask run --
host=0.0.0.0 --port=8080
```



Docker Build image

- The Docker build command builds Docker images from a **Dockerfile** and a “**context**”

```
# cat > Dockerfile
FROM ubuntu
RUN apt-get update && apt-get install -y python3 pip
RUN pip install flask
COPY app.py /opt/
ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0 --port=8080

root@node1:/home/vagrant/flaskapp# ls
app.py  Dockerfile

# docker build . -t ACCOUNT_ID/IMAGE_NAME
root@node1:/home/vagrant/flaskapp# docker build . -t gilbertfongan/newflaskapp
Sending build context to Docker daemon 3.584kB
Step 1/5 : FROM ubuntu
--> fb52e22af1b0
Step 2/5 : RUN apt-get update && apt-get install -y python pip
--> Using cache
--> 7fd0386c8e30
Step 3/5 : RUN pip install flask
--> Using cache
--> 9b782128bd9b
Step 4/5 : COPY app.py /opt/
--> Using cache
--> a564d04cf8cc
Step 5/5 : ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0 --port=8080
--> Using cache
--> 12d37cae3628
Successfully built 12d37cae3628
Successfully tagged gilbertfongan/newflaskapp:latest
```

Docker Push image

Once we've created an image, it's time to store it in an image registry to keep it safe and make it available to others

- ▶ To push an image to Docker Hub, we need to login with the Docker ID

```
root@node1:/home/vagrant# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
Username: gilbertfongan
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

- ▶ Before we can push an image, we need to tag it in a special way (if we don't specify values of registry or tag, Docker will assume Registry=docker.io and Tag=latest)

```
# docker image tag  newflaskapp:latest gilbertfongan/newflaskapp:latest
```

- ▶ Now we can push it to Docker Hub

```
root@node1:/home/vagrant# docker push gilbertfongan/newflaskapp
Using default tag: latest
b0fee53fe4bf: Pushing [=====] 3.578MB/4.707MB
b0fee53fe4bf: Pushing [=====] 4.06MB/4.707MB
] 3.807MB/360.7MB
] 1.669MB/4.707MB
] 2.711MB/360.7MB
5395b1dd90da: Pushing [>
4942a1abcbfa: Mounted from library/ubuntu
```

Docker Push image

The screenshot shows a web browser displaying the Docker Hub repository page for the user gilbertfongan and the repository newflaskapp. The URL in the address bar is https://hub.docker.com/repository/docker/gilbertfongan/newflaskapp.

The page header includes a message about Docker product subscriptions, navigation links for Explore, Repositories, Organizations, Help, and an Upgrade button. The user's profile icon and name gilbertfongan are also visible.

The main content area shows the repository structure: gilbertfongan > Repositories > newflaskapp. It features tabs for General, Tags, Builds, Collaborators, Webhooks, and Settings, with the General tab currently selected.

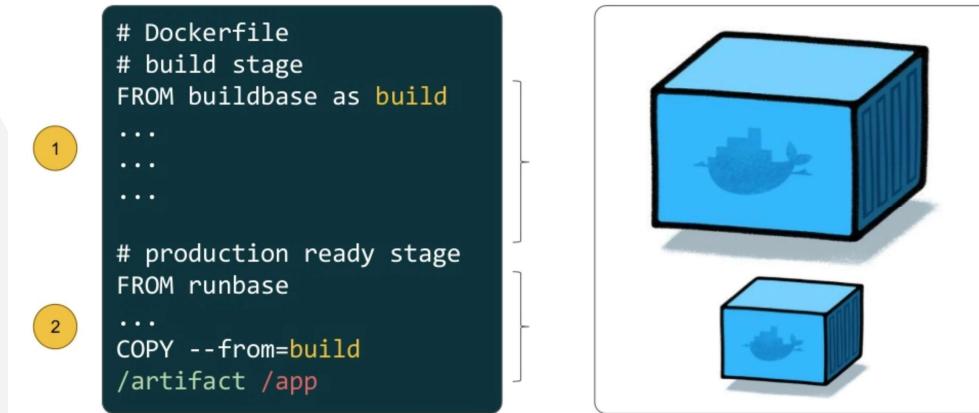
A callout box titled "Advanced Image Management" provides information on viewing all images and tags, cleaning up unused content, and recovering untagged images, noting it's available for Pro and Team accounts. A "View preview" link is also present.

The repository details section shows the name gilbertfongan / newflaskapp, a note that the repository does not have a description, and a timestamp indicating it was last pushed 3 hours ago.

The "Docker commands" section contains the command to push a new tag: docker push gilbertfongan/newflaskapp:tagname. A "Public View" button is located next to this section.

Dockerfile/Multi-stage Builds

- ▶ Docker images with complexity and big Instructions are bad => **More potential vulnerabilities** and possibly a **bigger attack surface**
- ▶ Multi-stage builds are all about optimizing builds without adding complexity.
- ▶ With **multi-stage** builds, we have a **single Dockerfile** containing multiple **FROM instructions**. Each FROM instruction is a **new build stage** that can easily COPY artefacts from **previous stages**.



Dockerfile/Multi-stage Builds EXAMPLE

```
FROM node:latest AS storefront
WORKDIR /usr/src/atsea/app/react-app
COPY react-app .
RUN npm install
RUN npm run build

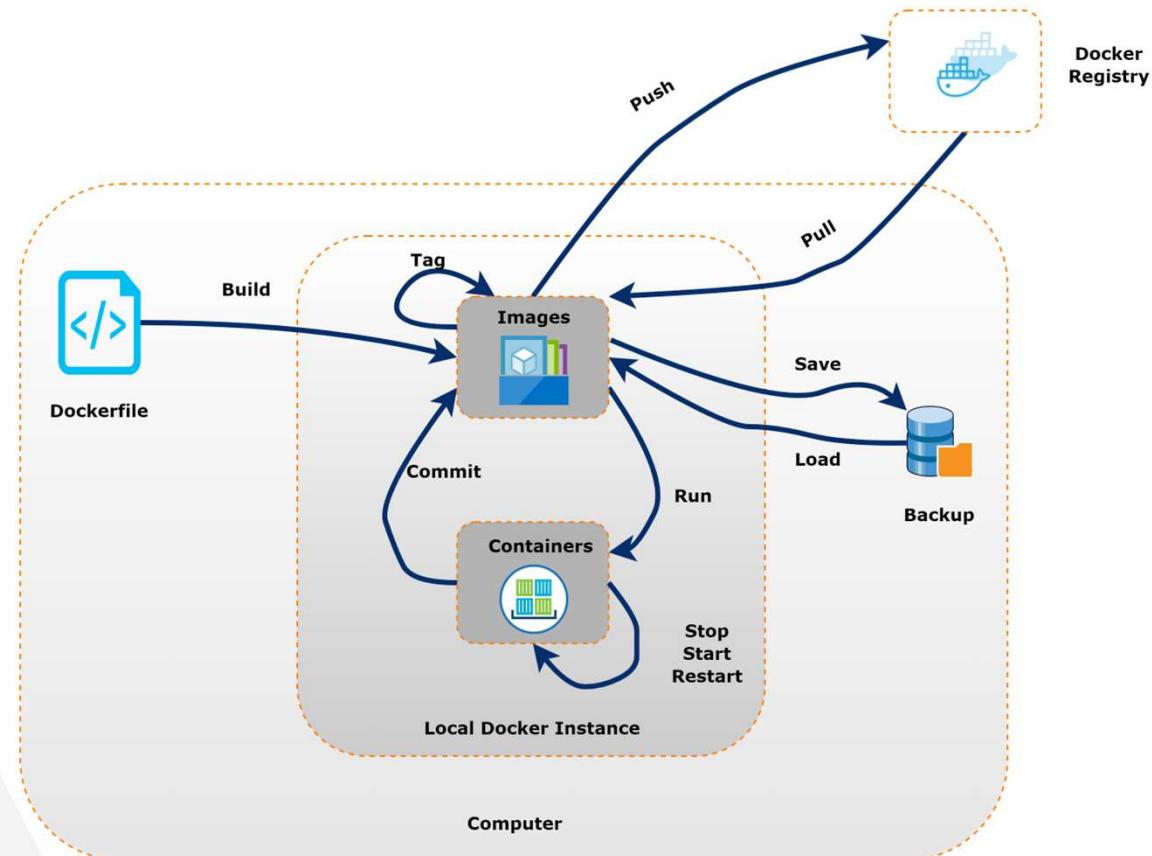
FROM maven:latest AS appserver
WORKDIR /usr/src/atsea
COPY pom.xml .
RUN mvn -B -f pom.xml -s /usr/share/maven/ref/settings-docker.xml dependency:resolve
COPY ..
RUN mvn -B -s /usr/share/maven/ref/settings-docker.xml package -DskipTests

FROM java:8-jdk-alpine
RUN adduser -Dh /home/gordon gordon
WORKDIR /static
COPY --from=storefront /usr/src/atsea/app/react-app/build/ .
WORKDIR /app
COPY --from=appserver /usr/src/atsea/target/AtSea-0.0.1-SNAPSHOT.jar .
ENTRYPOINT ["java", "-jar", "/app/AtSea-0.0.1-SNAPSHOT.jar"]
CMD ["--spring.profiles.active=postgres"]
```

Containerizing an App Commands

Commands	DESCRIPTION
Docker image build -t <repository_name>:<tagname> <build_context>	Command that reads a Dockerfile and containerizes the application. -t flag tags the image -f flag lets you specify the name and location of the Dockerfile
Docker image push <repository_name>:<tagname>	Push containerized app to image registry (by default Docker Hub)
Docker login [OPTIONS][SERVER]	Log in into a Docker registry
Docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Docker workflow



Docker Usage Sheet

Container Lifecycle

<code>Docker create <image></code>	Create a container without starting it
<code>Docker rename <CONTAINER_NAME> <NEW_CONTAINER_NAME></code>	Rename a container
<code>Docker run <IMAGE></code>	Create and start a container
<code>Docker run -rm <IMAGE></code>	Remove a container after it stops
<code>Docker run -td <IMAGE></code>	Start a container and keep it running
<code>Docker run -it <IMAGE></code>	Create, start the container, and run a command in it
<code>Docker run -it-rm <IMAGE></code>	Create, start the container, and run a command in it; after executing, the container is removed
<code>Docker rm <CONTAINER></code>	Delete a container if it's running
<code>Docker update <CONTAINER></code>	Update the configuration of a container

Image Lifecycle

<code>Docker build <URL></code>	Create an image from a Dockerfile
<code>Docker build -t <URL></code>	Build an image from a Dockerfile and tags it
<code>Docker pull <IMAGE></code>	Pull an image from a registry
<code>Docker push <IMAGE></code>	Push an image to a registry
<code>Docker import <URL/FILE></code>	Create an image from a tarball
<code>Docker commit <CONTAINER> <NEW_IMAGE_NAME></code>	Create an image from a container
<code>Docker rmi <IMAGE></code>	Remove an image
<code>Docker save <IMAGE> > <TAR_FILE></code>	Save an image to a tar archive stream to stdout with all parent layers, tags and versions
<code>Docker load <TAR_FILE/STDIN_FILE></code>	Load an image from a tar archive as stdin

Docker Usage Sheet

Start & Stop

Docker start <CONTAINER>	Start a container
Docker stop <CONTAINER>	Stop a running container
Docker restart <CONTAINER>	Start a running container and start it up again
Docker pause <CONTAINER>	Pause processes in a running container
Docker unpause <CONTAINER>	Unpause processes in a container
Docker wait <CONTAINER>	Block a container until other containers stop
Docker kill <CONTAINER>	Kill a container by sending SIGKILL to a running container
Docker attach <CONTAINER>	Attach local standard input, output, and error streams to a running container

Networking

Docker network ls	List of networks
Docker network rm <NETWORK>	Remove one or more networks
Docker network inspect <NETWORK>	Show information on one or more networks
Docker network connect <NETWORK> <CONTAINER>	Connect a container to a network
Docker network disconnect <NETWORK> <CONTAINER>	Disconnect a container from a network

Information

Docker ps	List running containers
Docker ps -a	List running and stopped containers
Docker logs <CONTAINER>	List the logs from a running container
Docker inspect <OBJECT_NAME/ID>	List low-level information on an object
Docker events <CONTAINER>	List real time events from a container
Docker port <CONTAINER>	Show port (or specific) mapping from a container
Docker top <CONTAINER>	Show running processes in a container
Docker stats <CONTAINER>	Show live resource usage statistics of containers
Docker diff <CONTAINER>	Show changes to files (or directories) on a filesystem
Docker image ls	Show all locally stored images
Docker history <IMAGE>	Show history of an image

MODULE III-2 : Container Deployment and Orchestration

PLAN

- ▶ **DOCKER COMPOSE**
 - Compose file
 - Deploy
 - Commands
- ▶ **DOCKER NETWORKING**
 - Drivers
 - Service Discovery
 - Commands

▶ **DOCKER SWARM:**

- Swarm Cluster
- Swarm services
- Swarm network mode
- Commands

▶ **DOCKER STACKS**

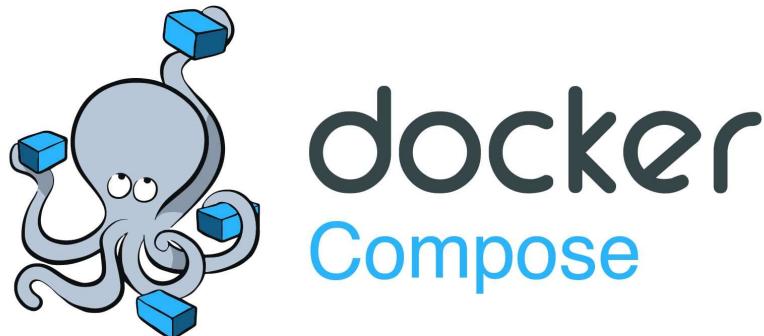
▶ **KUBERNETES**

- Architecture
- Pods
- Services
- Deployment
- Commands



Docker Compose

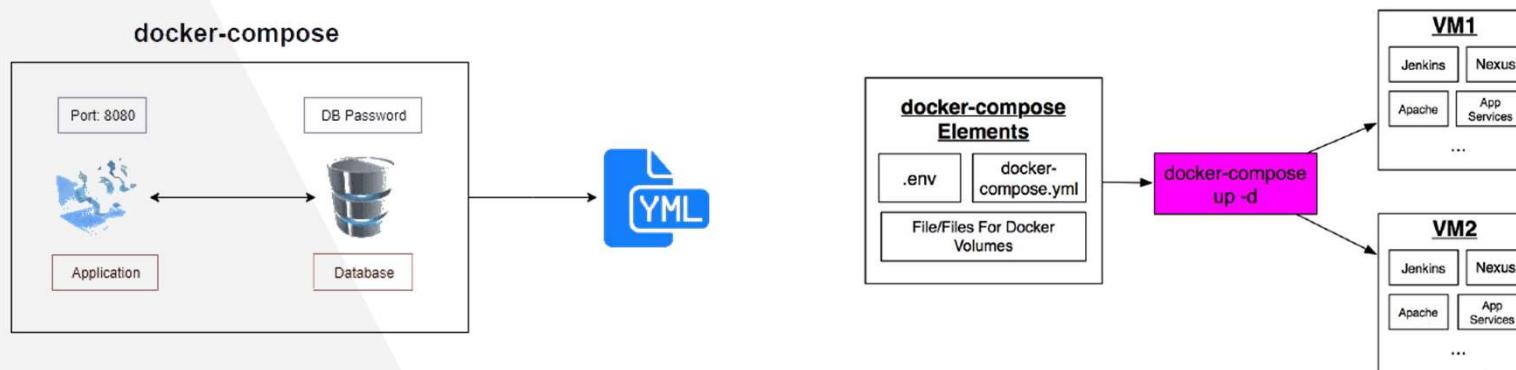
- ▶ In the beginning was Fig, created by Orchard
- ▶ In 2014, Docker Inc. acquired Orchard and re-branded Fig as Docker Compose
- ▶ Compose still an external Python binary that you have to install on a host running the Docker Engine.
- ▶ **Docker Compose** is a tool for defining and running multi-container Docker applications.
- ▶ Compose use a YAML file to configure application's services
- ▶ Then , with a single command , you create and start all the services from your configuration



Docker Compose

Docker Compose define multi-container (multi-service) apps in a **YAML** file, pass the YAML file to the docker-compose binary, and Compose deploys it through the Docker Engine API.

- ▶ Docker Compose lets you describe an entire app in a single declarative configuration file
- ▶ Deploy an entire app with a single command (**docker-compose up**)
- ▶ Once the app is deployed, you can manage its entire life cycle with a simple set of commands



Compose File

The example shown on the right shows a simple Compose file that defines a small **Flask app** with two services, a networks and a volumes [Simple web server that counts the number of visits and stores the value in Redis].

- ▶ Compose uses YAML files to define **multi-service applications**
- ▶ YAML is a **superset of JSON** so any JSON file should be valid YAML
- ▶ The default name for the Compose YAML file is “***docker-compose.yml***”
- ▶ A custom filenames can be specified with the use of **-f flag**

The Docker Compose file has 4 top-level keys :

- **Version**
- **Services**
- **Networks**
- **Volumes**

```
version: "3.5"
services:
  web-fe:
    build: .
    ports:
      - target: 5000
        published: 5000
    networks:
      - counter-net
    volumes:
      - type: volume
        source: counter-vol
        target: /code
  redis:
    image: "redis:alpine"
    networks:
      counter-net:
networks:
  counter-net:
volumes:
  counter-vol:
```

Compose File

The structure of the Docker Compose File is as follows

- ▶ **Version** : The version of the Compose file format (API) [Which can be depending on the Docker Engine release]. This does not define the version of Docker Compose or the Docker Engine
- ▶ **Services** : Define the different application services (Compose will deploy each of these services as its own container).
- **Build** : Build a new image using the instructions in the **Dockerfile** in the current directory.
- **Command** : Run a command (For Example to run Python app as the main App in the container.)
- **Ports** : Map port **5000** inside the container (target) to port **5000** on the host (published)
- **Networks** : Which network to attach the service's container to. Network should be already defined in the networks top-level key.
- **Volumes** : Which volume to attach the service's container to. The Volume should be already defined in the volume top-level key.

Deploy with Compose (1/4)

Example with Flask App : <https://gitlab.com/GilbertFongan/devops-book-labs/-/tree/main/Module4-Docker/Docker-Ubuntu-VM/counter-app>

- ▶ **Dockerfile** : Describes how to build the image for the web-fe service
- ▶ **App.py** : Is the python Flask application code
- ▶ **Requirements.txt** : List the Python Packages required for the App.
- ▶ **Docker-compose.yml** : Describes how Docker should deploy the App.

```
root@node1:/home/vagrant# cd counter-app/
root@node1:/home/vagrant/counter-app# ls
app.py  docker-compose.yml  Dockerfile  README.md  requirements.txt
```

Deploy with Compose (2/4)

The following command can be used to execute a Docker Compose command to bring up a Compose App (Multi-container App defined in a Compose file) : **\$ docker-compose up**

- ▶ Builds all required images
- ▶ Creates all required networks
- ▶ Starts all required containers.

The Docker-compose up command uses the **Docker-compose.yml** or **Dockercompose.yml** files by default.

In case the name of the compose file is different, it is necessary to specify the **-f** flag to indicate the custom file.

\$ docker-compose -f my-custom-docker-compose-file.yml up

The **-d** flag can also be specified to bring the App up in the background : **\$ docker-compose up -d**

```
root@node1:/home/vagrant/counter-app# docker-compose up -d
Building web-fe
Sending build context to Docker daemon 81.41kB
Step 1/5 : FROM python:3.6-alpine
3.6-alpine: Pulling from library/python
a0d0a0d46f8b: Downloading [=====]
c11246b421be: Download complete
ef6741e6e9c4: Downloading [=>
9d6fa827d5ce: Waiting
665bc1c2019c: Waiting
] 601.3kB/2.814MB
] 310.5kB/10.2MB
```

Deploy with Compose (3/4)

- After running the **Docker-Compose** command on our repository, we can discover three images built or pulled as part of the deployment

```
root@node1:/home/vagrant/counter-app# docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
counter-app_web-fe  latest   e7d24725fb72  3 minutes ago  53.7MB
gilbertfongan/newflaskapp  latest   12d37cae3628  31 hours ago  438MB
python               3.6-alpine  c5917c34066a  2 weeks ago   40.8MB
ubuntu               latest   fb52e22af1b0  3 weeks ago   72.8MB
redis                alpine   f6f2296798e9  4 weeks ago   32.3MB
```

- The list of running containers is as follows (We can notice that the names of each containers is prefixed with the name of the project or name of the working directory.).

```
root@node1:/home/vagrant/counter-app# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
5e92100d256b        redis:alpine        "docker-entrypoint.s..."   About a minute ago   Up About a minute   6379/tcp
3f3280f05b7b        counter-app_web-fe   "python app.py"         About a minute ago   Up About a minute   0.0.0.0:500
0->5000/tcp, ::5000->5000/tcp   counter-app_web-fe_1
```

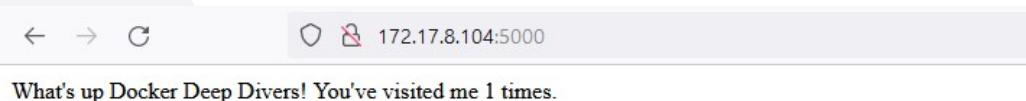
With the **scalability feature** of the compose service, each container has a numeric suffix that indicates the instance number.

Deploy with Compose (4/4)

- In addition to the services, Docker-compose also created the networks and volumes

```
root@node1:/home/vagrant/counter-app# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
3ce814c2603d    bridge    bridge      local
615c65b4a334    counter-app_counter-net  bridge      local
17b2b772f5e7    host      host       local
c364deaf96e8    none      null       local
root@node1:/home/vagrant/counter-app# docker volume ls
DRIVER      VOLUME NAME
local      3a16754d5c9ac1588efb2342dd94965dd6471bde703626bb0f6828bdbceb1f40
local      counter-app_counter-vol
```

- Thus, the application successfully deployed, it can be accessed



- To stop the Docker-Compose App without deleting the images and volumes created

```
root@node1:/home/vagrant/counter-app# docker-compose down
Stopping counter-app_redis_1 ... done
Stopping counter-app_web-fe_1 ... done
Removing counter-app_redis_1 ... done
Removing counter-app_web-fe_1 ... done
Removing network counter-app_counter-net
```

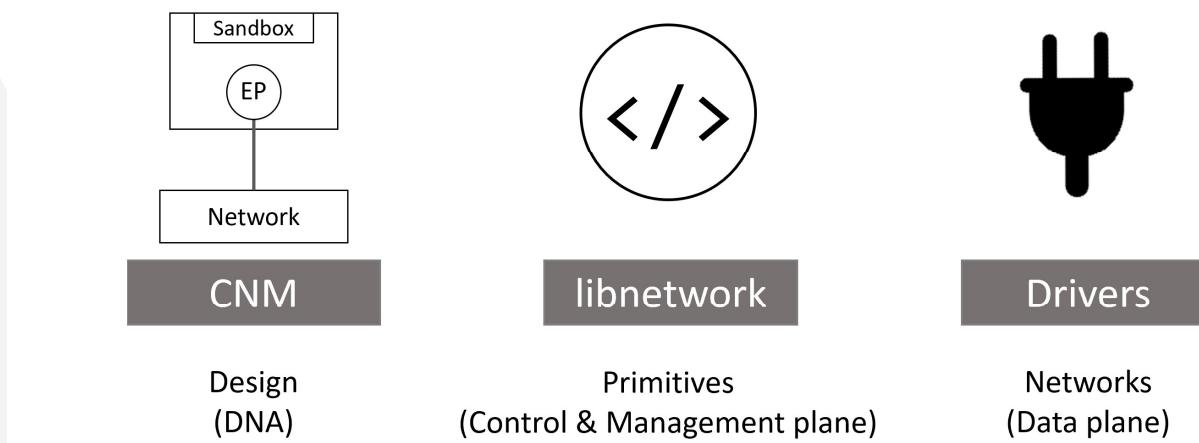
Docker Compose Commands

COMMANDS	DESCRIPTION
Docker-compose up	Deploy a Compose App. It expects the Compose file to be called <i>docker-compose.yml</i> or <i>docker-compose.yaml</i> , but you can specify a custom filename with the <i>-f</i> flag. It's common to start the App in the background with the <i>-d</i> flag
Docker-compose stop	Stop all containers in a Compose App without deleting them from the system
Docker-compose restart	Restart a Compose App that has been stopped with <i>docker-compose stop</i> . If you have made changes to your Compose App since stopping it, these changes will not appear in the restarted App. You will need to re-deploy the App to get the changes.
Docker-compose ps	List each container in the Compose App. It show the current state, the command each one is running, and network ports
Docker-compose down	Stop and delete a running Compose App. It deletes containers and networks, but not volumes and images.

Docker Networking

At the highest level, **Docker networking** comprises three major components :

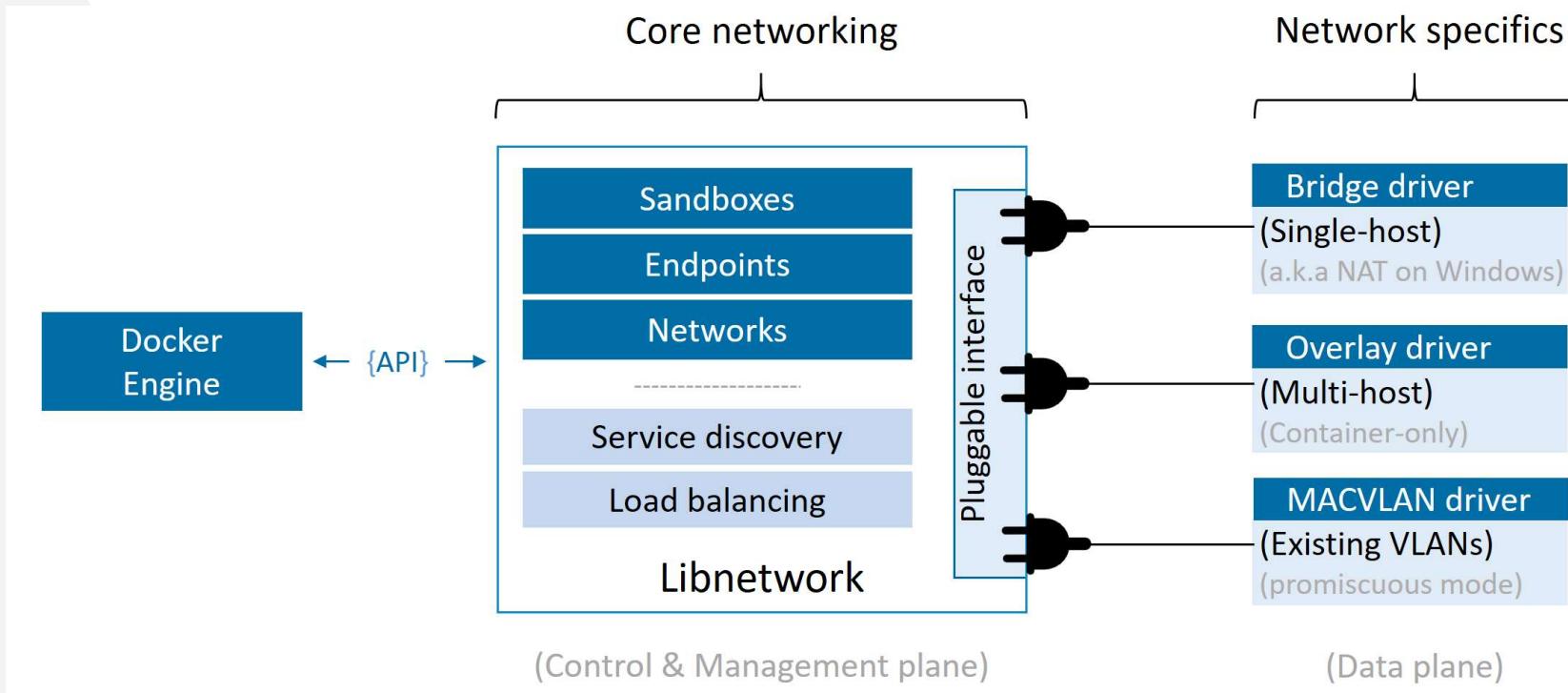
- ▶ The Container Network Model (CNM)
- ▶ Libnetwork
- ▶ Driver



Docker Networking

- ▶ Docker runs applications inside of containers, and these need to communicate over lots of different **networks**.
- ▶ Docker has solutions for **container-to-container networks**, as well as connecting to existing networks and **VLANs**.
- ▶ Docker networking is based on an open-source pluggable architecture called the **Container Network Model (CNM)**
- ▶ **Libnetwork** is Docker's real-world implementation of CNM, and it provides all of Docker's core networking capabilities.
- ▶ Drivers plug in to libnetwork to provide specific network topologies such as VXLAN overlay networks.

Docker Networking Drivers



Docker Networking Drivers

Docker ships with a set of native drivers that deal with the most common networking requirements :

- ▶ **Bridge (default)** : Containers in local **Docker0** bridge
- ▶ **Null** : Containers has no network interface
- ▶ **Host** : Containers use host's network interface
- ▶ **Overlay** : Multi-host
- ▶ **MACVLAN** : for existing VLANs

Null Network (None Driver)

- ▶ The **None driver** option for container networking disables the networking of a container while allowing the very same container to use a custom third-party network driver, if needed, to implement its networking requirements.
- ▶ None provides the functionality of disabling networking
- ▶ Form a container with none Network :

```
$ docker run --rm -dit \  
  --network none \  
  --name no-net-alpine \  
  alpine:latest \  
  ash
```

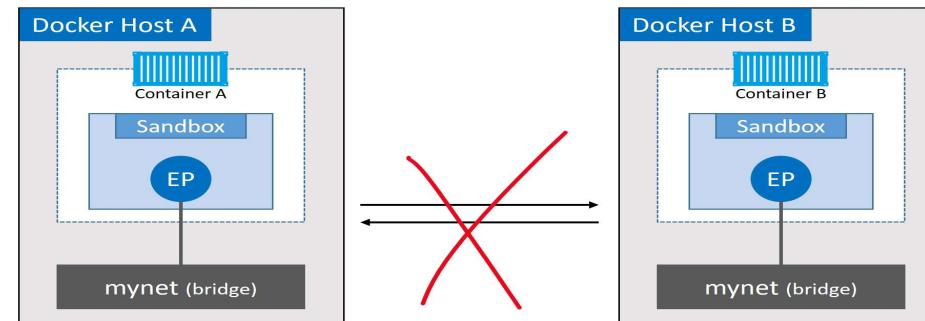
- ▶ Using '**-network none**' will result in a container with **no eth0**.

Single-host Bridge Network (1/4)

It is the simplest type of docker network.

- ▶ **Single-host** means it only exists on a single Docker host and can only connect containers that are on the same host
- ▶ Bridge means that it's an implementation of an **802.1d bridge** (layer 2 switch)

Docker on Linux creates single-host bridge networks with the built-in bridge driver, whereas Docker on Windows creates them using the built-in nat driver.



Single-host Bridge Network (2/4)

By default, every Docker host gets a single-host bridge network. On Linux it's called "**bridge**", and on Windows it's called "**Nat**". All new containers will attach to that network unless you override it on the command line with the **-network** flag

```
//Linux
$ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
333e184cd343    bridge    bridge      local

//Windows
> docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
095d4090fa32    nat       nat        local
```

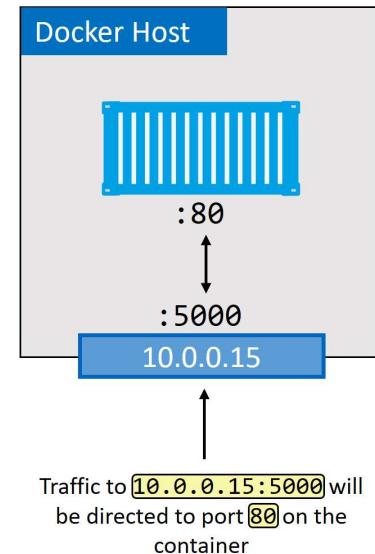
Command to create a new single-host bridge network called "**localnet**"

```
//Linux
$ docker network create -d bridge localnet

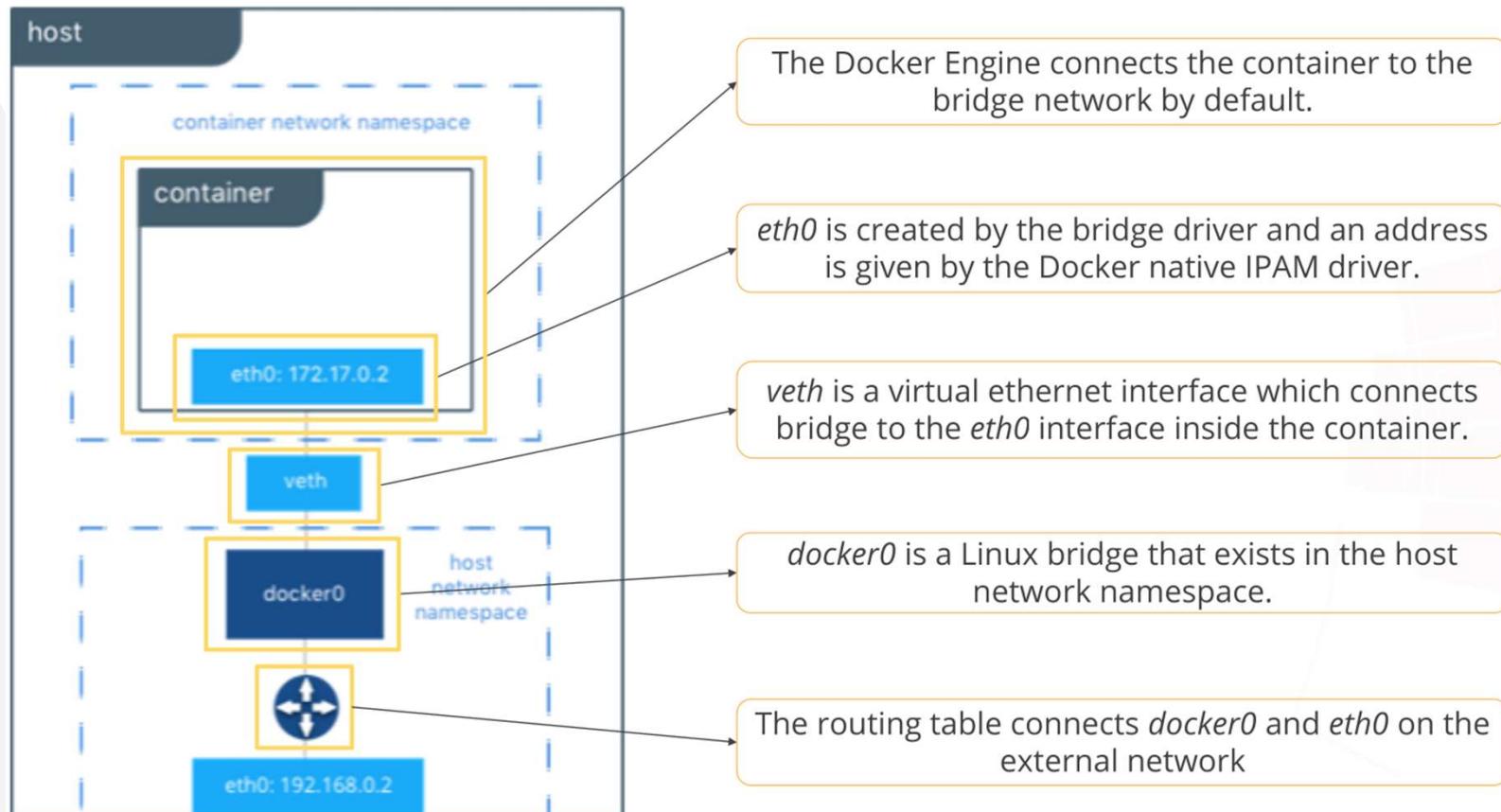
//Windows
> docker network create -d nat localnet
```

Single-host Bridge Network (3/4)

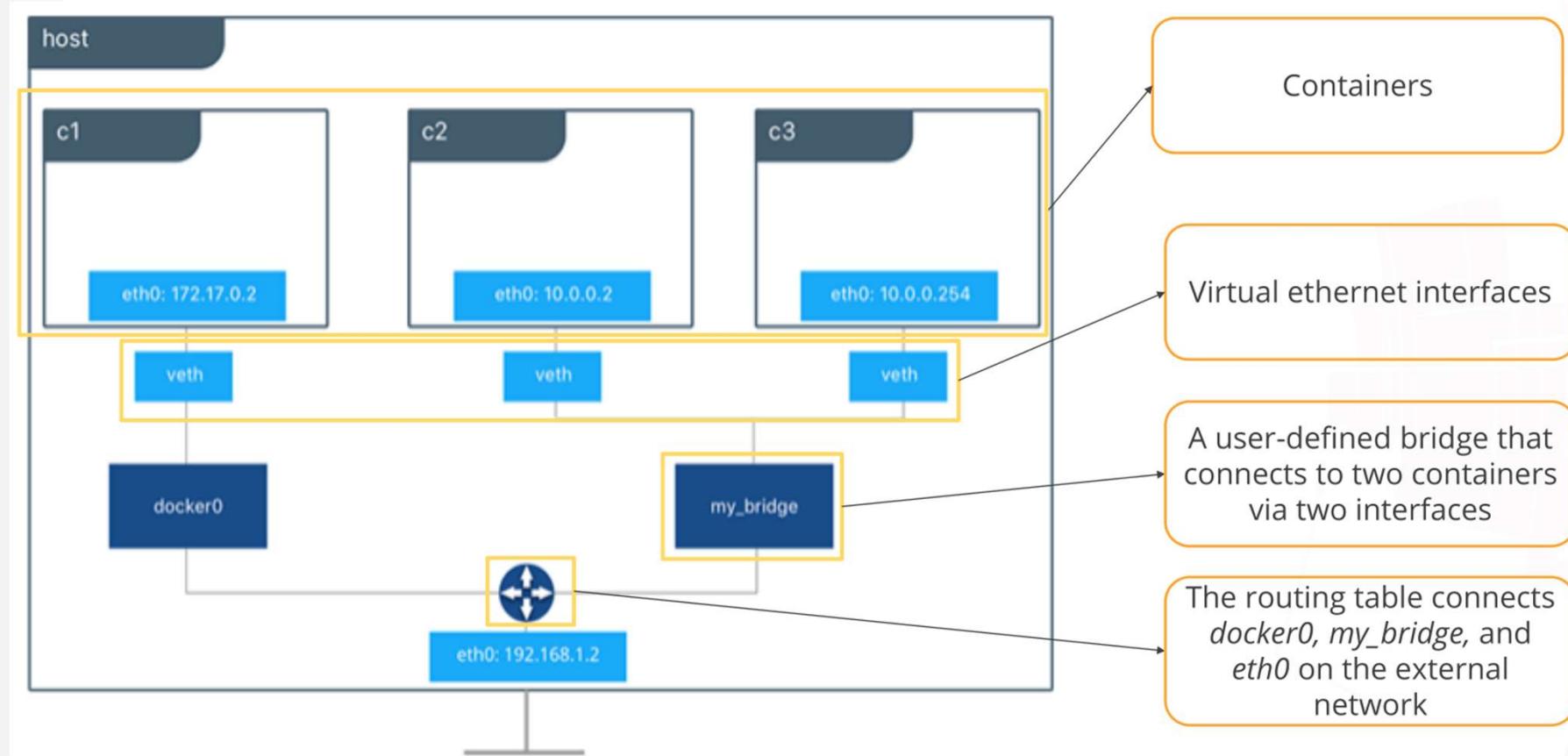
- ▶ Containers on Bridge networks can only communicate with other containers on the same network. However, you can get around this using **Port Mappings**.
 - ▶ **Port Mappings** map a container port to a specific port on the Docker Host. Any traffic hitting the Docker host on the configured port will be directed to the container.
-
- ▶ For this example, illustrated by this diagram, the application running in the container is operating on **Port 80**
 - ▶ All incoming traffic on the **Host 10.0.0.15** with port **5000** is mapped to **port 80** of the running container.



Single-host Bridge Network (4/4)



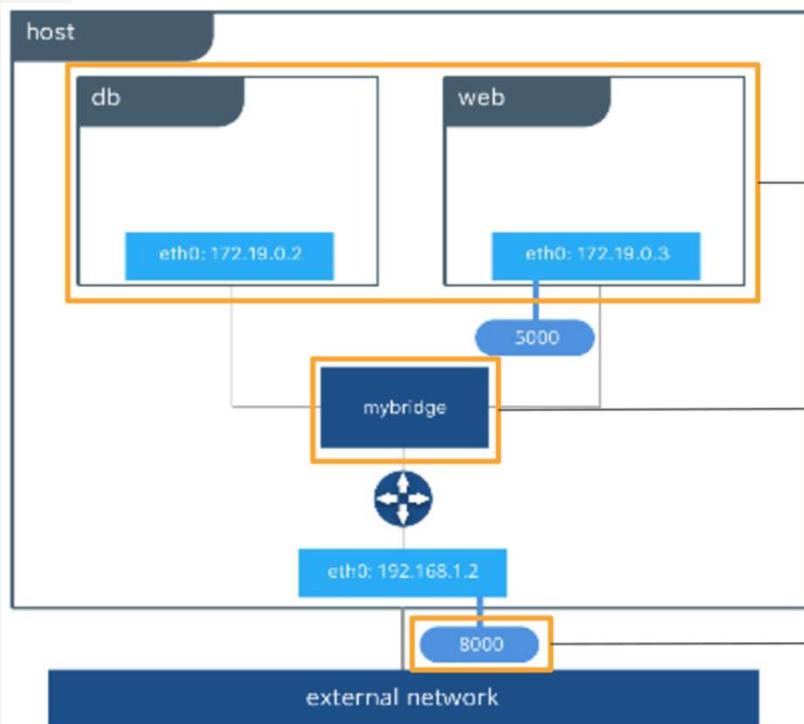
Single-host User-Defined Bridge Network



Single-host Bridge Network Comparison

Features	Default	User-Defined
Better isolation and interoperability between containerized applications	No	Yes
Automatic DNS resolution between containers	No	Yes
Attachment and detachment of containers on the fly	No	Yes
Configurable bridge creation	No	Yes
Linked containers share environment variables	Yes	No

Single-host Bridge Network Use Case



db and *web* are containers of an application called *pets*. This application is available on <host-ip>:8000.

mybridge is helping containers *web* to interact with *db* by its container name. This driver is a local scope driver.

An application *pets* is served on the host at port 8000.

Host Network

The **Host Network** driver option, as opposed to the bridge, eliminates the network isolation between the container and the host system by allowing the container to directly access the host network.

Features :

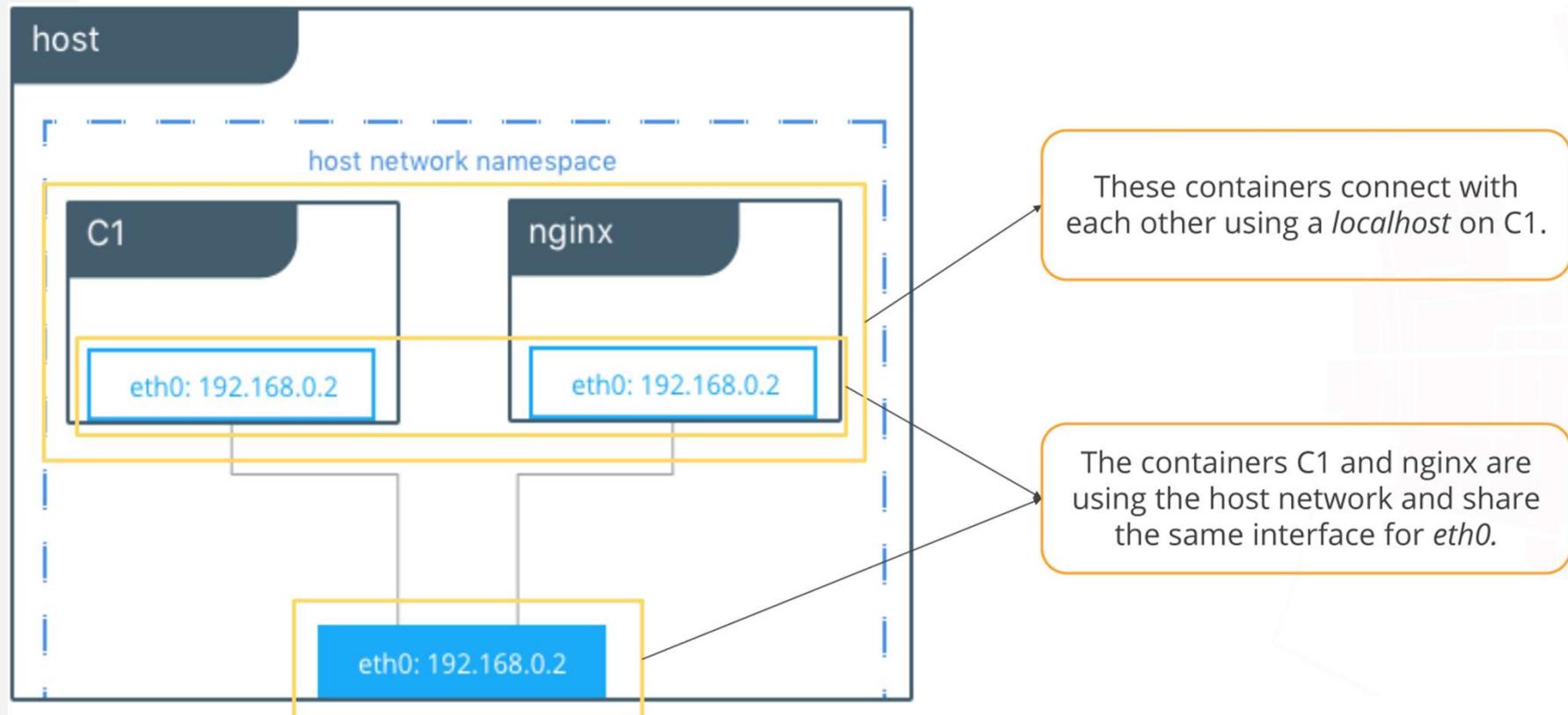
- ▶ An overlay network is used to manage swarm and service-related traffic
- ▶ Docker daemon Host network and ports are used to send data for individual swarm service

Advantages :

- ▶ Optimizes the performance (eliminating the need for NAT since the container ports are automatically published and available as host ports).
- ▶ Handles a large range of ports
- ▶ Does not require “userland-proxy” for each port.

NOTE : The host networking driver only works on Linux hosts

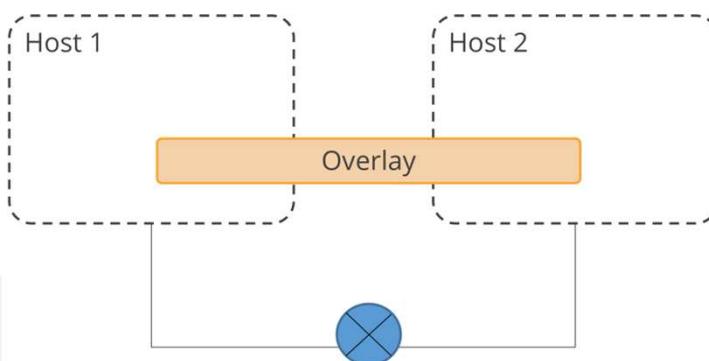
Host Network



Multi-Host Overlay Network (1/3)

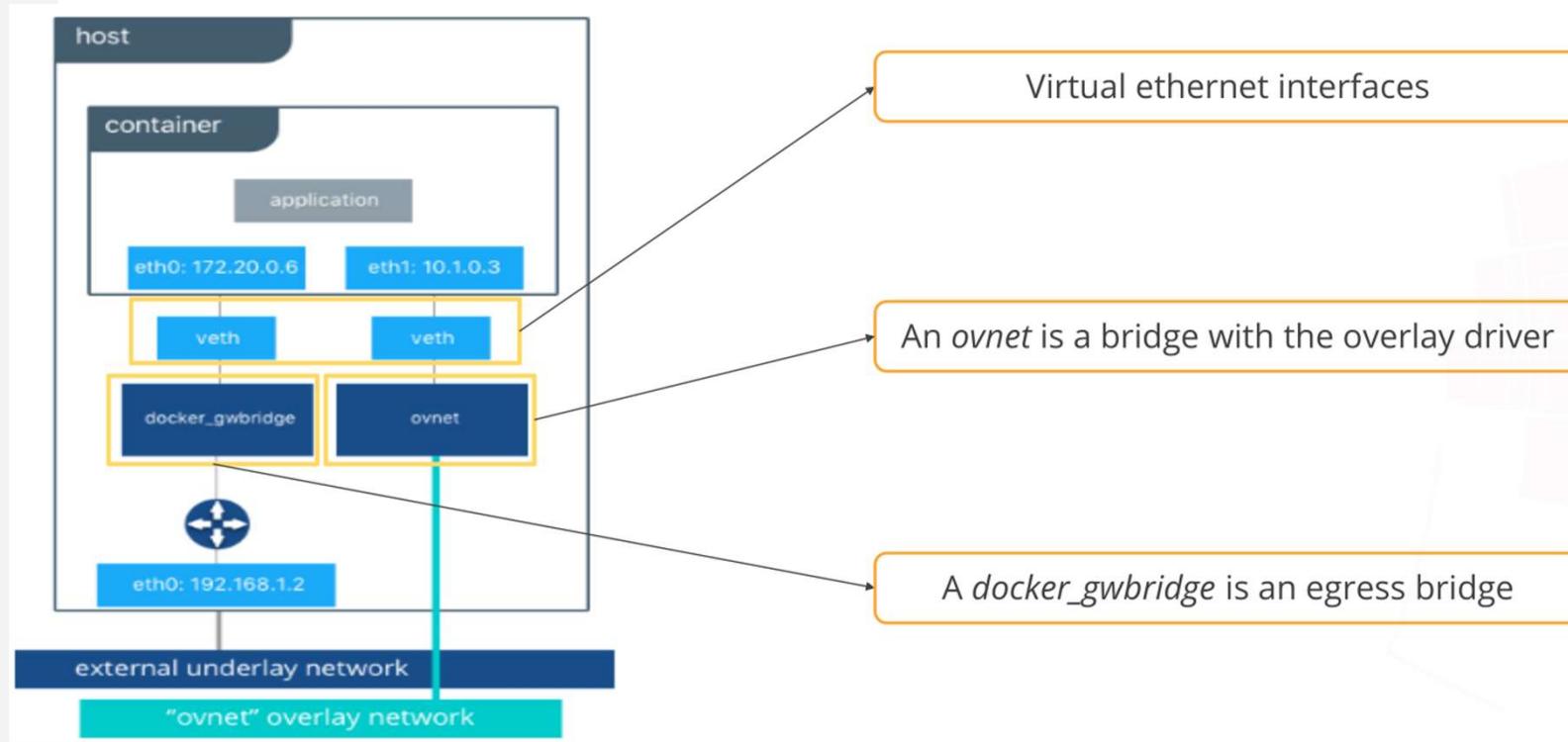
The **Multi-Host Overlay Network** driver option is a network type that spans multiple hosts, typically part of a cluster, allowing the containers traffic to be routed between hosts as containers or services from one host attempt to talk to others running on another host in the cluster..

- ▶ Allow to create a flat, secure, **layer-2 Network**, spanning multiple hosts.
- ▶ Docker provides a native driver for overlay networks. This makes creating them as simple as adding the **--d overlay** flag to the docker network create command.



Multi-Host Overlay Network (2/3)

Provisioning for an Overlay Network is automated by Docker Swarm Control Plane



Multi-Host Overlay Network (3/3)

- ▶ To create a new overlay network : `$ docker network create -d overlay devops`

```
root@node1:/home/vagrant# docker network create -d overlay devops
tkhh4fvf5o1vywp9mwwx4vjm7
```

- ▶ To list all networks on each node : `$ docker network ls`

NETWORK ID	NAME	DRIVER	SCOPE
8d3a7895a564	bridge	bridge	local
tkhh4fvf5o1v	devops	overlay	swarm
4a8eb79b81bd	docker_gwbridge	bridge	local
17b2b772f5e7	host	host	local
b1lxoak7r4nl	ingress	overlay	swarm
c364deaf96e8	none	null	local

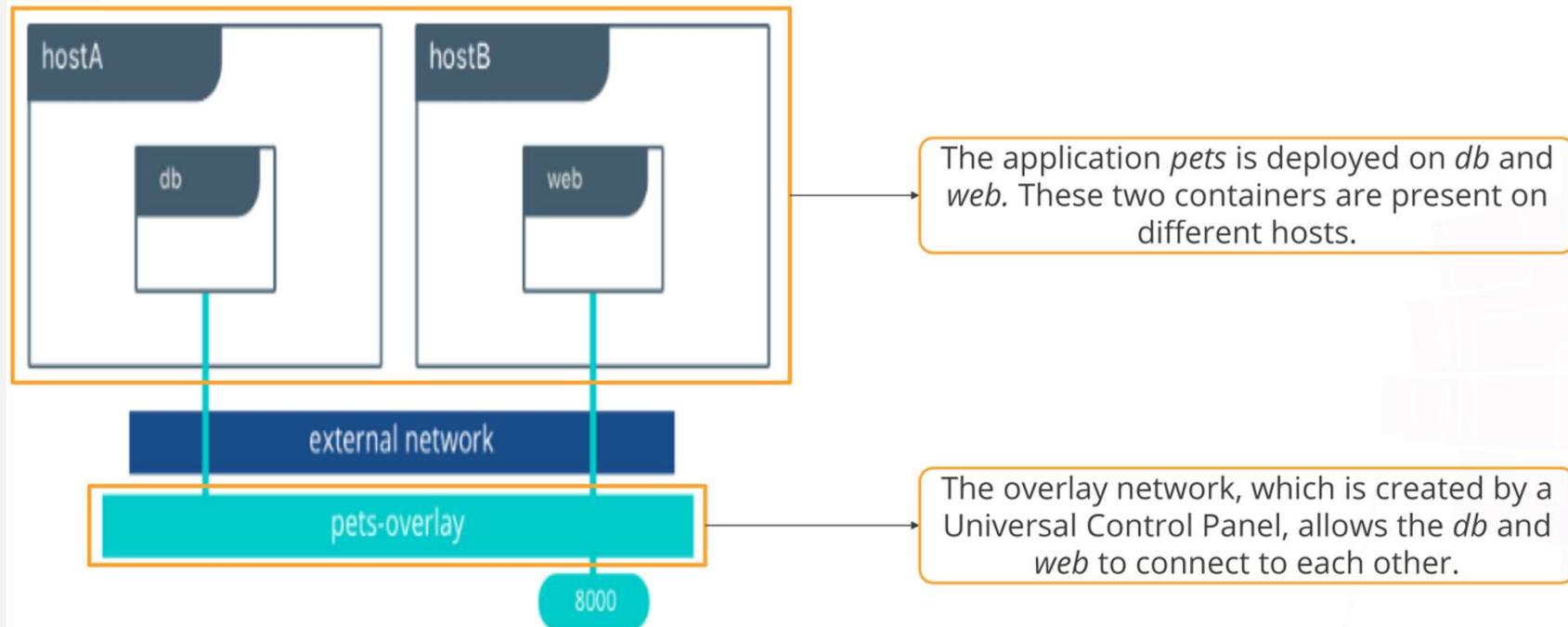
- ▶ To attach a service to the Overlay Network : `$ docker service create --name=test-devops --network devops -p 80:80`

```
gilbertfongan/demo:v1

root@node1:/home/vagrant# docker service create --name=test-devops --network devops -p 80:80 --replicas
12 gilbertfongan/demo:v1
jh9dv7mb4hkv8mo2w2gnchcnt
overall progress: 12 out of 12 tasks
1/12: running
2/12: running
3/12: running
4/12: running
5/12: running
6/12: running
7/12: running
8/12: running
9/12: running
10/12: running
11/12: running
12/12: running
verify: Service converged
```

- ▶ To inspect a created Overlay Network : `$ docker network inspect devops`

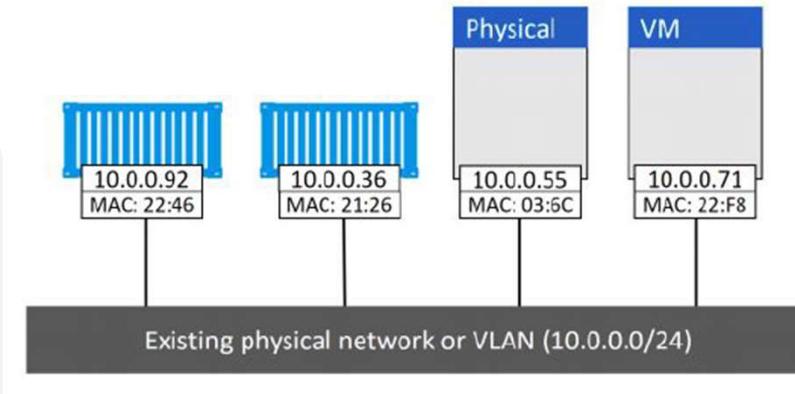
Multi-Host Overlay Network Use case



MAC VLAN (1/4)

The **MACVLAN Network** driver allows a user to change the appearance of a container on the physical network.

- ▶ A container may appear as a physical device with its own MAC address on the Network.
- ▶ The container is directly connected to the physical network instead of having its traffic routed through the Host Network.
- ▶ MACVLAN Network is used to assign MAC address to the virtual Network interface of containers



MAC VLAN (2/4)

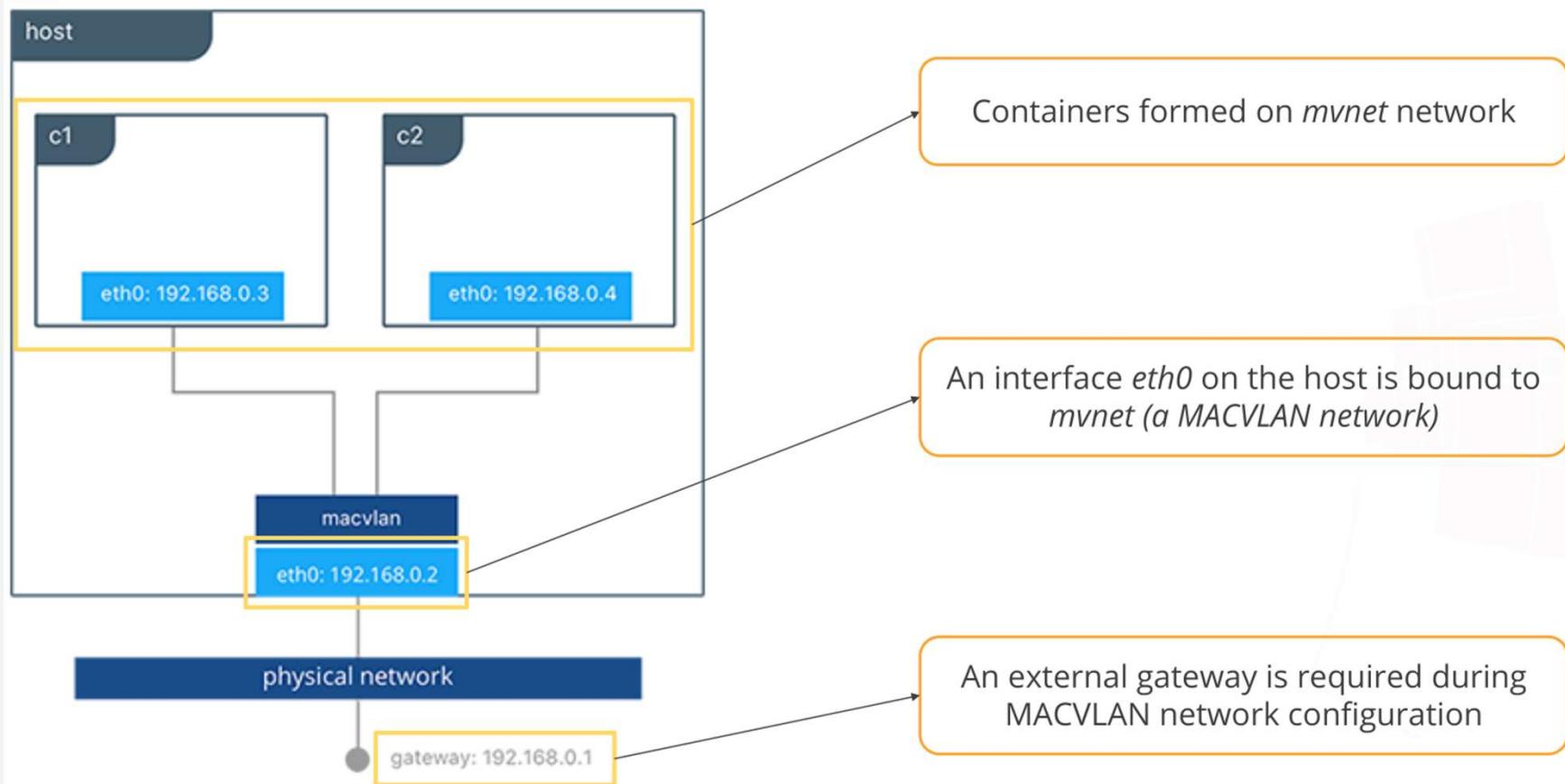
Advantages :

- ▶ Simple and lightweight architecture
- ▶ Direct access between physical Network and Containers
- ▶ Containers receive routable IP addresses that are present on the subnet of the physical Network

Precautionary measures :

- ▶ Cut down the large number of unique MAC to save the Network from damage
- ▶ Handle “promiscuous mode” which isn’t allowed on most public Cloud Platforms

MAC VLAN (3/4)



MAC VLAN (4/4)

- ▶ The MACVLAN driver needs these arguments about the Network : Subnet info, Gateway, Range of IP's it can assign to containers, Interface or Sub-interface on the Host to use.
- ▶ Create a new MACVLAN Network called "***macvlan5***" that will connect containers to VLAN5:

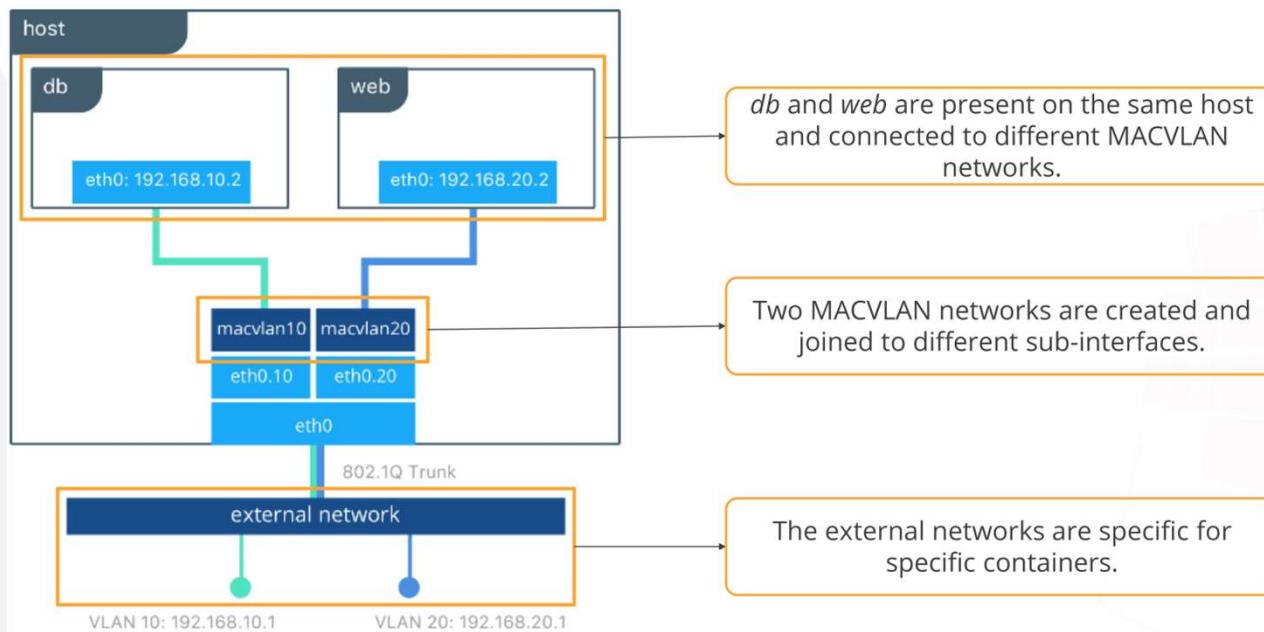
```
$ docker network create -d macvlan \
--subnet=172.16.0.0/24 \
--ip-range=172.16.0.0/25 \
--gateway=10.0.0.1 \
-o parent=eth0.5 \
macvlan5
```

- ▶ The MACVLAN5 Network is ready for containers. Create a container to deploy with the Network:

```
$ docker container run -d --name=test-macvlan --network macvlan5 alpine sleep 1d
```

MAC VLAN Use Cases

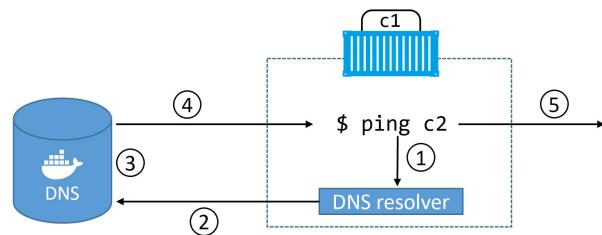
- ▶ Low-latency applications
- ▶ Network design which needs containers to be on the same subnet and use IPs as the external Host Network



Docker Network Service Discovery

Service discovery allows all **containers** and **Swarm** services to locate each other by name. The only requirement is that they have to be on the **same Network**.

This leverages Docker's embedded DNS Server and the DNS resolver in each container



1. Ping **c2** command invokes the local DNS resolver to resolve the name « **c2** ». All Docker containers have a **local DNS resolver**
2. In case that the local resolver doesn't have an IP address for « **c2** » in its local cache, it initiates a recursive query to the Docker DNS Server. The local resolver is pre-configured to know how to reach the Docker DNS server.
3. Docker DNS server holds name-to-IP mappings for all containers created with the `--name` or `--net-alias` flags. It knows the IP address of container « **c2** »
4. [Same Network] DNS resolver returns the IP address of « **c2** » to the local resolver in « **c1** »
5. The ping command is sent to the corresponding target IP address of « **c2** »

Docker Network Commands

COMMANDS	DESCRIPTION
Docker network ls	List all networks on the local Docker host.
Docker network create	Creates new Docker networks. By default, it creates them with the NAT driver on Windows and the Bridge driver on Linux. Driver type can be specified with the <code>-d</code> flag
Docker network inspect	Provides detailed configuration information about a Docker Network
Docker network prune	Deletes all unused networks on a Docker host.
Docker network rm	Deletes specific networks on a Docker host

Docker Swarm (1/5)

A **Swarm** is a term used to describe many Docker hosts or systems. So, **Docker Swarm** is a tool used for managing container configuration. The main features of Docker Swarm are :



- ▶ **Security** : Nodes Allow enforcement of encryption and mutual authentication to enhance Hight security in communications between nodes.
- ▶ **Scalability** : Automatic addition or removal of tasks that allow users to scale up or down as per their needs
- ▶ **Decentralized design** : Allow to create a Swarm from one disk image
- ▶ **Integration** : The cluster management has been integrated with Docker Engine. This allows users to manage swarms without requiring another orchestrations software.
- ▶ **Rolling updates** : Services updates on nodes can be made incrementally during rollout. In case of a problem, you can roll back to a previous safe service.
- ▶ **Declarative Service** : Allow to define the required state of a service
- ▶ **Service discovery** : Embedded DNS server can be used to query a container that runs within the Swarm.

Docker Swarm (2/5)

Docker Swarm is two (02) main things

- ▶ **Enterprise-grade secure cluster of Docker hosts** (Clustering)
- ▶ **Engine for orchestrating microservices apps** (Orchestration)

Clustering :

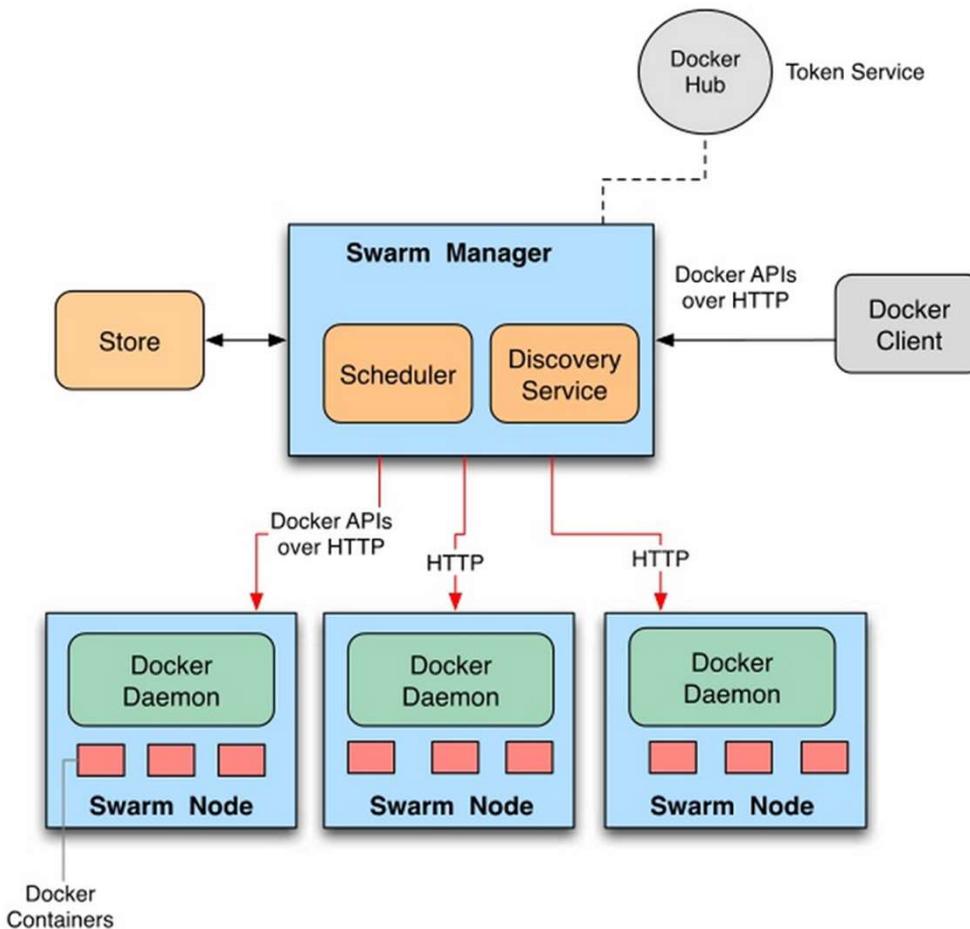


- ▶ Groups one or more Docker nodes and lets user manage them as a cluster
- ▶ Encrypted distributed cluster store and Network
- ▶ Mutual TLS and secure cluster join tokens
- ▶ PKI makes managing and rotating certificates as an easy task
- ▶ Add and remove nodes

Orchestration :

- ▶ Rich API exposed allow user to deploy and manage complex microservices Apps with ease
- ▶ Define Apps in declarative manifest files and deploy them with native Docker commands
- ▶ Rolling updates rollbacks and scaling operations.

Docker Swarm (3/5)

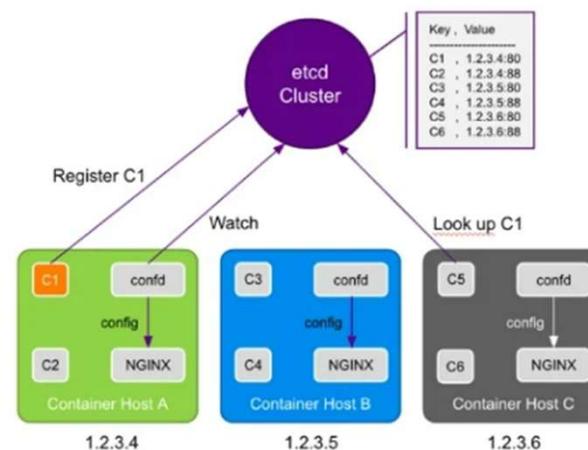


Docker Swarm (4/5)

- ▶ **Nodes are configured as managers or workers :**
- **Managers** : Control Plan of the Cluster – Manage the state of the cluster and dispatch tasks to workers
- **Workers** : Accept tasks from Managers and execute them

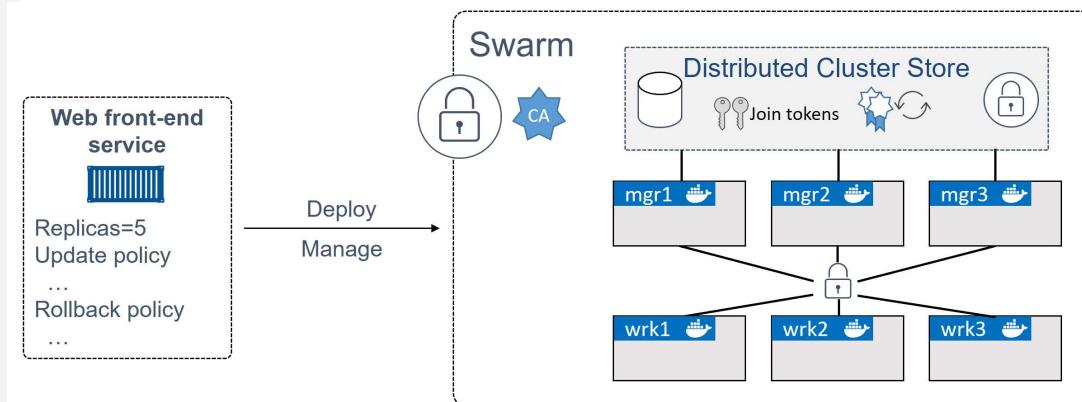
- ▶ **Configuration and state of the Swarm is held in a distributed etcd database located on all managers**

- Distributed K/V store based on directories
- Service definition queried using JSON-based HTTP APIs
- Clients handle failure or load balancing themselves
- Allows watch on changes



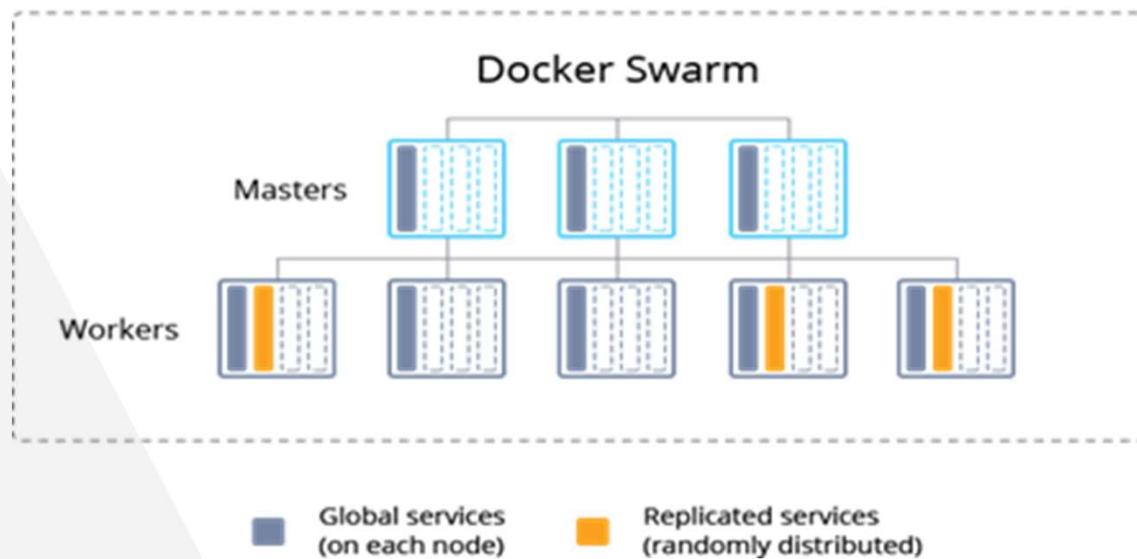
Docker Swarm (5/5)

- ▶ Swarm uses TLS to encrypt communications, authenticate nodes, and authorize roles:
- ▶ The atomic unit of scheduling on a Swarm is the service
- Service is a higher-level construct that wraps some advanced features around containers.
- A task or replica is a container wrapped in a service
- ▶ High-level view of Swarm cluster :



Swarm Mode

- ▶ **Replicated services (default)** : This deploys a desired number of replicas and distributes them as evenly as possible across the cluster.
- ▶ **Global services** : This runs a single replica on every node in the Swarm.



Build Swarm Cluster

To build a Swarm cluster with manager nodes and worker nodes :

- ▶ **Initialize the first manager node**
- ▶ **Join additional manager nodes (optional)**
- ▶ **Join worker nodes**

The following tasks can be performed :

- ▶ **Choose a node to initialize a new Swarm (Manager)**

```
$ docker swarm init \
--advertise-addr 172.17.8.104:2377 \
--listen-addr 172.17.8.104:2377
```

```
root@node1:/home/vagrant# docker swarm init --advertise-addr 172.17.8.104 --listen-addr 172.17.8.104
Swarm initialized: current node (m91tzq1frus72zijkl6dvz252) is now a manager.
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-336qe1yp4arw0eyi0t1616h6ujywh1gb50ea2c4b4lz50i5uuj-1fq9x4bayg3ivd6ezfsbqqudt 172.17.8.104:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

Build Swarm Cluster

"docker swarm init": Initialize a new swarm and make this node the first manager.

--advertise-addr : Swarm API endpoint that will be used to connect to the manager by other nodes in the Swarm. It's an optional flag and gives control over which IP gets used on nodes with multiple IPs.

--listen-addr : The node will accept Swarm traffic on this IP Address. Sometimes used to restrict Swarm to a particular IP on a system with multiple IPs.

The default port that swarm mode operates on is 2377 for secured(HTTPS) client-to-swarm connections

► **Join a worker to the Swarm** (Copy the output of the previous command)

\$ docker swarm join-token worker (*Run from Manager 1 node to extract the following commands and tokens required to add a new workers*)

\$ docker swarm join --token XXXXXXXXXXXXXXXX 172.17.8.104:2377 (*Run from Worker*)

```
root@node2:/home/vagrant# docker swarm join --token SWMTKN-1-336qe1yp4arw0eyi0t1616h6ujywh1gb50ea2
yg3ivd6ezfsbqqdt 172.17.8.104:2377
This node joined a swarm as a worker.
```

Build Swarm Cluster

► Join a Manager Node to the Swarm

```
$ docker swarm join-token manager //(Run from Manager 1 node to extract the following commands and tokens required to add a new workers)  
//Run from Manager 2  
$ docker swarm join \  
    --token XXXXXX...XXXXXXXXX\  
    172.17.8.4:2377 \  
    --advertise-addr 172.17.8.40:2377 \  
    --listen-addr 172.17.8.4:2377
```

172.17.8.40 IP Address of the new Manager Node.

► List the nodes in the Swarm

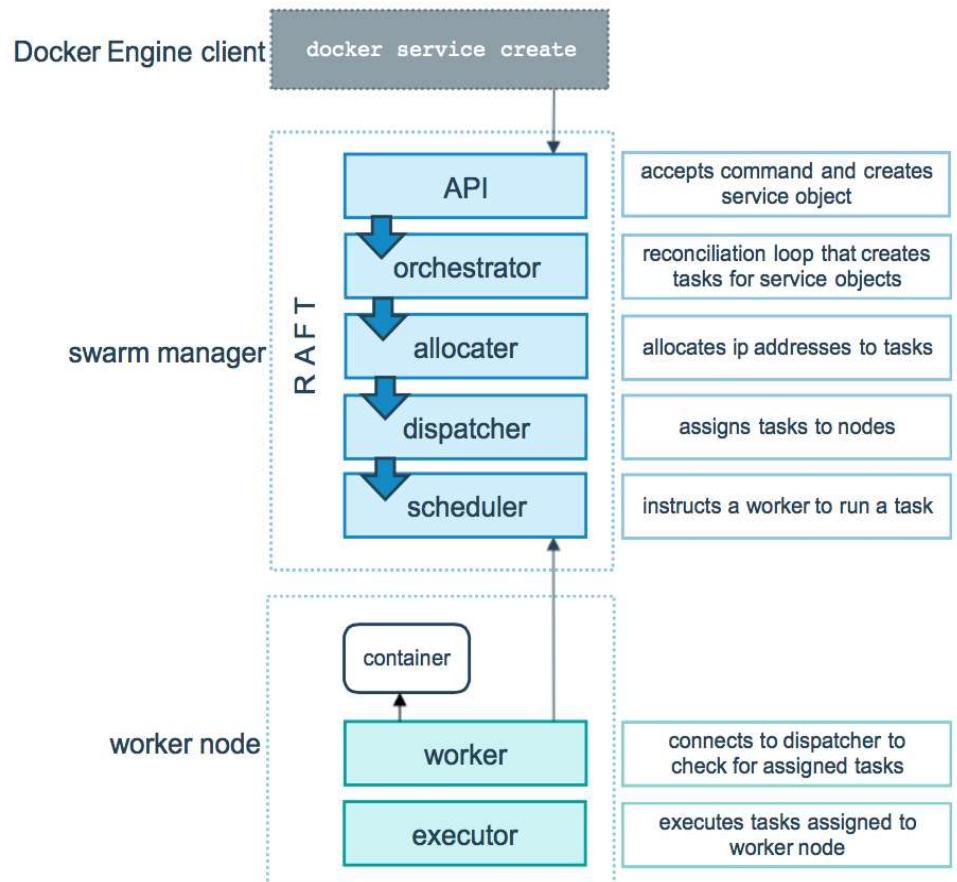
```
$ docker node ls
```

```
root@node1:/home/vagrant# docker node ls  
ID            HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS   ENGINE VERSION  
m91tzq1frus72zijkl6dvz252 *  node1    Ready   Active        Leader        20.10.8  
rh8n6xvpx4jbvpqbqzqgt7rpi  node2    Ready   Active          20.10.8
```

Swarm Services

To deploy new service :

- ▶ **Swarm manager** accepts your service definition as the desired state for the service.
- ▶ it schedules the service on nodes in the swarm as one or more **replica** tasks.
- ▶ The task run independently of each other on nodes in the Swarm



Swarm service create/update

► Create a service

```
$ docker service create --name hello-world \  
  --replicas 1 \  
    alpine ping docker.com
```

```
root@node1:/home/vagrant# docker service create --replicas 1 --name hello-world alpine ping docker.com  
v80hzro3dfc5wvu584e80c4ar  
overall progress: 1 out of 1 tasks  
1/1: running [=====>]  
verify: Service converged
```

```
$ docker service create --name redis \  
  -p 6369:6369 \  
  --replicas 2 \  
    redis
```

```
root@node1:/home/vagrant# docker service create --replicas 2 --name redis --publish 6369:6369 redis  
qya2r64kqpk0vcax1rdypw607  
overall progress: 2 out of 2 tasks  
1/2: running [=====>]  
2/2: running [=====>]  
verify: Service converged
```

Swarm service create/update

► Viewing and inspecting services

```
$ docker service ls
```

```
root@node1:/home/vagrant# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
v80hzro3dfc5  hello-world  replicated  1/1      alpine:latest
qya2r64kqpk0   redis       replicated  2/2      redis:latest
                                         *:6369->6369/tcp
```

```
$ docker service ps redis
```

```
root@node1:/home/vagrant# docker service ps redis
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE
n38vhzqpqrw  redis.1   redis:latest  node2    Running     Running 2 minutes ago
r9eeodsh4qgd  redis.2   redis:latest  node1    Running     Running about a minute ago
```

```
$ docker service inspect redis
```

```
root@node1:/home/vagrant# docker service inspect redis
[{"Service": {
  "ID": "znod5qkqvr5k4w0f8vinjx9ga",
  "Version": {
    "Index": 1892
  },
  "CreatedAt": "2021-09-28T12:32:51.303293741Z",
  "UpdatedAt": "2021-09-28T12:32:51.305878024Z",
  "Spec": {
    "Name": "redis",
    "Labels": {},
    "TaskTemplate": {
      "ContainerSpec": {
        "Image": "redis:latest@sha256:0918f1195cad14d21ffa",
        "Init": false,
        "StopGracePeriod": 10000000000,
        "DNSConfig": {},
        "Isolation": "default"
      },
      "Resources": {
        "Limits": {},
        "Reservations": {}
      },
      "RestartPolicy": {
        "Condition": "any",
        "Delay": 5000000000,
        "MaxAttempts": 0
      }
    }
  }
}}
```

Swarm service create/update

► Scaling services (help with overloaded traffic)

```
$ docker service scale redis=4
```

```
root@node1:/home/vagrant# docker service scale redis=4
redis scaled to 4
overall progress: 4 out of 4 tasks
1/4: running  [=====>]
2/4: running  [=====>]
3/4: running  [=====>]
4/4: running  [=====>]
verify: Service converged
```

```
$ docker service ls
```

```
root@node1:/home/vagrant# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
f3mlbyncfkem  hello-world  replicated  1/1      alpine:latest
znod5qkqvr5k   redis      replicated  4/4      redis:latest  *:6369->6369/tcp
```

```
$ docker service ps redis
```

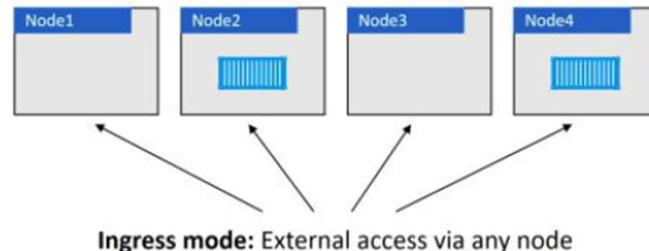
```
root@node1:/home/vagrant# docker service ps redis
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE
mzqu796m60rg  redis.1  redis:latest  node2    Running      Running 6 minutes ago
p51nw1t438k   redis.2  redis:latest  node1    Running      Running 6 minutes ago
gf8c6yontu8d  redis.3  redis:latest  node2    Running      Running 4 minutes ago
pitpdq4lhrot  redis.4  redis:latest  node1    Running      Running about a minute ago
```

Docker Network mode (ingress load balancing)

Swarm supports two(02)publishing modes that make services accessible from outside of the Cluster

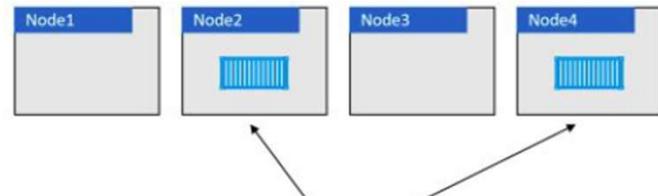
- ▶ **Ingress mode (default)** : Services published (with `--publish`) can be accessed from any node in the Swarm

```
# docker service create -d --name example \
  --publish published=5000, target=80 \
  nginx
```



- ▶ **Host mode** : Services published (with `--publish` and add `mode=host`) can only be accessed via nodes running service replicas

```
# docker service create -d --name example2 \
  --publish published=5001, target=8080, mode=host \
  nginx
```



Swarm Rolling Updates

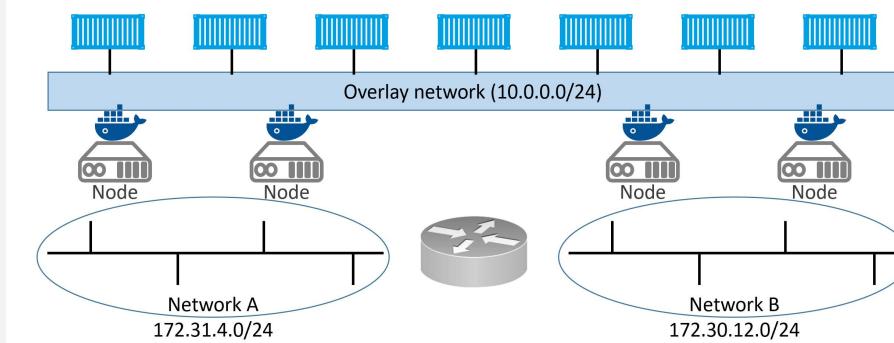
Docker services update is very easy and simplify

► **Create a new Overlay network**

```
$ docker network create -d overlay devops
```

```
root@node1:/home/vagrant# docker network create -d overlay devops
tkhh4fvf5o1vywp9mwwx4vjm7
```

The overlay network creates a new layer 2 container network on top of potentially multiple different underlying networks



Swarm Rolling Updates

► Verify network

```
$ docker network ls

root@node1:/home/vagrant# docker network create -d overlay devops
tkhh4fvf5o1vywp9mwwx4vjm7
root@node1:/home/vagrant# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
8d3a7895a564   bridge    bridge    local
tkhh4fvf5o1v   devops    overlay   swarm
4a8eb79b81bd   docker_gwbridge bridge    local
17b2b772f5e7   host      host     local
bxlxoak7r4nl   ingress   overlay   swarm
c364deaf96e8   none     null     local
```

► Deploy a new service and attach it to the network

```
$ docker service create --name=test-devops --network devops -p 80:80 --replicas 12 gilbertfongan/demo:v1
```

```
root@node1:/home/vagrant# docker service create --name=test-devops --network devops -p 80:80 --replicas
12 gilbertfongan/demo:v1
jhd9v7mb4hkv8mo2w2gnchcnt
overall progress: 12 out of 12 tasks
1/12: running
2/12: running
3/12: running
4/12: running
5/12: running
6/12: running
7/12: running
8/12: running
9/12: running
10/12: running
11/12: running
12/12: running
verify: Service converged
```

Swarm Rolling Updates

Passing the service, the ***-p 80:80 flag*** ensure that a Swarm-wide mapping is created that maps all traffic, coming into any node in the Swarm on **port 80**, through to **port 80** inside of any **service replica**.

Ingress mode : Define the mode of publishing a port on every node in the Swarm

► Deploy the update

```
$ docker service ls
```

```
root@node1:/home/vagrant# docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
f3milbyncfkem  hello-world  replicated  1/1      alpine:latest
znod5qkqvr5k   redis      replicated  1/1      redis:latest
jhd9v7mb4hkv   test-devops  replicated  12/12    gilbertfongan/demo:v1  *:80->80/tcp
```

```
$ docker service update --image gilbertfongan/demo:v2 --update-parallelism 2 --update-delay 20s test-devops
```

```
root@node1:/home/vagrant# docker service update --image gilbertfongan/demo:v2 --update-parallelism 2 --update-delay 20s test-devops
test-devops
overall progress: 12 out of 12 tasks
1/12: running
2/12: running
3/12: running
4/12: running
5/12: running
6/12: running
7/12: running
8/12: running
9/12: running
10/12: running
11/12: running
12/12: running
verify: Service converged
```

Docker Swarm Commands

COMMANDS	DESCRIPTION
Docker swarm init	Initialize/Create a new swarm
Docker swarm join-token	Reveals the command and tokens needed to join workers and managers to existing Swarms. <i>Docker swarm join-token manager</i> command is used to expose the command to join a new manager <i>Docker swarm join-token worker</i> command is used to expose the command to join a new worker
Docker node ls	List all nodes in the Swarm including which are managers and leader
Docker service create	Command to create a new service
Docker service ls	List running services in the Swarm and gives basic info on the state of the service and any replicas it's running
Docker service ps <service>	Give more detailed information about individual service replicas
Docker service inspect	Give very detailed information on a service
Docker service scale	Scale the number of replicas in a service up and down
Docker service update	Update many of the properties of a running service
Docker service logs	View the logs of a service
Docker service rm	Delete a service from the Swarm

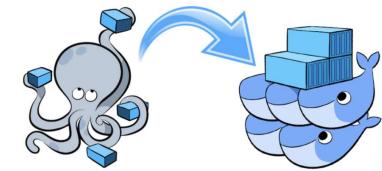
Docker Stacks

- ▶ Help to define complex multi-service apps in a **single declarative file**.
- ▶ While Docker is a great tool for development and testing, **Docker stacks** are great tools for **scale** and **production**.
- ▶ Provide a simple way to deploy the App and Manage its entire lifecycle :

- Health checks
- Scaling
- Updates and Rollbacks

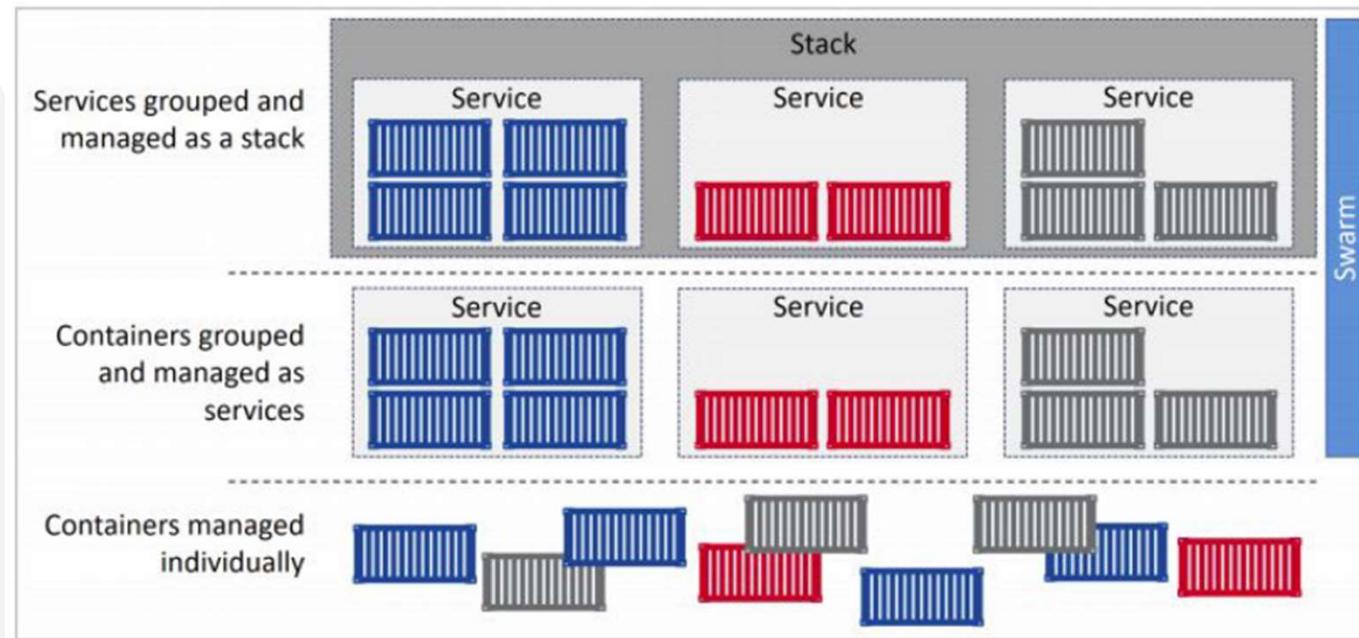
▶ The stack file includes the entire stack of services that make up the App in the form of Compose file :

- **Services**
- **Volumes**
- **Networks**
- **Secrets**



Docker Stacks

- ▶ **Stacks** are often compared to **Compose** with the only difference being that it deploys on a **cluster swarm**
- ▶ They are at the top of the Docker application hierarchy.
- ▶ They build on top of services, which turn build on top of containers



Stacks deploy example

Create a NGINX and MySQL container

► Define services "web-app.yml"

```
Version : '3'  
Services :  
  Web:  
    image:nginx  
    Ports:  
    - "8081:80"  
  mysql:  
    Image: mysql  
    Environment:  
      MYSQL_ALLOW_EMPTY_PASSWORD : "yes"
```

Stacks deploy example

► Deploy the stack

```
$ docker stack deploy -c web-app.yml webapp  
Creating network webapp_default  
Creating service webapp_web  
Creating service webapp_mysql
```

A default network is created

► List Docker stacks

```
$ docker stack ls  
NAME      SERVICES      ORCHESTRATOR  
webapp    2            Swarm  
  
$ docker stack services webapp  
ID        NAME          MODE        REPLICAS      IMAGE          PORTS  
j0uprwm4oua9  webapp_mysql  replicated  1/1          mysql:latest  
qg4nqt7sdmo3  webapp_web   replicated  1/1          nginx:latest  *:808  
1->80/tcp
```

Stacks deploy example

► Task making up the services

```
$ docker stack ps webapp
```

ID	NAME	IMAGE	NODE	DESIRED
STATE	CURRENT STATE	ERROR	PORTS	
ow1buo3i4mfj	webapp_mysql.1	mysql:latest	node1	Running 12 minutes ago
jnv24yv13ekz	webapp_web.1	nginx:latest	node2	Running 12 minutes ago

► Delete Docker stacks

```
$ docker stack rm webapp
```

```
Removing service webapp_mysql
Removing service webapp_web
Removing network webapp_default
```

Docker Stacks Commands

COMMANDS	DESCRIPTION
docker stack deploy	Command used to deploy and update stacks of services defined in a stack file which is usually docker-stack.yml
Docker stack ls	List all stacks on the Swarm, including how many services they have
Docker stack ps	Gives detailed information about a deployed stack. List which node each replica is running on, and shows desired state and current state
docker stack rm	Delete a stack from the Swarm.

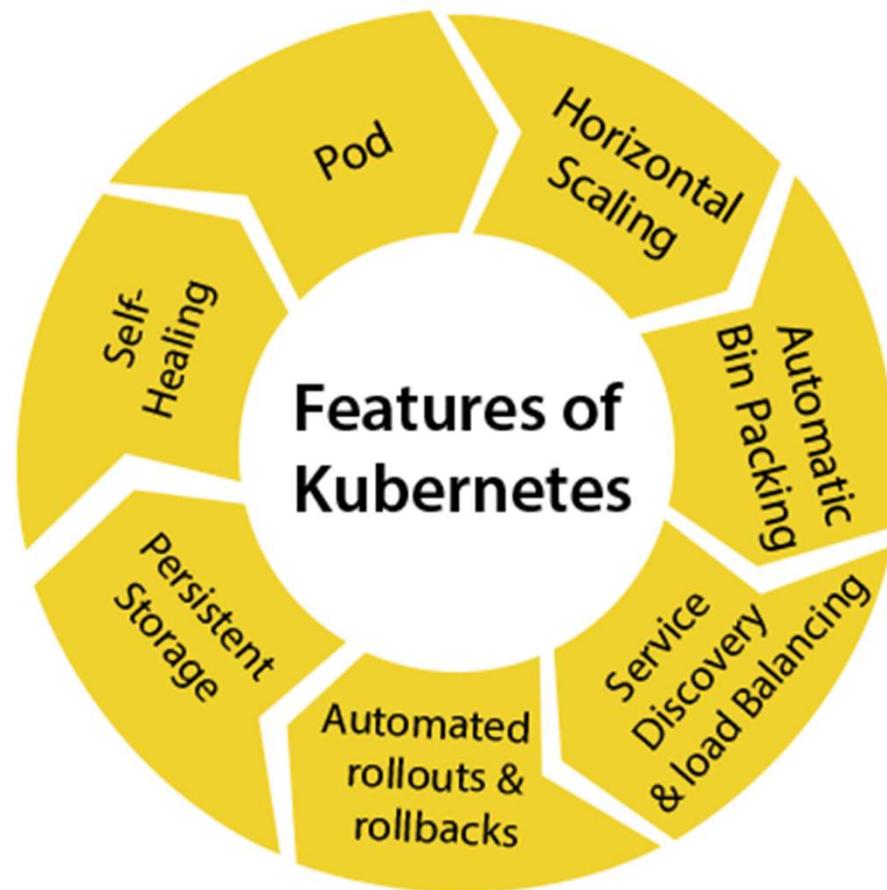
Kubernetes (K8s)

- ▶ **Service for Cluster Management**
- ▶ Open sourced by **Google**
- ▶ Applications **orchestrator**
- ▶ Comparable to Docker Swarm
- ▶ Automates the deployment, scaling and management of containerized microservice applications
- ▶ Support Docker and Rkt runtimes
- ▶ Portable and flexible

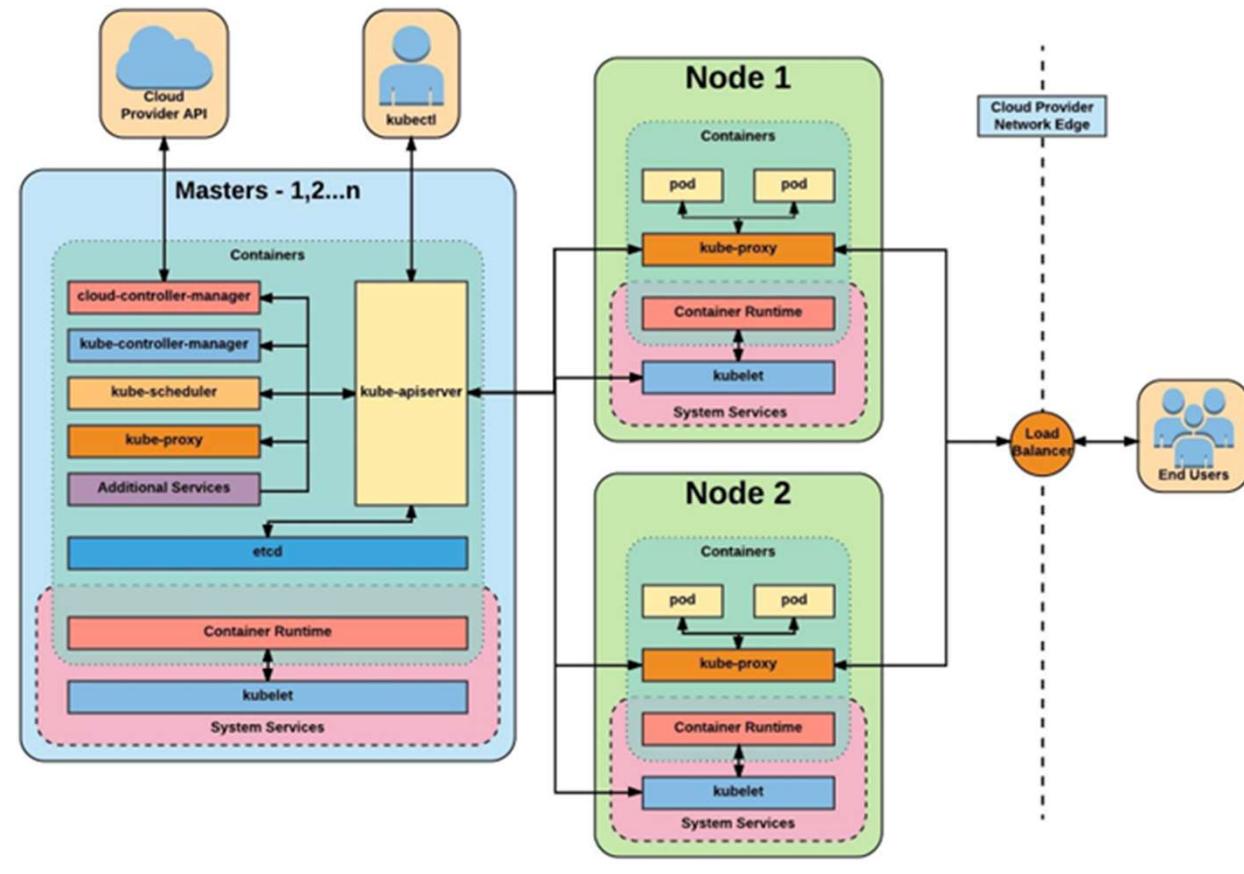


kubernetes

Kubernetes



Kubernetes Architecture



Kubernetes Architecture

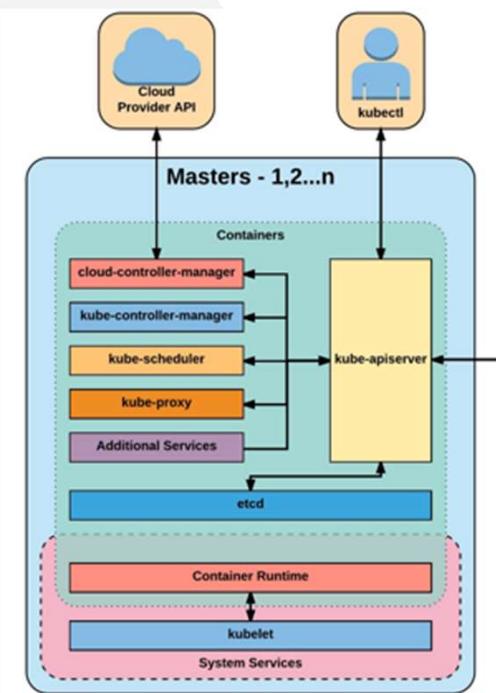
Master :

- ▶ Responsible for cluster management
- ▶ Entry point for administrative commands
- ▶ Several masters behind a load balancer for HA architecture

Worker (node) :

- ▶ Launch Pods of an application
- ▶ Communicates with the Master
- ▶ Provides resources to Pods

Kubernetes Architecture / Master



► Kube-apiserver

- Entry point exposing the K8s HTTP REST API

► Etcd

- Provide consistent and highly available key-value store used for persisting cluster state
- Configuration and service discovery

► Kube-scheduler

- Select the node on which a pod will be launched
- Considers the resources needed and those available

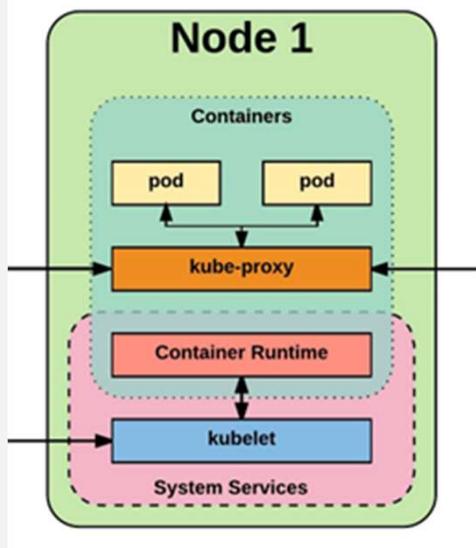
► Kube-controller-manager

- Process for managing the different controllers (node, replication, endpoint)
- Corrective actions if necessary

► Cloud-controller-manager

- Provides cloud-provider specific integration capability into the core control loop
- Add additional controller to handle Persistent Volume Labels

Kubernetes Architecture / Worker (node)



► Kubelet

- Process running on each machine in the cluster
- Ensure that a pod's containers rotate according to specification
- Communicates with the Master

► Kube-proxy

- Allows services to be exposed to the outside
- Manages network rules

► Container runtime engine

- Containerd(docker)
- CRI-O
- Rkt

Kubernetes Additional Services

- ▶ **Kube-dns**

Provides cluster wide DNS Services. Services are resolvable to :

<service>.<namespace>.svc.cluster.local

- ▶ **Heapster**

Metrics collector for Kubernetes cluster, used by some resources such as the Horizontal Pod Autotscaler

- ▶ **Kube-dashboard**

General purpose web-based UI for Kubernetes

Kubernetes Concept

- ▶ **Cluster** : A collection of hosts that aggregate their available resources including CPU, RAM, Disk, and their devices into a usable pool.
- ▶ **Master** : A collection of components that make up the control plane of Kubernetes and are responsible for all cluster decisions including both scheduling and responding to cluster events.
- ▶ **Node/Worker** : A single host, physical or virtual capable of running pods. He is managed by the Master(s), and at a minimum runs both Kubelet and Kube-proxy to be considered part of the Cluster.
- ▶ **Namespace** : A logical cluster or environment. Primary method of dividing a cluster or scoping access
- ▶ **Label** : Key-value pairs that are used to identify, describe and group together related sets of objects. Labels have a strict syntax and available character set.
- ▶ **Selector** : Use labels to filter or select objects.

Kubernetes Object Categories

- ▶ Management of applications launched on the Cluster (Deployment, Pod)
- ▶ Discovery and Load Balancing (Service)
- ▶ Configuration of applications (ConfigMap, Secret)
- ▶ Storage (PersistentVolume, PersistentVolumeClaim)
- ▶ Cluster configuration and metadata (Namespace)

Kubernetes Init and Config

Initialize the cluster with a Pod Network

```
$ kubeadm init --apiserver-advertise-address=MASTER_IP --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all  
root@node01:/home/vagrant# kubeadm init --apiserver-advertise-address=10.10.1.120  
--pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all  
[init] Using Kubernetes version: v1.22.3  
[preflight] Running pre-flight checks  
[preflight] Pulling images required for setting up a Kubernetes cluster  
[preflight] This might take a minute or two, depending on the speed of your intern  
et connection  
[preflight] You can also perform this action in beforehand using 'kubeadm config i  
mages pull'  
[certs] Using certificateDir folder "/etc/kubernetes/pki"  
[certs] Generating "ca" certificate and key  
[certs] Generating "apiserver" certificate and key  
[certs] apiserver serving cert is signed for DNS names [kubernetes kubernetes.defa  
ult kubernetes.default.svc kubernetes.default.svc.cluster.local node01] and IPs [1  
0.96.0.1 10.10.1.120]  
[certs] Generating "apiserver-kubelet-client" certificate and key  
[certs] Generating "front-proxy-ca" certificate and key  
[certs] Generating "front-proxy-client" certificate and key  
[certs] Generating "etcd/ca" certificate and key  
[certs] Generating "etcd/server" certificate and key  
[certs] etcd/server serving cert is signed for DNS names [localhost node01] and IP  
s [10.10.1.120 127.0.0.1 ::1]  
[certs] Generating "etcd/peer" certificate and key  
[certs] etcd/peer serving cert is signed for DNS names [localhost node01] and IPs  
[10.10.1.120 127.0.0.1 ::1]  
[certs] Generating "etcd/healthcheck-client" certificate and key  
[certs] Generating "apiserver-etcd-client" certificate and key  
[certs] Generating "sa" key and public key
```

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 10.10.1.120:6443 --token achk05.u8mglsx3nt027tcs \  
--discovery-token-ca-cert-hash sha256:ebc61f8b4c310372d3ef5539a67d52ad7685  
9468eb63cac4dbd50f5d6fb110b0
```

Kubernetes Init and Config

Join the cluster with a Worker

```
$ kubeadm join 10.10.1.120:6443 --token achk05.u8mglsx3nt027tcs --discovery-token-ca-cert-hash sha256:ebc61f8b4c310372d3ef5539a67d52ad76859468eb63cac4dbd50f5d6fb110b0

root@node02:/home/vagrant# kubeadm join 10.10.1.120:6443 --token achk05.u8mglsx3nt
027tcs \
>     --discovery-token-ca-cert-hash sha256:ebc61f8b4c310372d3ef5539a67d52ad76
859468eb63cac4dbd50f5d6fb110b0
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get
cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kube
let/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Check all nodes from Master node

```
$ kubectl get nodes

root@node01:/home/vagrant# kubectl get nodes
NAME      STATUS    ROLES          AGE      VERSION
node01    Ready     control-plane,master   87m      v1.22.3
node02    Ready     <none>           3m1s    v1.22.3
```

Kubernetes Init and Config

► Show Merged Kubeconfig settings

```
$ kubectl config view
```

► Display list of contexts

```
$ kubectl config get-contexts
```

► Set the default context to MY_CLUSTER_NAME

```
$ kubectl config use-context MY_CLUSTER_NAME
```

► Get a list of users

```
$ kubectl config view -o jsonpath='{.users[*].name}'
```

► Display addresses of the Master and services

```
$ kubectl cluster-info
```

Kubernetes Objects Structure

- ▶ **ApiVersion** : Identifies the version of the schema the object should have
- ▶ **Kind** : Identifies the schema the object should have
- ▶ **Metadata** : Adding name, labels, annotations, timestamp, namespace
- ▶ **Spec** : Component specification / description

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: vote
  name: vote
  namespace: vote
spec:
  containers:
    - name: www
      image: nginx:1.14.2
```

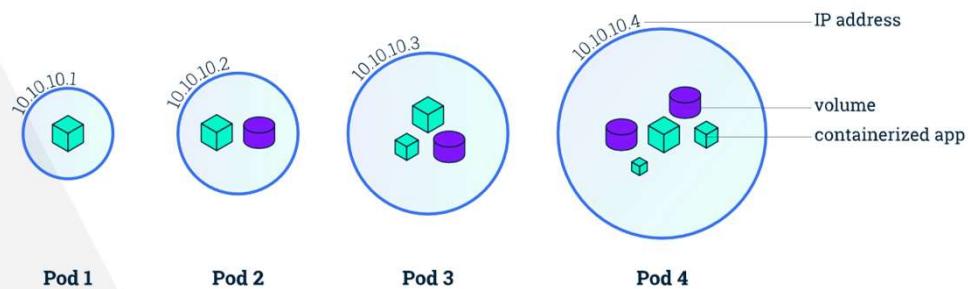
```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: vote
  name: vote
  namespace: vote
spec:
  type: NodePort
  ports:
    - name: "vote-service"
      port: 5000
      targetPort: 80
      nodePort: 31000
  selector:
    app: vote
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: www-domain
  namespace: vote
spec:
  rules: NodePort
  - host: www.example.com
    http:
      paths:
        - backend:
            serviceName: www
            servicePort: 80
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80
```

Kubernetes Pods

- ▶ **Smallest unit of deployment in Kubernetes**
- ▶ **A group of one or more application containers**
 - Shared storage, as Volumes
 - Share Network, as a unique cluster IP address
- ▶ **Application split into several specifications of Pods**
- ▶ **Configuration information that determine how a container should run**



Kubernetes Pods Example

Here is an example of configuration for the "Hello World !

- ▶ **Containers** : describes the containers that make up our Pod, it is a Yaml list (there can be several containers) ;
- **Container.name** : the name of the container, which may or may not be the name of the induced Pod;
- **Container.image** : The Docker image called to generate the container (then Pod)
- **Container.imagePullPolicy**: "Always" to get the latest version of the Docker image tag
- **Container.ports**: The listening ports of the container
- **Container.stdin & container.tty** : To invoke a TTY/Stdin to be able to execute commands and enter the container with a Shell
- **Container.livenessProbe** : Probe that ensures the proper functioning of the container, and more precisely of the service embedded by the container.

```

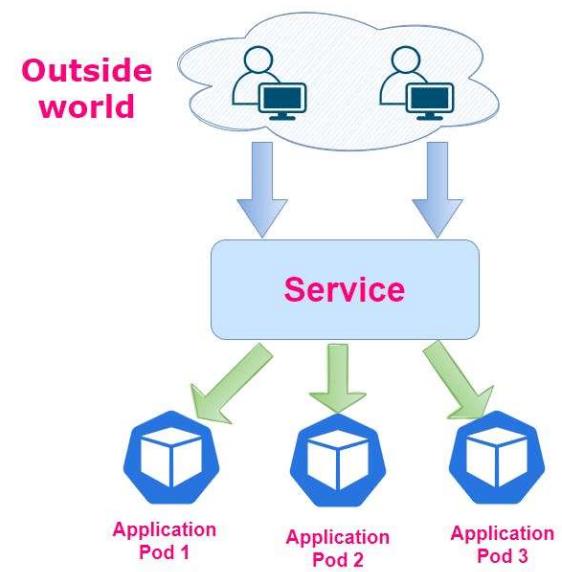
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
spec:
  containers:
    - name: hello-world
      image: tutum/hello-world
      imagePullPolicy: Always
      ports:
        - containerPort: 80
      stdin: true
      tty: true
      livenessProbe:
        httpGet:
          path: /
          port: 80
      initialDelaySeconds: 10
      periodSeconds: 10
  
```

Kubernetes Pods Commands

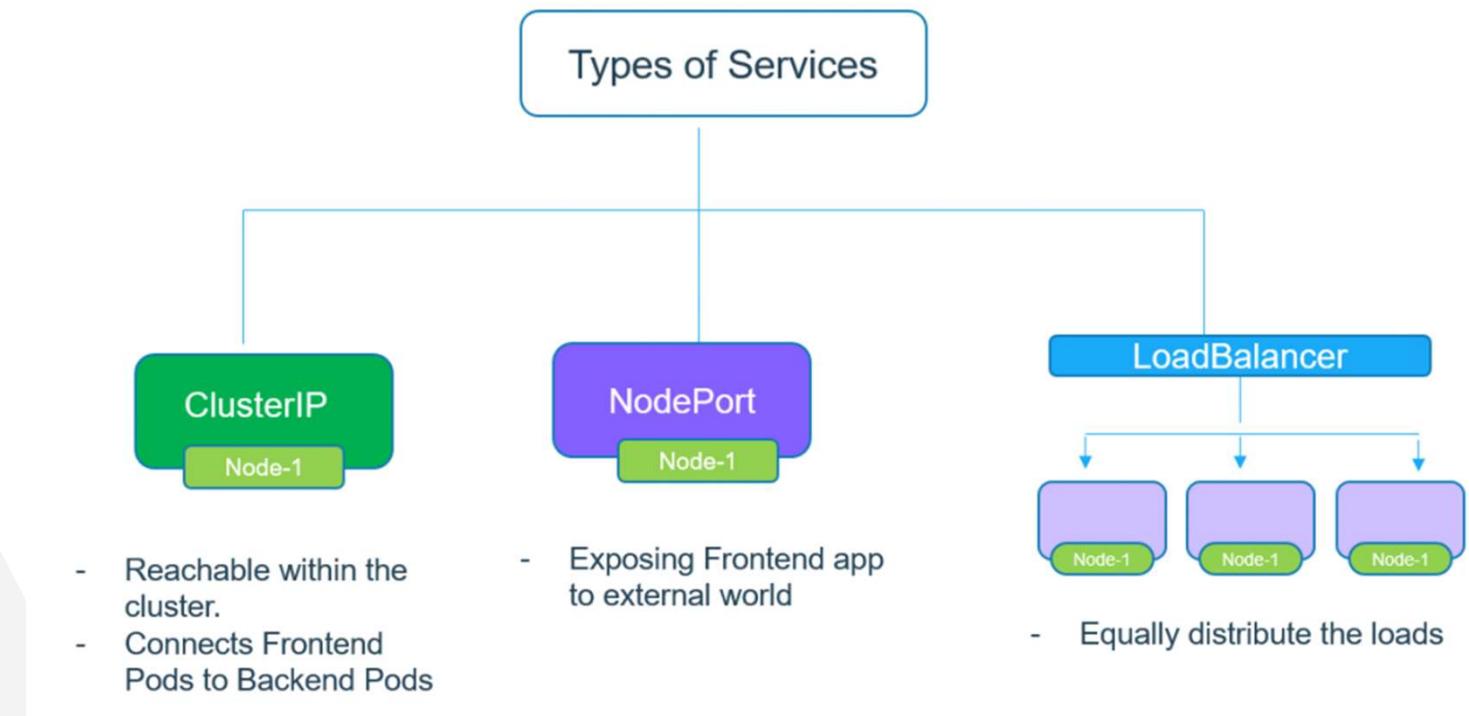
COMMANDS	DESCRIPTION
<code>Kubectl create -f POD.yaml</code>	Command used to run Pod based on the YAML file "POD.yaml"
<code>Kubectl get pods</code>	List all pods deployed
<code>Kubectl describe pod POD_NAME</code>	Describe a given pod
<code>Kubectl logs POD_NAME [-c CONTAINER_NAME]</code>	Get logs from a Pod or/and specifically a container running inside the Pod
<code>Kubectl exec POD_NAME [-c CONTAINER_NAME] -- COMMAND</code>	Execute command in an existing Pod or/and specially a container running inside the Pod
<code>Kubectl delete pod POD_NAME</code>	Remove or delete a Pod
<code>Kubectl port-forward POD_NAME HOST_PORT:CONTAINER_PORT</code>	Forwarding Port in a Pod (Allows to publish the port of a Pod on the host machine)

Kubernetes Services

- ▶ Abstraction way to expose an Application running on a set of Pods
- ▶ Ensures decoupling
- Replicas/microservice instances
- Microservice consumers
- ▶ Load balancing between the underlying Pods
- ▶ Persistent IP Address



Kubernetes Services types



Kubernetes Services types

These services can be used in different ways based on the types.

- ▶ **ClusterIP (default)** : Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the Cluster
- ▶ **NodePort** : Exposes the Service on the same port of each selected Node in the Cluster using NAT (Network Address Translation). Makes the Service accessible from outside the Cluster using <NodeIP>:<NodePort>.
- ▶ **LoadBalancer** : Creates an external load balancer in the current cloud provider (AWS, Azure, GCP) and assigns a fixed, external IP to the Service.

Kubernetes Services Example

Here is an example of configuration for the "vote" service

- ▶ **ApiVersion** : It specifies api version for the service
- ▶ **Kind** : It specifies what you are going to achieve with the file (deployment, pod, service, secret, jobs, ingress)
- ▶ **Metadata** : Information about the kind specified.
- ▶ **Spec** : Specification related to this kind
- ▶ **Selector** : Be careful to specify the correct label name here what we have used while creating the deployment. Else, it won't work properly.
- ▶ **Type** : Type of service (ClusterIP, NodePort, LoadBalancer,...)
- ▶ **Ports** : Specify the name of the service port, port number and target port

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: vote
  name: vote
  namespace: vote
spec:
  selector:
    app: vote
  type: NodePort
  ports:
  - name: "vote-service"
    port: 5000
    targetPort: 80
    nodePort: 31000
```

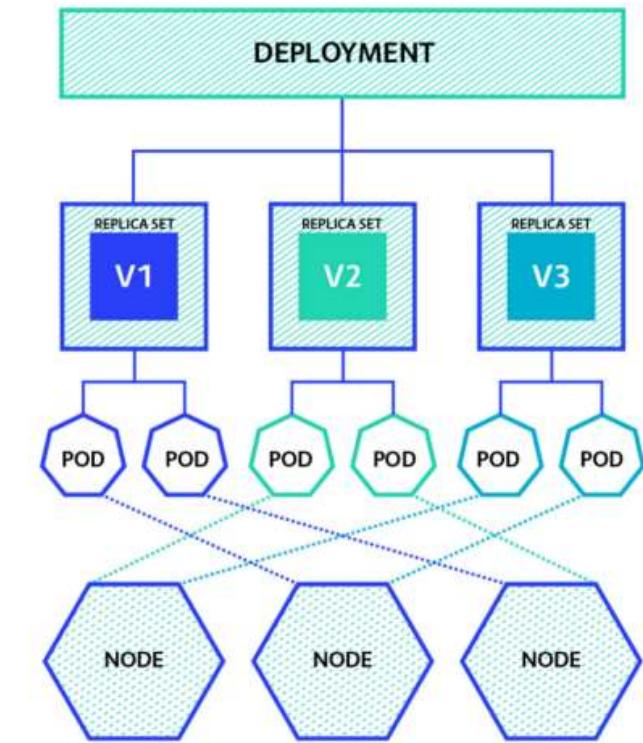
Kubernetes Service Commands

COMMANDS	DESCRIPTION
<code>Kubectl create -f SERVICES.yaml</code>	Command used to create services based on the YAML file "SERVICES.yaml"
<code>Kubectl get services</code>	List all services deployed in the namespace. With [--all-namespaces] to list in all namespaces
<code>Kubectl describe svc SERVICE_NAME</code>	Description of a Service
<code>Kubectl delete svc SERVICE_NAME</code>	Remove or delete a Service
<code>Kubectl port-forward svc/SERVICE_NAME 5000</code>	Listen on Local port 5000 and forward to port 5000 on service backend
<code>Kubectl port-forward svc/SERVICE_NAME 5000:TARGET_PORT</code>	Listen on Local port 5000 and forward to service target port

Kubernetes Deployment

A **Kubernetes deployment** is a resource object that provides declarative updates to applications and allows you to explain life cycle.

- ▶ **Different levels of abstraction**
 - Deployment
 - ReplicaSet
 - Pod
- ▶ **Pod generally created via a Deployment**
- ▶ **A Deployment manages ReplicaSets**
- ▶ **ReplicaSet**
 - A version of the application
 - Manages a set of Pods of the same specification
 - Ensures that the Pods are active



Kubernetes Deployment

Deployment defines a "desired state"

- Specification of a Pod and the desired number of replicas
- ▶ **A controller to converge the "current state"**
- ▶ **Manages updates of an application**
- Rollout/Rollback/Scaling
- Creation of a new ReplicaSet when updating the application
- ▶ **Different update strategies**
- Rolling update, blue/green, canary

Kubernetes Deployment Example

The following is an example of a Deployment. It creates a ReplicaSet to bring up three nginx Pods:

- ▶ **ApiVersion** : It specifies api version(v1) for the service
- ▶ **Kind** : It specifies what you are going to achieve with the file (deployment, pod, service, secret, jobs, ingress)
- ▶ **Metadata.labels** : Pods are labeled app:nginx
- ▶ **Spec.replicas**: Number of replicas created by the deployment (03)
- ▶ **Spec.selector** : Defines how the Deployment finds which Pods to manage
- ▶ **Spec.selector.matchLabels** : Map of key-value pairs
- ▶ **Spec.containers** : Create one container, name it and specify image and ports

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80
```

Kubernetes Deployment Commands

COMMANDS	DESCRIPTION
<code>Kubectl create deployment nginx --image=nginx Kubectl apply -f DEPLOYMENT.yaml</code>	Command used to create deployment based on the YAML file "DEPLOY.yaml"
<code>Kubectl get deployments</code>	List all deployments in the namespace. With [--all-namespaces] to list in all namespaces
<code>Kubectl get deployment DEPLOYMENT</code>	List a particular deployment
<code>Kubectl describe deployment DEPLOYMENT</code>	Description of a Deployment
<code>Kubectl delete deploy DEPLOYMENT</code>	Remove or delete a Deployment. Add [-l name=myLabel] to delete deployment with Label name=myLabel
<code>Kubectl set image deployment/DEPLOYMENT www=image:v2</code>	Rolling update "www" containers of "DEPLOYMENT", updating the image
<code>Kubectl rollout history deployment/DEPLOYMENT</code>	Check the history of deployments including the revision
<code>Kubectl rollout undo deployment/DEPLOYMENT</code>	Rollback to the previous deployment revision
<code>Kubectl rollout undo deployment/DEPLOYMENT --to-revision=2</code>	Rollback to a specific revision
<code>Kubectl rollout status -w deployment/DEPLOYMENT</code>	Watch rolling update status of "DEPLOYMENT" deployment until completion
<code>Kubectl rollout restart deployment/DEPLOYMENT</code>	Rolling restart of the "DEPLOYMENT" deployment
<code>Kubectl autoscale deployment DEPLOYMENT --min=2 --max=10</code>	Auto scale a deployment "DEPLOYMENT"

MODULE III-3 : Container Infrastructure

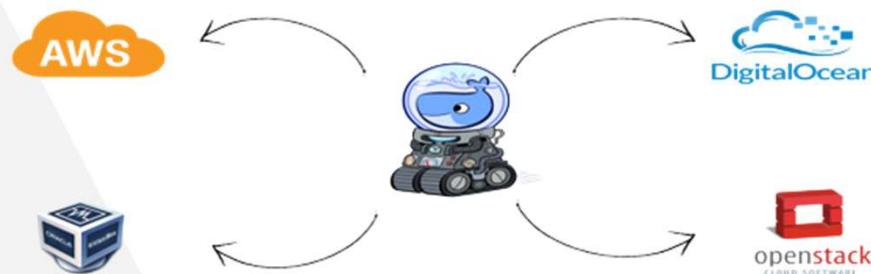
PLAN

- ▶ DOCKER MACHINE
- ▶ DOCKER DESKTOP



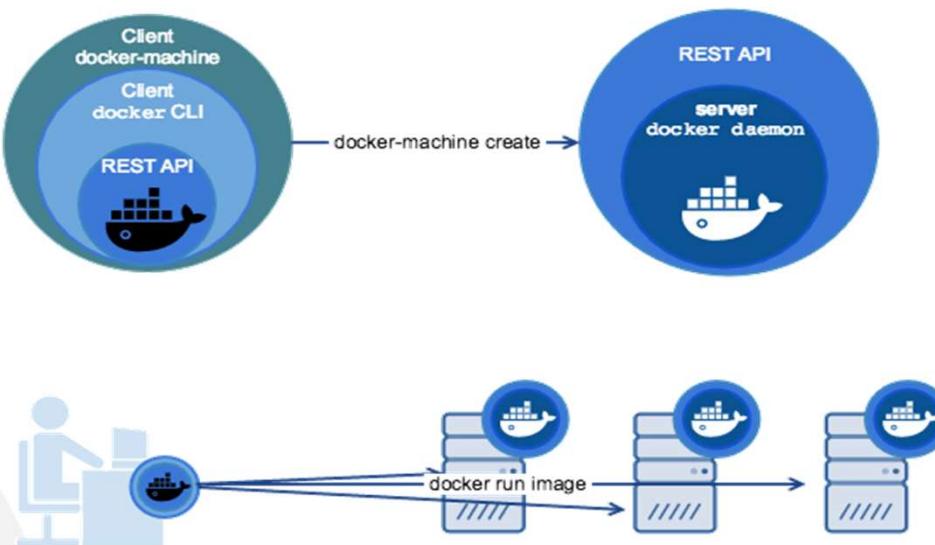
Docker Machine

- ▶ **Docker Machine** lets you create Docker hosts on your computer (**VirtualBox, VMWare**), on cloud providers (**AWS, Azure**), and inside your **own data center**.
- ▶ It creates servers, installs Docker
- ▶ It allows us to control the Docker engine of a VM created using docker-machine remotely
- ▶ Docker Machine is another command-line utility used for managing one or more local or remote machines
- ▶ Local machines are often run in separate VirtualBox instances.

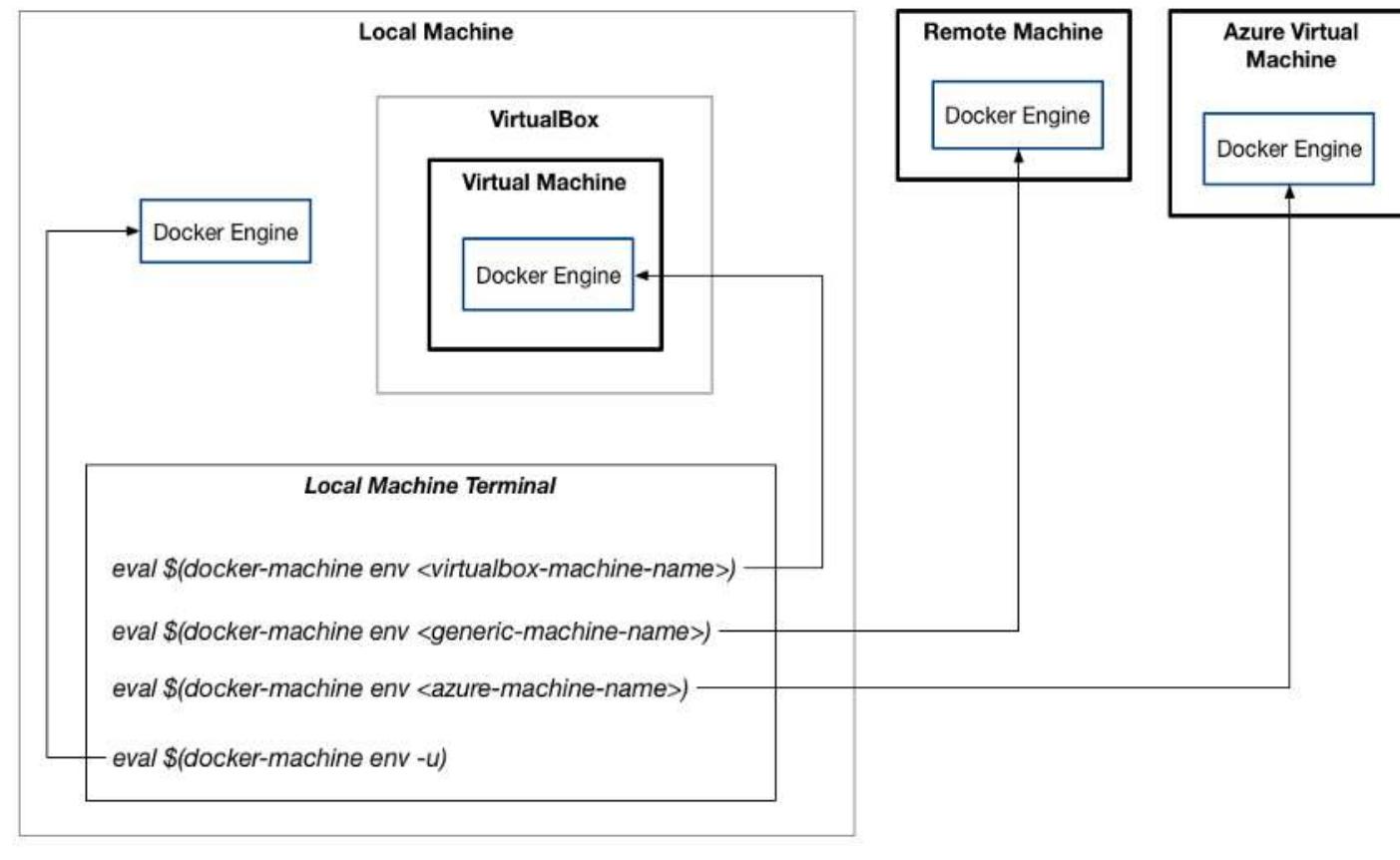


Docker Machine Driver

- ▶ The **drivers** concept act as a connector to **3rd party services** such as **Azure, AWS, etc.**
- ▶ Allows to create a complete set of resources around the VM to easily manage it from each service's admin portal
- ▶ Generic driver allows you to convert an actual(existing) VM into a Docker-machine



Docker Machine Driver



Docker Machine Creation

► **Install Docker-machine (old method-depreciated)**

```
$ curl -L https://github.com/docker/machine/releases/download/v0.14.0/docker-machine-uname -s-uname -m >/tmp/docker-machine  
&& chmod +x /tmp/docker-machine && sudo cp /tmp/docker-machine /usr/local/bin/docker-machine
```

► **Verifying version**

```
$ docker-machine version
```

► **Create a Docker Machine on VirtualBox**

```
$ docker-machine create -d virtualbox new-machine
```

► **Create a Docker machine with "generic" driver (remote existing VM)**

```
$ docker-machine create --driver generic \  
  --generic-ip-address=${MACHINE_IP} \  
  --generic-ssh-key ${SSH_PUBLIC_KEY}\  
  --generic-ssh-user ${SSH_USER} \  
  new-machine
```

Docker Machine Creation

► Create Docker Machine on Azure

```
docker-machine create --driver azure \  
    --azure-availability-set="MACHINE_NAME-as" \  
    --azure-subscription-id="${SUBSCRIPTION}" \  
    --azure-location "${AZURE_LOCATION}" \  
    --azure-open-port 80 \  
    --azure-open-port 443 \  
    --azure-size "${AZURE_MACHINE_SIZE}" \  
    --azure-subnet "${AZURE_VNET_NAME}-subnet" \  
    --azure-vnet "${AZURE_VNET_NAME}" \  
    --azure-resource-group "${RESOURCE_GROUP}" \  
    new-machine
```

► List the machine you have created

```
$ docker-machine ls
```

► Start a Docker Machine

```
$ docker-machine start new-machine
```

Docker Machine Commands

► Deploy containers to a remote host (**new-machine**)

- Change the local docker environment variables to the **new-machine** ones

```
$ eval $(docker-machine env demo-machine)
```

- Validate the active Docker Machine you point to

```
$ docker-machine active
```

- Run a container : Docker commands are not run locally but on Docker-machine

```
$ docker run -p 80:80 hello-world
```

- Curl the container deployed on Docker-machine

```
$ curl $(docker-machine ip new-machine):80
```

► SSH to Docker Machine

```
$ docker-machine ssh new-machine
```

► Copy files to/from the machine

```
$ docker-machine scp ~/loalfile.txt new-machine:~/
```

Docker Machine Commands

COMMANDS	DESCRIPTION
Docker-machine config	Print the connection config for machine
Docker-machine env	Display the commands to set up the environment for the Docker client
Docker-machine inspect	Inspect information about a machine
Docker-machine ip	Get the IP address of the machine
Docker-machine kill	Kill a machine
Docker-machine ls	List machines
Docker-machine provision	Re-provision existing machines
Docker-machine regenerate-certs	Regenerate TLS Certificates for a machine
Docker-machine restart	Restart a machine
Docker-machine start	Start a machine
Docker-machine status	Get the status of a machine
Docker-machine stop	Stop a machine
Docker-machine upgrade	Upgrade a machine to the latest version of Docker

Docker Desktop

Docker Desktop is an easy-to-install application for your Mac, Linux, or Windows environment that enables you to build and share containerized applications and microservices.

- ▶ Docker Desktop replaces Docker Machine !!!
- ▶ It provides a simple interface that enables you to manage your containers, applications, and images directly from your machine without having to use the CLI to perform core actions
- ▶ <https://docs.docker.com/desktop/>



PART I.V.

Configuration Management

...

MODULE IV-1 : Ansible

PLAN

- ▶ ANSIBLE ARCHITECTURE
- ▶ ANSIBLE INSTALLATION
- ▶ ANSIBLE CONFIGURATION
- ▶ ANSIBLE INVENTORY
- ▶ ANSIBLE COMMANDS
- ▶ ANSIBLE PLAYBOOKS
 - Structure
 - Example
 - Roles and variables
- ▶ ANSIBLE GALAXY
- ▶ ANSIBLE TOWER

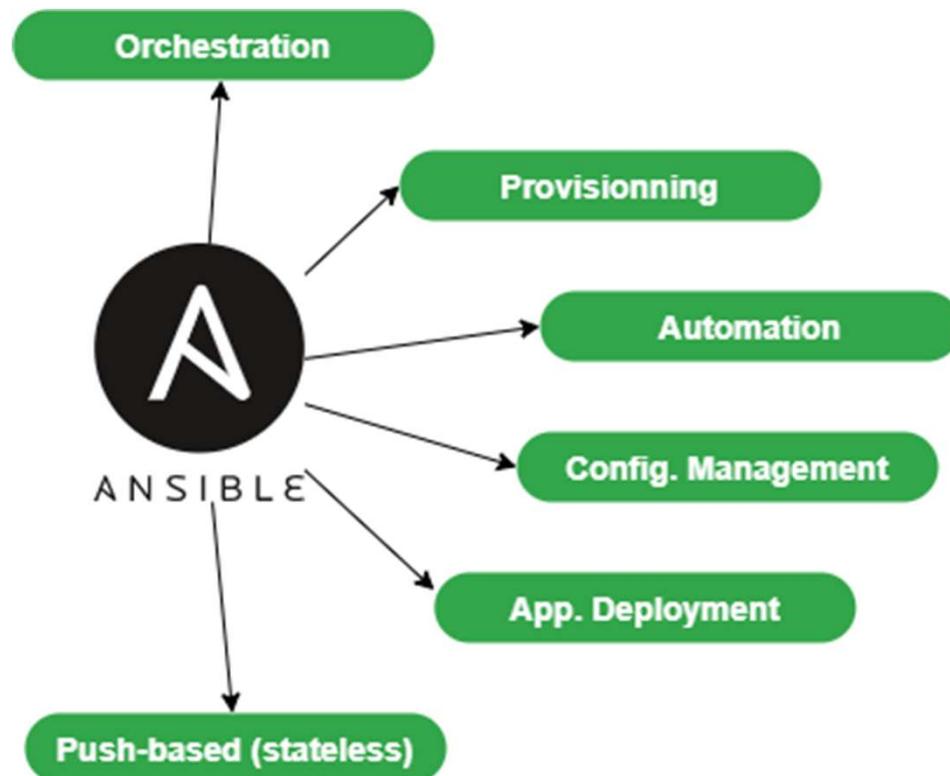


Ansible

- ▶ Ansible® is an open source, command-line IT automation software application written in **Python supported by Red Hat**
- ▶ **Immutable infrastructure** approach
- ▶ It allows you to **configure systems, deploy software** and **orchestrate** more advanced computing tasks
- ▶ **Continuous deployments** or permanent updates **without downtime**
- ▶ It also enables provisioning of virtual machines, containers and network, as well as complete cloud computing infrastructures.
- ▶ It also has a strong focus on **security** and **reliability**, featuring minimal moving parts
- ▶ It uses **OpenSSH** for transport (with other transports and pull modes as alternatives)
- ▶ Uses a **human-readable language** that is designed for getting started quickly without a lot of training.

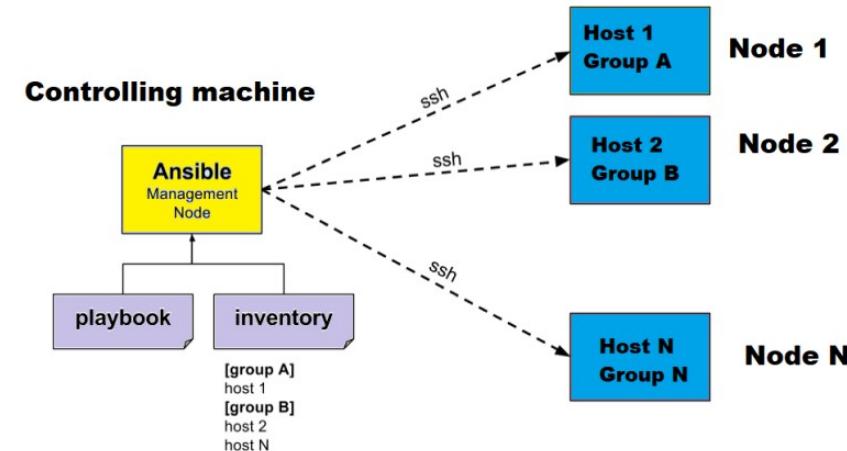


Why Ansible?



Ansible features

- ▶ **Management Node** : Controles the entire execution of the playbook
 - Enables SSH connection
 - Pushes and executes the small modules on the host machine
 - Installs the software
- ▶ **Inventory** : Provides the list of hosts where the Ansible need to be run
- ▶ Ansible removes the modules once those are installed so expertly
- ▶ There are **no daemons, servers, or databases** required



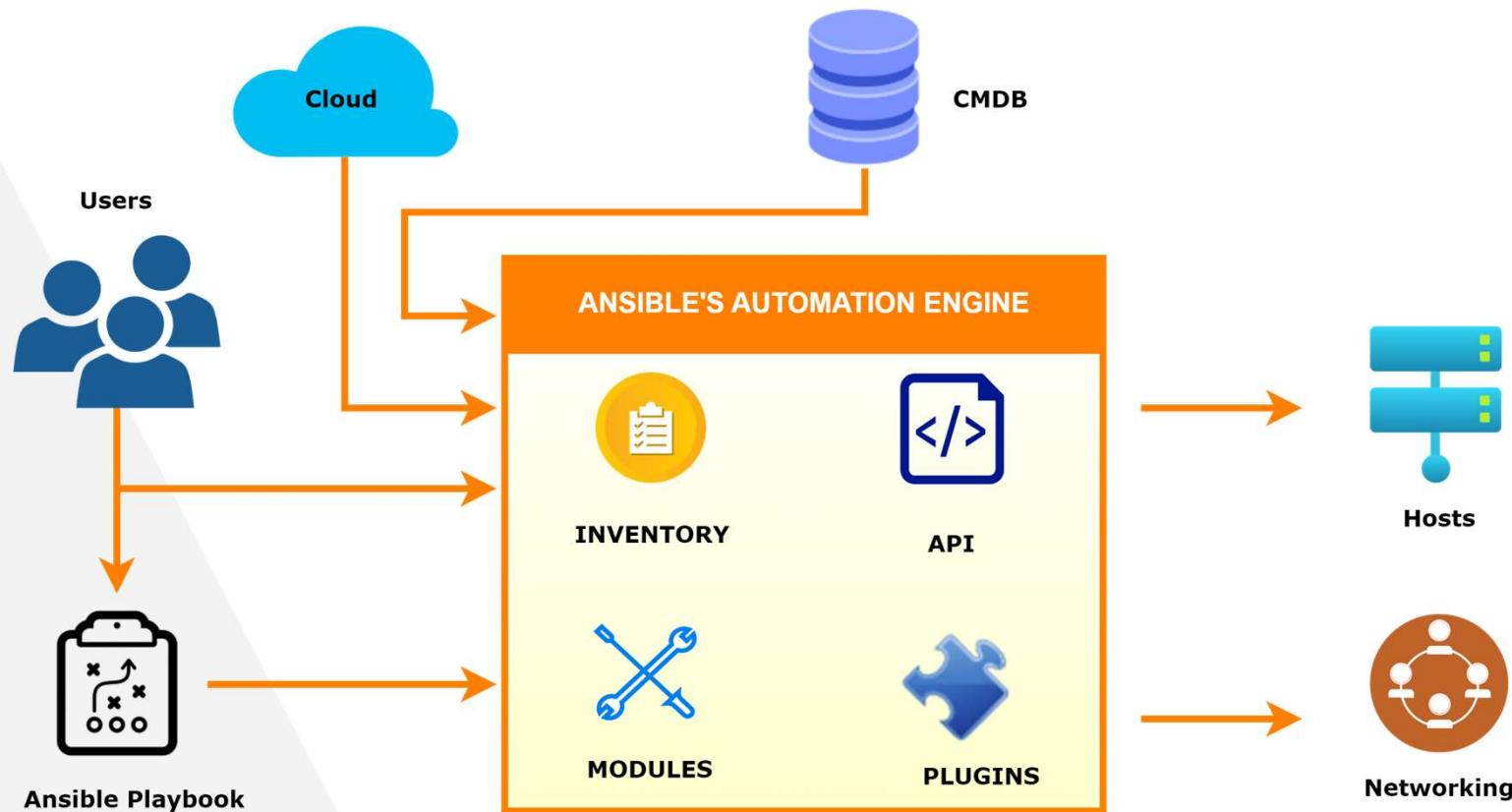
Ansible vocabulary (1/2)

- ▶ **Management Machine** : Machine on which Ansible is installed. Since Ansible is **agentless**, no software is deployed on the managed servers.
- ▶ **Inventory**: A file containing information about the managed servers
- ▶ **Playbook** : A simple file in YAML format defining the target servers and the tasks to be performed
- ▶ **Play** : An execution of Playbook
- ▶ **Task** : A block defining a procedure to be executed (e.g. create a user or a group, install a software package, etc)
- ▶ **Module** : Group of similar Ansible commands that are expected to be executed from the client-side

Ansible vocabulary (2/2)

- ▶ **Tag** : Name set of the task and could be used later for just issuing certain group tasks or specific tasks
- ▶ **Role** : Allows organizing the Playbooks and all the other necessary files (templates, scripts, etc) to facilitate the sharing and reuse of code.
- ▶ **Collection** : Includes a logical set of playbooks, roles, modules, and plugins
- ▶ **Facts** : Global variables containing information about the system (machine name, system version, network interface and configuration)
- ▶ **Notifier** : Attributed to the task that shall call the handler and when the output is modified.
- ▶ **Handlers** : To cause a service to be stopped or restarted in the event of a change

Ansible architecture (1/2)

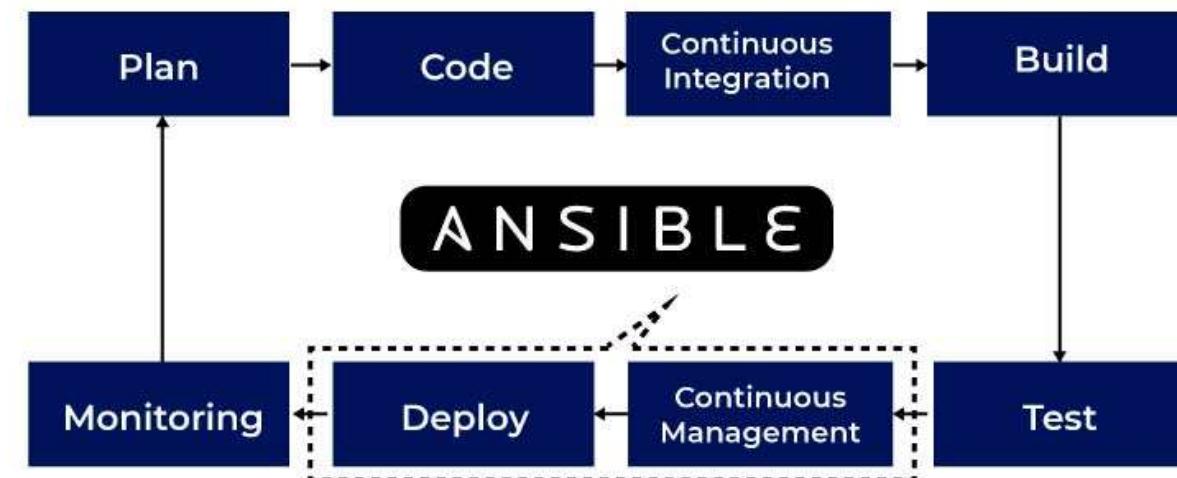


Ansible architecture (2/2)

- ▶ **Modules** : Ansible stacks all functions as module utilities for reducing the duplication and handling the maintenance
- ▶ **Plugins** : Amplify Ansible's Core functionality. They are executed on the control node
- ▶ **Inventory** : Depicts the machine that it shall handle in the file and gathers every machine in a group which you have chosen
- ▶ **APIs** : The Ansible APIs function as the bridge of Public and Private cloud services
- ▶ **CMDB** : Kind of repository that acts as the data warehouse for IT installations
- ▶ **Hosts** : Node systems that are automated using Ansible and machines like Linux, and Windows.
- ▶ **Networking** : Ansible is used for automating different networks and this uses the simple, powerful, secure agentless automation framework for IT development and operations.

Ansible in DevOps

The integration is a major factor for modern test-driven and application design. **Ansible** helps in integrating it by **providing a stable environment** for both the **Operations** and **Development** and it results in **Continuous orchestration**



Ansible Installation

► Install on Linux Control Node :

- **Redhat/CentOS :** \$ sudo yum install epel-release && sudo yum install ansible
- **Fedora :** \$ sudo dnf install ansible
- **Ubuntu :** \$ sudo apt-get install ansible
- **PIP :** \$ sudo pip install ansible

► Install on Windows (Via WSL)

```
$ sudo apt-get update  
$ sudo apt-get install python-pip git libffi-dev libssl-dev -y  
$ pip install --user ansible pywinrm
```

Ansible Configuration

- ▶ Ansible supports several sources for configuring its behavior
 - Configuration settings
 - Command-line options
 - Playbook keywords
 - Variables
- ▶ Each category overrides any information from all lower-precedence categories
- ▶ Last "defined" wins and overrides any previous definitions

Ansible Configuration settings

- ▶ Configuration settings include both values from the `ansible.cfg` file and environment variables.

- `ANSIBLE_CONFIG` (environment variable if set)
- `Ansible.cfg` (in the current directory)
- `~/.ansible.cfg` (in the home directory)
- `/etc/ansible/ansible.cfg` (default)

```
[lisa@DRDEVL ~]$ ansible --version
ansible 2.8.18
  config file = /home/lisa/.ansible.cfg
  configured module search path = ['/home/lisa/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.8 (default, Oct 11 2019, 15:04:54) [GCC 8.3.1 20190507 (Red Hat 8.3.1-4)]
```

- ▶ Content of config file

```
[msleczek@vm0-net projekt_A]$ cat ansible.cfg
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false

[privilegeEscalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
[msleczek@vm0-net projekt_A]$
```

Ansible SSH Key Authentication

Ansible uses SSH for authentication and assumes keys are in place (Controller)

```
$ ssh-keygen
[vagrant@ansible-controller ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:XUSUfRLl7IXDR1wC/wGTC9ruje+tcP7x/sz5kYxHT1g vagrant@ansible-controller
The key's randomart image is:
+---[RSA 2048]---+
 . 000
 oo".
 . +=*
 +.B.. o
 . S.. ...
 + = . +.o
 B o..o...=
 . + O+...=O
 . O=000X
+---[SHA256]---+
```

Activate PubkeyAuthentication & ChallengeResponseAuthentication (Target)

<pre>GNU nano 2.3.1 File: /etc/ssh/sshd_config #Port 22 #AddressFamily any #ListenAddress 0.0.0.0 #ListenAddress :: HostKey /etc/ssh/ssh_host_rsa_key #HostKey /etc/ssh/ssh_host_dsa_key HostKey /etc/ssh/ssh_host_ecdsa_key HostKey /etc/ssh/ssh_host_ed25519_key # Ciphers and keying #RekeyLimit default none # Logging #SyslogFacility AUTH SyslogFacility AUTHPRIV #LogLevel INFO # Authentication: #LoginGraceTime 2m #PermitRootLogin yes #StrictModes yes #MaxAuthTries 6 #MaxSessions 10 PubkeyAuthentication yes # The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2 # but this is overridden so installations will only check .ssh/authorized_keys</pre>	<pre>GNU nano 2.3.1 File: /etc/ssh/sshd_config #AuthorizedPrincipalsFile none #AuthorizedKeysCommand none #AuthorizedKeyCommandUser nobody # For this to work you will also need host keys in /etc/ssh/ssh_known_hosts #HostbasedAuthentication no # Change to yes if you don't trust ~/.ssh/known_hosts for # HostbasedAuthentication #IgnoreUserKnownHosts no # Don't read the user's ~/.rhosts and ~/.shosts files #IgnoreRhosts yes # To disable tunneled clear text passwords, change to no here! #PasswordAuthentication yes #PermitEmptyPasswords no #PasswordAuthentication no # Change to no to disable s/key passwords ChallengeResponseAuthentication yes #ChallengeResponseAuthentication no # Kerberos options #KerberosAuthentication no #KerberosOrLocalPasswd yes #KerberosTicketCleanup yes #KerberosGetAFSToken no #KerberosUseKuserok yes # GSSAPI options</pre>		
Get Help Exit	WriteOut Justify	Read File Where Is	Prev F Next F

Ansible SSH Authentication

► **Setting up and transferring SSH keys allows playbooks to be run automatically (Controller)**

```
$ ssh-copy-id vagrant@172.17.11.5
Last login: Wed Nov 23 23:20:13 2022 from 10.0.2.2
[vagrant@ansible-controller ~]$ ssh-copy-id vagrant@172.17.11.5
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/vagrant/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
Password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'vagrant@172.17.11.5'"
and check to make sure that only the key(s) you wanted were added.
```

► **Connecting with SSH key authorized on Target Node without asking password (Controller)**

```
$ ssh vagrant@172.17.11.5
[vagrant@ansible-controller ~]$ ssh vagrant@172.17.11.5
Last login: Wed Nov 23 22:57:22 2022 from 10.0.2.2
[vagrant@ansible-target ~]$ █
```

► **Using passwords is possible (optionally)**

```
GNU nano 2.3.1          File: /etc/ssh/sshd_config

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to no to disable s/key passwords
#ChallengeResponseAuthentication yes
ChallengeResponseAuthentication no
```

Ansible Inventory

- ▶ The default location for inventory is a file called `/etc/ansible/hosts`
- ▶ A different inventory file can be specified at the command line using the "`-i <path>`" option
- ▶ Multiple inventory file can be used at the same time
- ▶ Inventory can be pulled from dynamic or Cloud sources or different formats (YAML, ini)
- ▶ Example (Basic INI & YAML)

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

Module7-Ansible > Ansible-CentOS-VM > lampstack-ansible-playbook > hosts

```
1 [webservers]
2 172.17.11.5
3 [webservers:vars]
4 ansible_password=vagrant
5
```

Ansible Ad hoc Commands (1/4)

- ▶ **Ad-hoc commands are quick and easy but not reusable**
- ▶ **Uses the `/usr/bin/ansible` command-line tool to automate a single task on one or more managed nodes**
- ▶ **They are used when you want to issue some commands on one or more server(s)**
- ▶ **Format**

```
$ ansible [pattern] -m [module] -a "[ARGUMENTS]" or  
$ ansible <HOSTS> [-m <MODULE_NAME>] -a <"ARGUMENTS"> -u <USERNAME> [--become]
```

- **HOSTS** : Entry in the inventory file. To specify all host, use "all" or "*"
- **MODULE_NAME** : Modules available in the Ansible such as *file*, *copy*, *yum*, *shell* and *apt*
- **ARGUMENTS** : Pass values required by the module and can change according to the module used
- **USERNAME** : Specify the **user account** in which Ansible can execute commands.
- **Become** : Specify when to run operations that need ***sudo*** privilege.

Ansible Ad hoc Commands (2/4)

► Parallelism and shell commands

- To run reboot for all company servers in the group webservers in 11 parallel forks :

```
$ ansible webservers -a "sbin/reboot" -f 11 -u USERNAME
```

► Managing files for file transfer : For SCP (secure copy protocol) to transfer many files to multiple machines in parallel

- Transferring file on many machine in webservers group

```
$ ansible webservers -m copy -a "src=/etc/ssh/sshd_config dest=/tmp/sshd_config"
```

- Creating a new directory

```
$ ansible webservers -m file -a "dest=/tmp/new_dir mode=777 owner=user group=userg state=directory"
```

- Deleting directories (recursively) and files

```
$ ansible webservers -m file -a "dest=/tmp/new_dir state=absent"
```

Ansible Ad hoc Commands (3/4)

► Managing packages

- Ensure that yum package is installed

```
$ ansible webservers -m yum -a "name=nano state=present"
```

- Ensure a specific version of a package is installed

```
$ ansible webservers -m yum -a "name=nano-7.0 state=present"
```

- Ensure a package is the latest version

```
$ ansible webservers -m yum -a "name=nano state=latest"
```

- Ensure a package is not installed

```
$ ansible webservers -m yum -a "name=nano state=absent"
```

► Managing Users and Groups

- Create user accounts

```
$ ansible all -m user -a "name=gilbert password=<encrypted password>"
```

- Remove user accounts

```
$ ansible all -m user -a "name=gilbert state=absent"
```

Ansible Ad hoc Commands (4/4)

► Managing services

- Ensure a service is started on all webservers group

```
$ ansible webservers -m service -a "name=httpd state=started"
```

- Restart the service

```
$ ansible webservers -m user -a "name=httpd state=restarted"
```

- Check if a service is stopped

```
$ ansible webservers -m user -a "name=httpd state=stopped"
```

► Gathering facts

- Discovered variables about a system. Facts can be used to implement conditional execution of tasks and get ad hoc information about the systems.

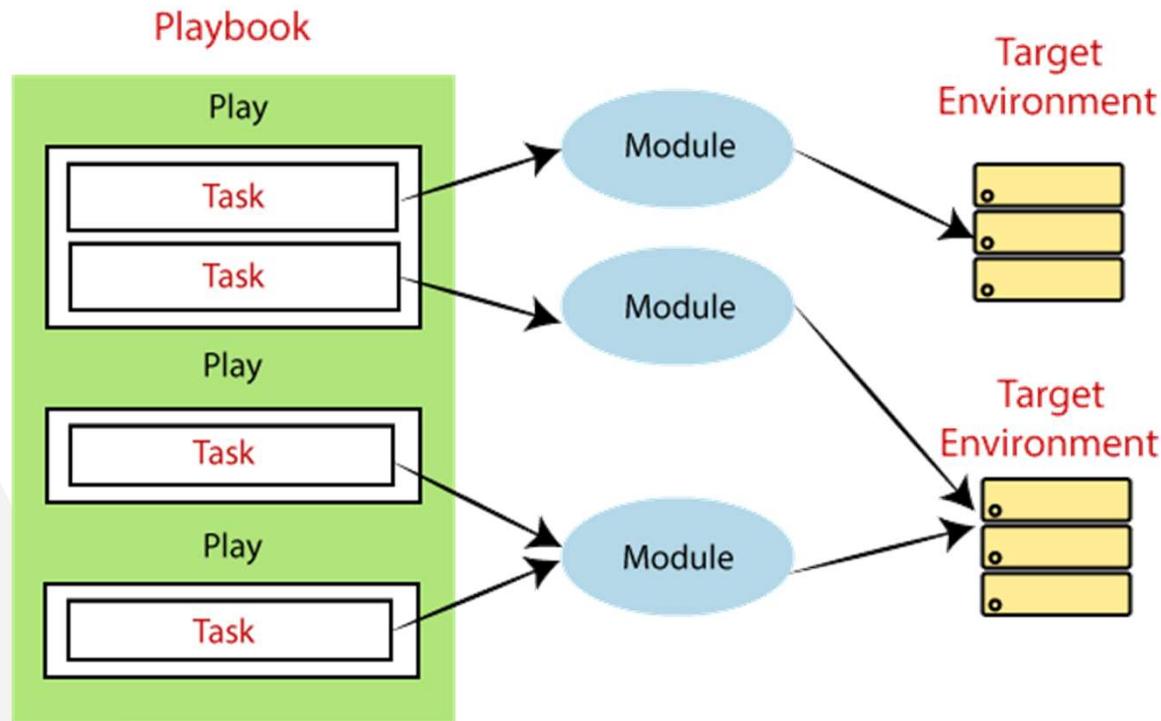
```
$ ansible all -m setup
```

```
[vagrant@ansible-controller ansible_demo]$ cat hosts
target1 ansible_host=172.17.11.5
[vagrant@ansible-controller ansible_demo]$ ansible target1 -m ping -i hosts
target1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Ansible Playbooks

- ▶ **Ansible Playbooks** offer a repeatable, reusable, simple configuration management and multi-machine deployment system
- ▶ Playbooks are the files where Ansible code is written (in **YAML** format): variables, files, templates, etc.
- ▶ These files are descriptions of the **desired state** of your systems
- ▶ Configuration management runbook with powerful control over scripting orchestrating
- ▶ Describes which hosts to configure, and ordered list of tasks to perform on those hosts
- ▶ Can use **Version Control System**

Ansible Playbook Structure



- ▶ Playbooks are a collection of one or more plays
- ▶ Each play map a set of instructions (tasks) defined against a particular host
- ▶ Tasks call modules

Ansible Playbook Example

PLAYBOOK

- ▶ **PLAY : Test connectivity to target servers**
- ▶ **HOSTS : All (All machine in the inventory file)**
- ▶ **TASK : Gathering facts**
- ▶ **TASK : Ping test**
- ▶ **PLAY : RECAP**

```
-  
  name : Test connectivity to target servers  
  hosts: all  
  tasks:  
    - name: Ping test  
      ping:
```

```
[vagrant@ansible-controller ansible_demo]$ ansible-playbook ping.yaml -i inventory.txt  
PLAY [Test connectivity to target servers] *****  
TASK [Gathering Facts] *****  
ok: [target1]  
TASK [Ping test] *****  
ok: [target1]  
PLAY RECAP *****  
target1 : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Ansible Playbook Example

PLAYBOOK

- ▶ **PLAY : Install Apache to target server**
- ▶ **HOSTS : All (All machine in the inventory file)**
- ▶ **TASK : Gathering facts**
- ▶ **TASK : Install httpd**
- ▶ **HANDLERS : Restart httpd**
- ▶ **PLAY : RECAP**

```
---
- hosts: all
  remote_user : vagrant
  become: yes
  tasks:
    - name: Install httpd
      yum: name=httpd update_cache=yes state=latest
      notify :
        - restart httpd
  handlers:
    - name : restart httpd
      service: name=httpd state=restarted
```

```
[vagrant@ansible-controller ansible_demo2]$ ansible-playbook apache.yml -i hosts
PLAY [all] ****
TASK [Gathering Facts] ****
ok: [172.17.11.5]
TASK [Install httpd] ****
changed: [172.17.11.5]
RUNNING HANDLER [restart httpd] ****
changed: [172.17.11.5]
PLAY RECAP ****
172.17.11.5 : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

```
[vagrant@ansible-controller ansible_demo2]$ cat hosts
[webservers]
172.17.11.5
```

Ansible Expressions (1/3)

▶ Conditionals

Use variables to depend on the value of other variables

- Based on ansible_facts

```
tasks:  
  - name: Shut down Debian flavored systems  
    ansible.builtin.command: /sbin/shutdown -t now  
    when: ansible_facts['os_family'] == "Debian"
```

- Using Register and When : Based on registered variables

Configure a service after it is upgraded by an earlier task

```
tasks:  
  
  - name: Register a variable  
    ansible.builtin.shell: cat /var/www/html/index.html  
    register: html_contents  
  
  - name: Use the variable in conditional statement  
    ansible.builtin.shell: echo "html server contains the word sale"  
    when: html_contents.stdout.find('sale') != -1
```

Ansible Expressions (2/3)

► Iterating over a simple list

- Using loop

For simple **loops** and is equivalent to **with_list**

```
---
- hosts: all
  remote_user : vagrant
  become: yes
  tasks:
    - name: Add several users
      ansible.builtin.user:
        name: "{{ item }}"
        state: present
        groups: "wheel"
    loop:
      - testuser1
      - testuser2
```

- Using with_

Be careful when changing **with_items** to **loop**, as **with_items** performed implicit single-level flattening.

The playbook above uses **with_items** to create users

```
---
- hosts: all
  become: true
  gather_facts: no
  tasks:
    - name: install packages
      apt:
        name: "{{item}}"
        state: present
    with_items:
      - apache2
      - ufw
      - mysql
```

Ansible Expressions (3/3)

► Using `var_files`

Variables may also be included in a separate file, using the `vars_files` section.

The var “`http_package`” is filled in “`vars.yml`”

```
---
- hosts: all
  vars_files:
    - vars.yml
  become: true
  tasks:
    - name: install packages
      apt:
        name: "{{http_package}}"
        state: present
```

► Jinja2 expression

Ansible uses Jinja2 templating to enable dynamic expressions and access to variables and facts.

Example : Create a template for a configuration file and deploy it to multiple environments

Ansible Playbook roles & vars

- ▶ Two of the key components of making playbooks reusable are Ansible variables and roles
- ▶ Roles allow you to call a set of variables, tasks, and handlers by simply specifying a defined role

```
---  
- hosts: linuxservers  
  tasks:  
    - name: Install Apache Web Server  
      yum: name=httpd state=latest  
    - name: upload index page  
      get_url: url=http://www.purple.com/faq.html dest=/var/www/html/index.html  
  
    notify:  
      - openport80  
      - startwebserver  
  
  handlers:  
    - name: openport80  
      service: name=httpd state=started  
    - name: startwebserver  
      firewalld: port=80/tcp permanent=true state=enabled immediate=yes  
  
[linuxservers]  
ansibleclient2 ansible_host=172.17.11.5 webserver_pkg=httpd  
  
[linuxservers:vars]  
http_server_port=80
```

Ansible Playbook roles & vars

► Creating a roles

```
$ sudo ansible-galaxy init linuxwebserver
```

```
$ sudo tree linuxwebserver
```

- **Defaults** : Default variables for the role
- **Files** : Contains files which can be deployed via this role
- **Handlers** : Contains handlers, which may be used by this role or outside
- **Meta** : Defines some meta data for this role
- **Tasks**: Contains the main list of tasks to be executed by the role
- **Templates** : Contains templates which can be deployed via this role
- **Tests** : Contains test example using this role
- **Vars** : Other variables for the role

```
[vagrant@node1 ~]$ cd /etc/ansible/roles/  
[vagrant@node1 roles]$ sudo ansible-galaxy init linuxwebserver  
- Role linuxwebserver was created successfully
```

```
[vagrant@node1 roles]$ sudo tree linuxwebserver/  
linuxwebserver/  
└── defaults  
    └── main.yml  
── files  
── handlers  
    └── main.yml  
── meta  
    └── main.yml  
── README.md  
── tasks  
    └── main.yml  
── templates  
── tests  
    └── inventory  
        └── test.yml  
── vars  
    └── main.yml
```

8 directories, 8 files

Ansible Playbook roles & vars

► Edit tasks files

```
$ sudo nano linuxwebserver/tasks/main.yml
```

GNU nano 2.3.1

File: linuxwebserver/tasks/main.yml

```
---
# tasks file for linuxwebserver

- name: Install Apache Web Server
  yum: name=httpd state=latest
  notify:
    - openport80
    - startwebserver
```

► Edit Handlers files

```
$ sudo nano linuxwebserver/handlers/main.yml
```

GNU nano 2.3.1

File: linuxwebserver/handlers/main.yml

```
---
# handlers file for linuxwebserver

- name: openport80
  service: name=httpd state=started

- name: startwebserver
  firewalld: port=80/tcp permanent=true state=enabled immediate=yes
```

Ansible Playbook roles & vars

► Write playbook which call the role created

```
$ pwd
$ sudo mkdir ~/deploy_with_role
$ cd ~/deploy_with_role
$ sudo nano webserver-roles.yml
```

```
[vagrant@node1 roles]$ pwd
/etc/ansible/roles
[vagrant@node1 roles]$ sudo mkdir ~deploy_with_role
[vagrant@node1 roles]$ cd ~deploy_with_role
[vagrant@node1 deploy with role]$ sudo nano webserver-roles.yml
```

GNU nano 2.3.1 File: webserver-roles.yml

```
---
- hosts: linuxservers
  remote_user : vagrant
  become: yes
  roles:
    - webserver

- hosts: ansibleclient2
  remote_user : vagrant
  become: yes
  tasks:
    - name: upload index page
      url:
        url: http://www.purple.com/faq.html
        method : GET
        dest : /var/www/html/index.html
        force_basic_auth: yes
        validate_certs : no
```

► Edit Inventory files

```
$ sudo nano hosts
[vagrant@node1 deploy with role]$ sudo nano hosts
```

GNU nano 2.3.1 File: hosts

```
[linuxservers]

ansibleclient2 ansible_host=172.17.11.5

[linuxservers:vars]
http_server_port=80
```

Ansible Playbook roles & vars

Result and display

```
$ ansible-playbook webserver-roles.yml -i hosts
```

```
[vagrant@node1 deploy_with_role]$ ansible-playbook webserver-roles.yml -i hosts

PLAY [linuxservers] ****
TASK [Gathering Facts] ****
ok: [ansibleclient2]

TASK [webserver : Install Apache Web Server] ****
ok: [ansibleclient2]

PLAY [ansibleclient2] ****
TASK [Gathering Facts] ****
ok: [ansibleclient2]

TASK [upload index page] ****
changed: [ansibleclient2]

PLAY RECAP ****
ansibleclient2      : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

The screenshot shows a web page from a browser window. The address bar indicates the URL is 172.17.11.5. Below the address bar, there is a link to "Skip to main content". The main content area features a large headline "Save on A Mattress Set" with a subtext "Make magic happen overnight and save up to \$800 on a mattress and Ascent® Adjustable Base." There are two sections of promotional offers:

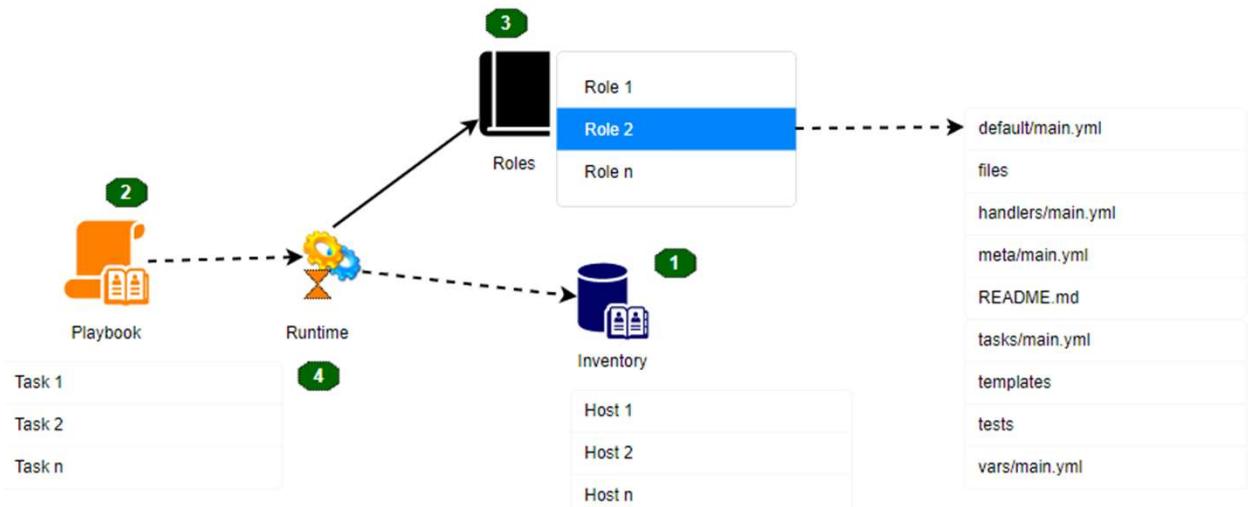
- Up to \$300 off a Mattress**
 - Limited Time! Get up to \$800 off a mattress set this season.
- Save \$500 on the Ascent® Adjustable Base***
 - Available in TwinXL, Queen, King, Cal King, and Split King.

Below these offers, there is a note about the bundle: "30% off the NewDay™ Essential Bundle". Further down, there are links for "Shop Mattresses" and "Terms Apply". At the bottom of the page, there is a "Feedback" section with a link to "Includes pillows, sheets, bedding, and seating".

Ansible Playbook roles & vars

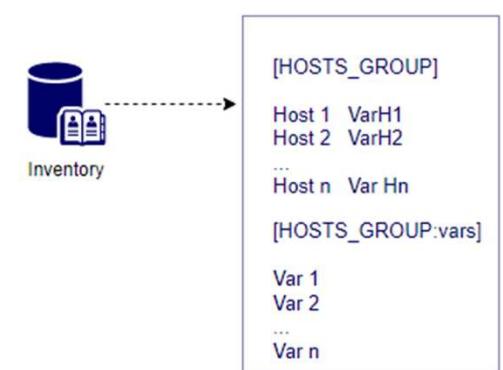
- ▶ Ansible variables are one of the ways that make playbooks more generic.
- ▶ Variables can be defined in multiple locations

- Inventory file
- Playbook
- Role definition
- Runtime



Ansible Playbook roles & vars/Inventory

- ▶ Variables can be assigned right along with the host definition in inventory
- ▶ Variables are set at both a group and individual host level
- ▶ Actions : Task on role definition/Handlers/Inventory



```
[vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/tasks/main.yml
GNU nano 2.3.1
File: /etc/ansible/roles/linuxwebserver/tasks/main.yml
```

```
---
```

```
# tasks file for linuxwebserver
- name: Install Apache Web Server
  yum: name={{webserver_pkg}} state=latest
  notify:
    - openport80
    - startwebserver
```

```
[vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/handlers/main.yml
GNU nano 2.3.1
File: /etc/ansible/roles/linuxwebserver/handlers/main.yml
```

```
---
```

```
# handlers file for linuxwebserver
- name: openport80
  service: name=httpd state=started
- name: startwebserver
  firewalld: port={{http_server_port}}/tcp permanent=true state=enabled immediate=yes
```

```
[vagrant@node1 deploy with role]$ sudo nano hosts
```

```
GNU nano 2.3.1
File: hosts
```

```
[linuxservers]
```

```
ansibleclient2 ansible_host=172.17.11.5 webserver_pkg=httpd
```

```
[linuxservers:vars]
```

```
http_server_port=80
```

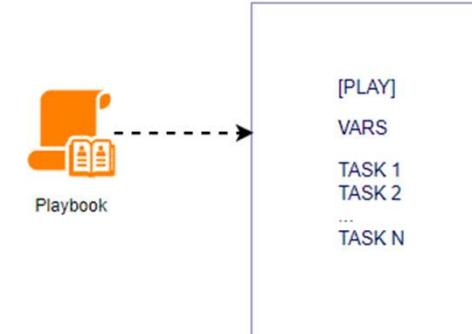
Ansible Playbook roles & vars/Playbook

► The same variables can also be defined directly within the plays without calling the role

```
---
- hosts: linuxservers
  remote_user : vagrant
  Become : yes
  vars :
    http_server_port:80
    webserver_pkg : httpd

  tasks:
    - name: Install Apache Web Server
      yum: name={{webserver_pkg}} state=latest
      notify:
        - openport80
        - startwebserver
    - name: upload index page
      get_url: url=http://www.purple.com/faq.html dest=/var/www/html/index.html

  handlers:
    - name: openport80
      service: name=httpd state=started
    - name: startwebserver
      firewalld: port={{http_server_port}}/tcp permanent=true state=enabled immediate=yes
```



[PLAY]
VARS
TASK 1
TASK 2
...
TASK N

[linuxservers]
ansibleclient2 ansible_host=172.17.11.5

[linuxservers:vars]

Ansible Playbook roles & vars/Role

- ▶ Variables can be defined in either the “vars” or “default” directory of the role
- ▶ “Default” directory stores values that are default settings which can be overridden when using “vars” directory

```
[vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/tasks/main.yml
GNU nano 2.3.1
File: /etc/ansible/roles/linuxwebserver/tasks/main.yml

---
# tasks file for linuxwebserver

- name: Install Apache Web Server
  yum: name={{webserver_pkg}} state=latest
  notify:
    - openport80
    - startwebserver
```

```
[vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/handlers/main.yml
GNU nano 2.3.1
File: /etc/ansible/roles/linuxwebserver/handlers/main.yml

---
# handlers file for linuxwebserver

- name: openport80
  service: name=httpd state=started

- name: startwebserver
  firewalld: port={{http_server_port}}/tcp permanent=true state=enabled immediate=yes
```

```
[vagrant@node1 deploy with role]$ sudo nano hosts
GNU nano 2.3.1
File: hosts

[linuxservers]
ansibleclient2 ansible_host=172.17.11.5
[linuxservers:vars]
```

```
[vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/vars/main.yml
GNU nano 2.3.1
File: /etc/ansible/roles/linuxwebserver/vars/main.yml

---
# vars file for linuxwebserver

webserver_pkg: httpd
http_server_port: 80
```

Ansible vars precedence model

- ▶ Priority increases as you move down the list
- ▶ To override absolutely any other defined variable, you should do it at runtime with the “`-e`” flag



Ansible Playbook roles & vars/Runtime

▶ Despite having changed the packages to be installed and the server port on the inventory, their explicit definition via the runtime prevails

```
[vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/tasks/main.yml      [vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/handlers/main.yml
GNU nano 2.3.1                                         File: /etc/ansible/roles/linuxwebserver/tasks/main.yml   GNU nano 2.3.1                                         File: /etc/ansible/roles/linuxwebserver/handlers/main.yml
---
# tasks file for linuxwebserver
-
- name: Install Apache Web Server
  yum: name={{webserver_pkg}} state=latest
  notify:
    - openport80
    - startwebserver
  -
- name: openport80
  service: name=httpd state=started
  -
- name: startwebserver
  firewalld: port={{http_server_port}}/tcp permanent=true state=enabled immediate=yes

[vagrant@node1 deploy with role]$ sudo nano /etc/ansible/roles/linuxwebserver/vars/main.yml
GNU nano 2.3.1                                         File: /etc/ansible/roles/linuxwebserver/vars/main.yml
---
# vars file for linuxwebserver
webserver_pkg: tomcat
http_server_port: 8080

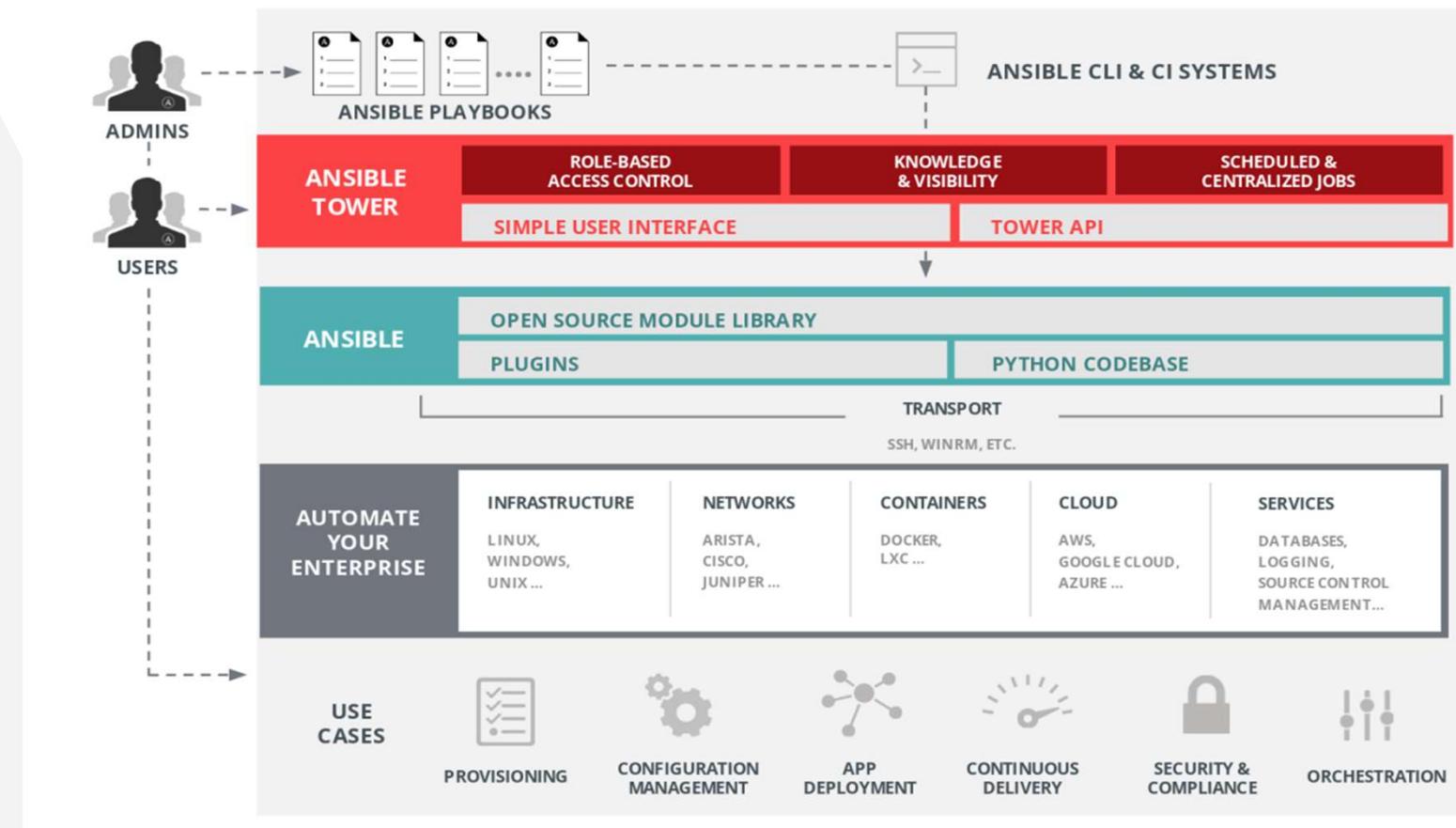
[vagrant@node1 deploy with role]$ ansible-playbook webserver-roles.yml -i hosts -e 'webserver_pkg=httpd http_server_port=80'
[vagrant@node1 deploy_with_role]$ ansible-playbook webserver-roles.yml -i hosts -e 'webserver_pkg=httpd http_server_port=80'
PLAY [linuxservers] ****
TASK [Gathering Facts] ****
ok: [ansibleclient2]
TASK [webserver : Install Apache Web Server] ****
ok: [ansibleclient2]
PLAY [ansibleclient2] ****
TASK [Gathering Facts] ****
ok: [ansibleclient2]
TASK [upload index page] ****
changed: [ansibleclient2]
PLAY RECAP ****
ansibleclient2 : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Ansible Galaxy

- ▶ Ansible galaxy is essentially a large public repository of Ansible roles
- ▶ Contains many roles that are constantly evolving and increasing

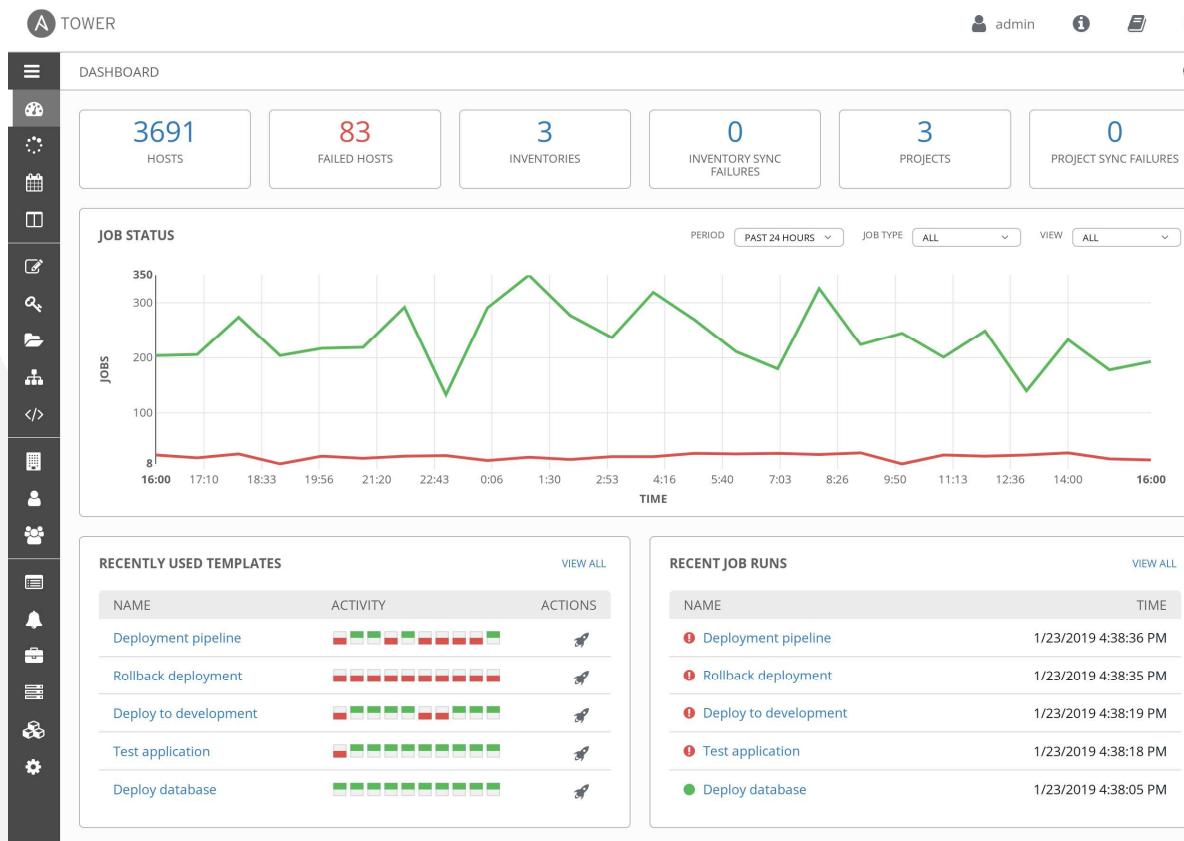
The screenshot shows the Ansible Galaxy website interface. On the left is a dark sidebar with navigation links: Home, Search, and Community. The main content area has a header for 'Community Authors > community > docker'. The central part displays the 'docker' role card, which includes a red circular icon with a white letter 'A', the role name 'docker', a brief description 'Modules and plugins for working with Docker', and a 'Details' tab selected. To the right of the card are metrics: a green checkmark icon, '4.1 / 5 Score', '10836512 Downloads', and buttons for 'Login to Follow', 'Issue Tracker', 'Repo', and 'Website'. Below the card, there are two sections: 'Info' and 'Content Score'. The 'Info' section contains installation instructions (\$ ansible-galaxy collection install community.docker), an 'Install Version' dropdown set to '3.2.1 released 19 days ago (latest)', and a 'Tags' field with 'docker'. It also lists the 'Docker Community Collection' and provides links for 'docs', 'Azure Pipelines' (status 'succeeded'), and 'coverage' (75%). A note states that the collection does not support Windows targets. The 'Content Score' section shows a 'Community Score' of 4.1/5 based on 3 surveys, with a progress bar and survey details for 'Quality of docs?', 'Ease of use?', 'Does what it promises?', 'Works without change?', and 'Ready for production?'.

Ansible Tower



Ansible Tower

► Ansible Tower (AWX) is a web-based solution that makes Ansible even more easy to use



MODULE IV-2 : Other Tools

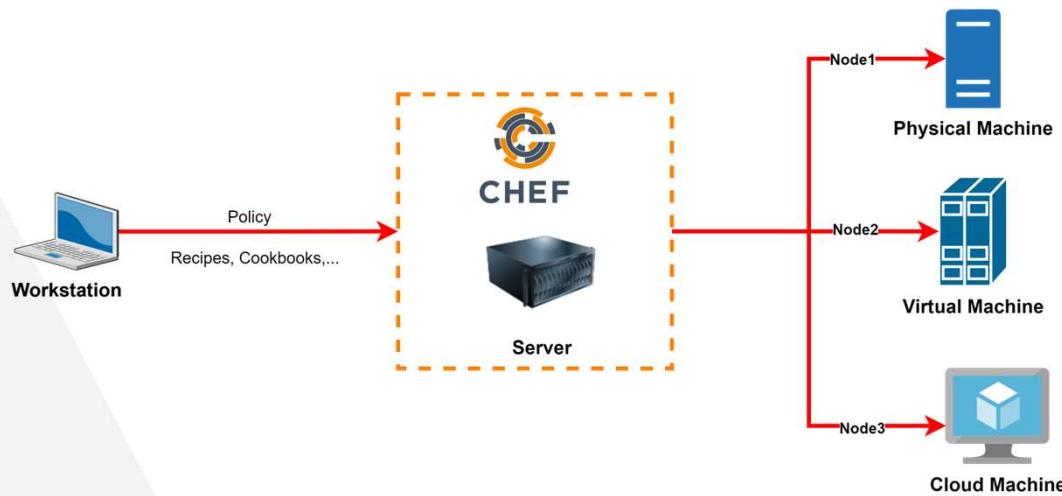
PLAN

- ▶ **CHEF**
 - Chef Expression
 - Chef Workstation
 - Chef Server
 - Chef Nodes
 - Chef Tools
- ▶ **PUPPET**
 - Architecture
 - Commands



Chef

- ▶ Chef® is an open source, systems management and cloud infrastructure automation platform
- ▶ Chef transforms infrastructure into code to automate server deployment and management.
- ▶ Chef is a **configuration management** tool for dealing with machine setup on physical servers, virtual machines and in the cloud
- ▶ Used by several companies including **Facebook, Yahoo, Etsy**.
- ▶ There are three major Chef components : **Workstation, Server, and Nodes**



Chef Expression (1/2)

► **Recipe**

It can be defined as a collection of attributes which are used to manage the infrastructure

► **Cookbook**

It is a collection of recipes

► **Resource**

It is the basic component of a recipe used to manage the infrastructure with different kind of states

- **Package**

- **Service**

- **User**

- **Group**

- **Template**

- **Cookbook_file**

- **Execute**

- **File**

Chef Expression (2/2)

► Roles

- Help to improve configuration for redundant servers that all perform the same basic tasks.
- It is a categorization that describes what a specific machine is supposed to do
- Determines tools and settings to give to a specific machine

► Environment

A designation meant to help an administrator know what stage of the production process a server is a part of. By default, an environment called “`_default`” is created. **Environments** can be created to **tag a server** as part of a process group

► Data Bags

- Data Bags store global variables as JSON data .
- Indexed for searching and can be loaded by a Cookbook or accessed during search

Chef Workstation

- The **Workstation** is the location from which all of **Chef configurations are managed**
- Holds all the configuration data that can later be pushed to the central **Chef Server**
- These configurations are tested in the workstation **before pushing it into the Chef Server**
- A **workstation** consists of a **command-line tool** called **Knife**, that used to interact with the Chef Server
- There can be multiple Workstations that together manage the central Chef Server
- Workstations are responsible for performing the below functions :
 - Writing **Cookbooks** and **Recipes** that will later be pushed to the central **Chef Server**
 - **Managing Nodes** on the central **Chef Server**

Chef Workstation/Recipes

- ▶ A **Chef recipes** is a file that groups related resources, such as everything needed to configure a web server, database, or a Load balancer.
- ▶ These Recipes describe a series of resources that should be in a particular state, i.e. Packages that should be installed, services that should running, or files that should be written.

```
#linux install httpd server, add fw rules and start service

package "httpd" do
  action :install
end

include_recipe "apache::fwrules"

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

Chef Workstation/Cookbook

- ▶ A **Chef cookbook** is the fundamental unit of configuration and policy distribution
- ▶ Defines a scenario and contains everything that is required to support that scenario
 - **Recipes** that specify which Chef Infra built-in resources to use (in order)
 - **Attributes** values which allow environment-based configurations such as **dev** or **prod**
 - **Custom resources** for extending Chef Infra beyond the built-in resources
 - **Files** and **Templates** for distributing information to systems
 - **Metadata (metadata.rb)** which contains information about the cookbook such as the name, description and version

```
#linux install httpd server, add fw rules and start service

package "httpd" do
  action :install
end
```

Chef Workstation / Managing Nodes

The **Workstation** will have the required command-line utilities, to control and manage every aspect of the central Chef Server

- ▶ **Adding a new Node to the central Chef Server**
- ▶ **Deleting a Node from the central Chef Server**
- ▶ **Modifying Node configurations etc.**

To perform above functions, Workstation have two major components :

- ▶ **Knife Utility** : Command line tool for communication with the central Chef Server (Adding, removing, changing configurations of Nodes – Upload Cookbooks and roles)
- ▶ **Local Chef repository** : Store every configuration component of central Chef Server (With Knife utility)

Chef Server

- ▶ **Chef Server** acts as a hub for configuration data
- ▶ Stores **Cookbooks**, the **policies** applied to Nodes, and **metadata** that describes each registered Node that is being managed by the **Chef-Client**
- ▶ Send configuration details (Recipes, Templates, and file distributions) to the Node through **Chef-client**
- ▶ Scalable approach which distributes the configuration effort throughout the organization

Chef Nodes

Nodes can be a cloud-based virtual server or an on-premise physical server, that is managed using central Chef Server.

Chef-client is the main component agent present on the **Node** that will establish communication with the Central Chef Server

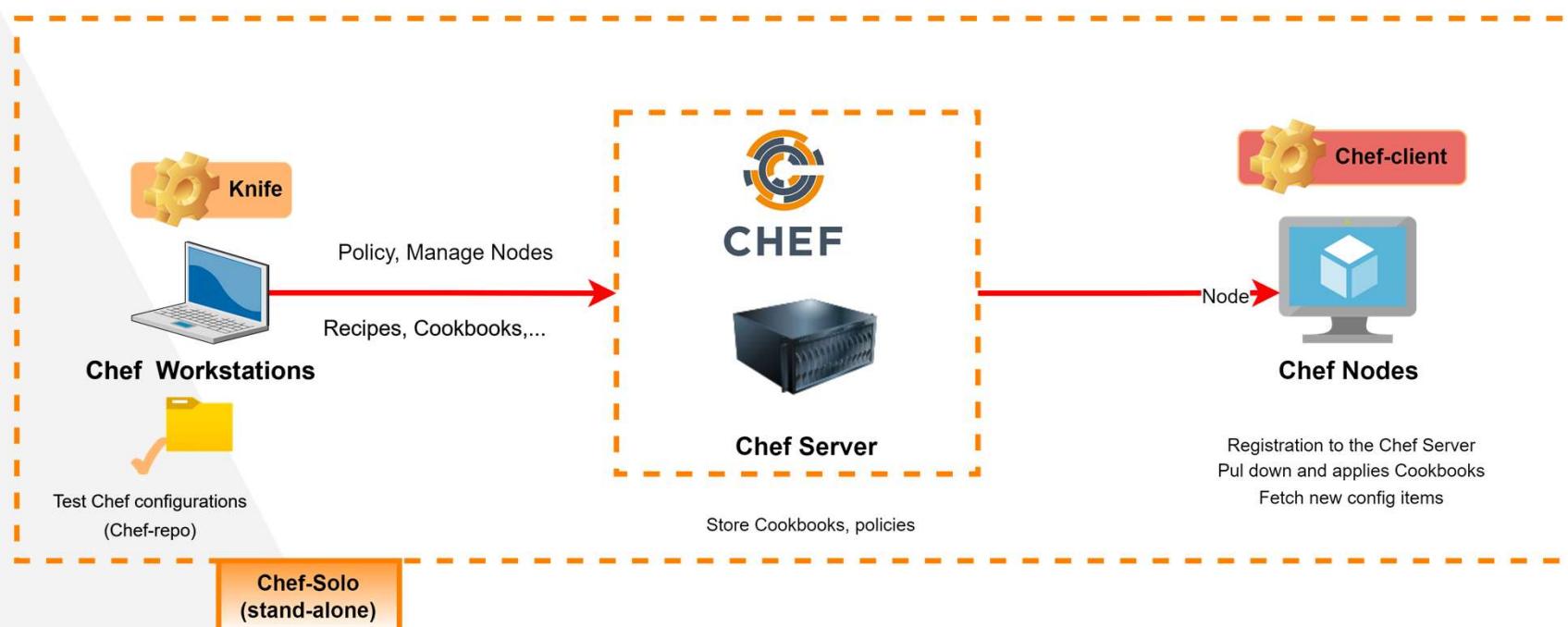
The following functions are performed by Chef Client :

- ▶ **Responsible for interacting with the central Chef Server**
- ▶ **Manages the initial registration of the Node to the central Chef Server**
- ▶ **Pull down and applies Cookbooks on the Node**
- ▶ **Periodic polling of the central Chef Server to fetch new configuration items if any**

Chef Tools

- ▶ **Chef-solo (standalone mode)**
 - **Chef-solo** is a command that executes Chef Client in a way that does not require the Chef Server to converge Cookbooks
 - Run as a daemon
 - Uses Chef Client's local mode and does not support some functionality (Centralized distribution of Cookbooks and Authentication)
 - Cookbook can be run from a **local directory** and a **URL** at which a "**tar.gz**" archive is located
- ▶ **Knife**
 - Interaction between a local **chef-repo** and the **Chef Server**
 - Helps users to manage Nodes, **Cookbooks** and **recipes**, **Roles**, **Environments**, and **Data Bags**

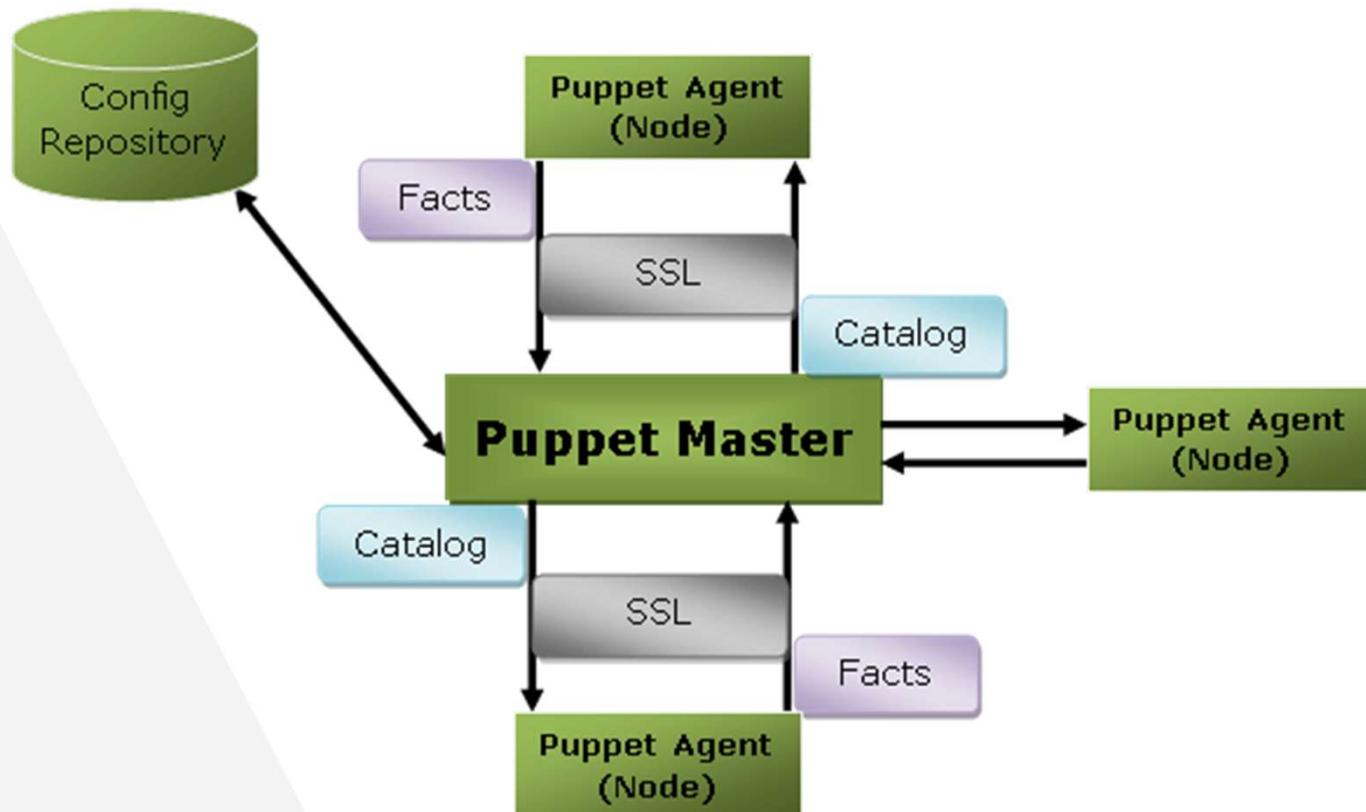
Chef / Workflow



Puppet

- ▶ **Puppet** uses a declarative language that models the infrastructure as a series of resources
- ▶ **Manifests** consist of a set of **JSON files**, pull together these resources and define the desired state of the final platform
- ▶ **Puppet** stores **manifests** on the servers and uses them to create compiled configuration instructions as needed, feeding them to agents via **REST APIs**
- ▶ **Facter** is a puppet tool that discovers and reports **facts** about nodes which are then used to create the **manifests** and **configurations**.
- ▶ **Facts** include built-in details of the overall platform
- ▶ Puppet architecture , master and agents (can run in a server-only model with command line access)

Puppet Architecture



Puppet Architecture

► Puppet Master

- Handles all the configuration related process in the form of puppet codes
- **Create catalog** and sends it to the targeted Puppet Agent
- Installed on a Linux based system
- **SSL certificates** are checked and marked

► Puppet Node (Slave or Agent)

- Client installed, maintained, and managed by the Puppet Master
- Agent can be configured on any OS
- Applies configurations receive from Master to get the system into a **desired state**
- Send a report back to the Master

Puppet Architecture

► **Config Repository**

Storage area where all the servers and nodes related configurations are stored, and these configurations can be pulled as per requirements

► **Facts**

Key-value data pair. It contains information about the node or Master Machine.

It represents a puppet client states such as Operating System, network interface, IP Address, etc.

► **Catalog**

Compiled format which result from the entire configuration and manifest files.

Catalog can be applied on the target machine.

Puppet Command

COMMANDS	DESCRIPTION
Puppet factor	Show all factors
Puppet agent --enable	Enable puppet agent to run on node
Puppet resource package	Show all installed packages
Puppet resource	Show all managed resources
Puppet module list	List all installed module
Puppet config print all	Print all configuration settings

PART V. Service Operations

...

MODULE V-1 : IT Operations and Monitoring

PLAN

- ▶ **DEVOPS MONITORING**
- ▶ **PROMETHEUS**
 - Architecture
 - Configuration file
 - Metric types
 - Exporter
- ▶ **GRAFANA**
 - Data source
 - Dashboard



DevOps Monitoring

- ▶ **DevOps Monitoring** is the practice of **tracking and measuring** the **performance and health** of systems and applications in order to **identify and correct** issues early
- ▶ Collect data on everything from CPU utilization to disk space to application response times
- ▶ Helps teams avoid outage or degradation of service
- ▶ Be able to debug and gain insight
- ▶ **Continuous monitoring** is the process of **regularly and vigilantly** checking systems, networks, and data for signs of **performance degradation**

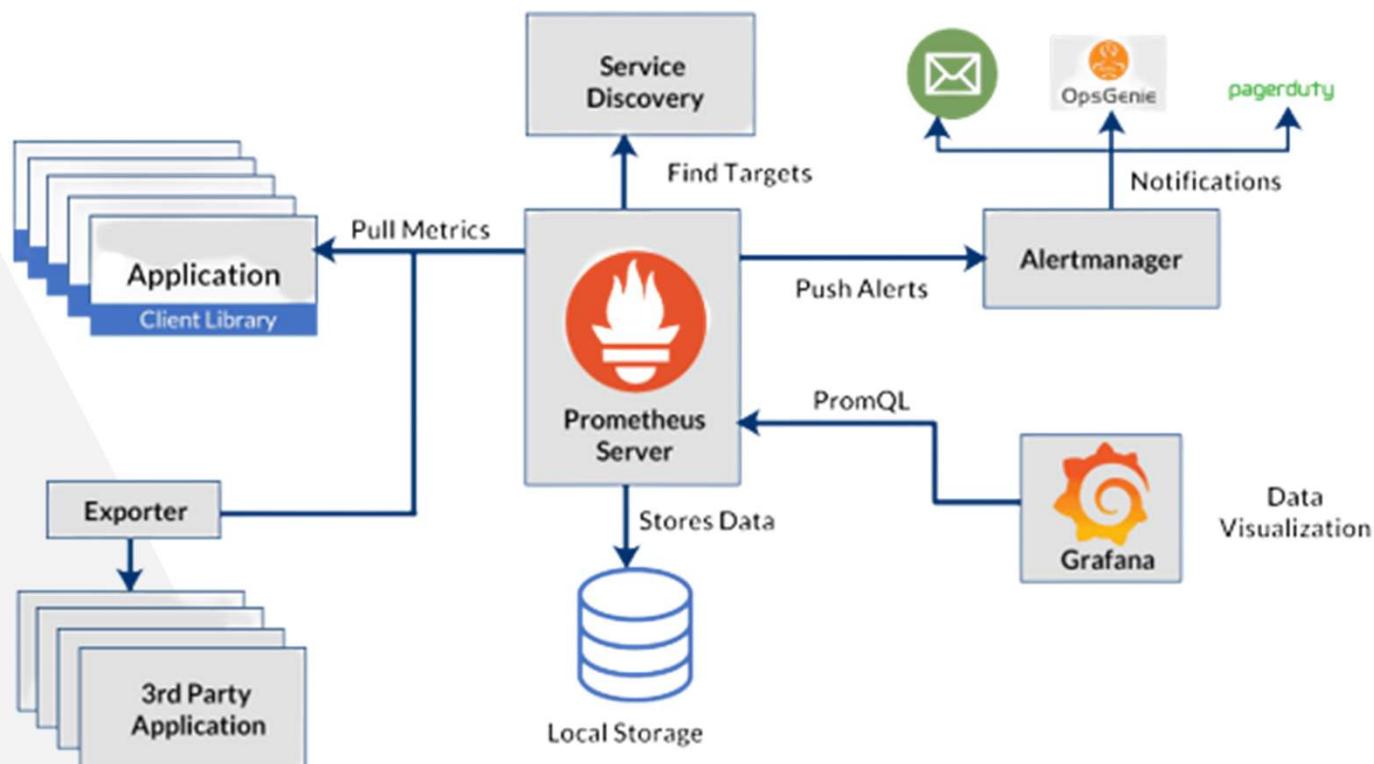
Prometheus

- ▶ **Prometheus** is an open-source technology designed to provide monitoring and alerting functionality for Cloud-native environments.
- ▶ Collect and store metrics as **time-series** data, recording information with a timestamp
- ▶ Collect and record labels (key-value pairs)

Key features of Prometheus include :

- ▶ **Multidimensional data model**
- ▶ **PromQL**
- ▶ **No reliance on distributed storage**
- ▶ **Pull model**
- ▶ **Pushing time-series data**
- ▶ **Monitoring target discovery**
- ▶ **Visualization**

Prometheus Architecture



Prometheus Architecture

- ▶ **Prometheus server** : Scrapes and stores time series data
- ▶ **Client Libraries** : Help in pulling metrics from the applications and send it to the Prometheus Server (C#, Node.js...)
- ▶ **Service Discovery** : Plays a key role in dynamic environments. Enables Prometheus to identify the applications it needs to monitor and pull metrics from
- ▶ **Local storage** : Store the data locally in a custom data store
- ▶ **Alertmanager** : Receive alerts from the Prometheus Server and turn them into notifications
- ▶ **Exporter** : Libraries and servers which help in exporting existing metrics from third-party systems as Prometheus metrics (HAProxy, StatsD, Graphite)
- ▶ **Node Exporter** : For physical and virtual machine metrics – hardware and kernel metrics (Network, disk, CPU, RAM)
- ▶ **Data Visualization** : Provides direct access to enter any expression and visualize its result either in a table or graphed over time (Console templates, Grafana)

Prometheus Configuration file (1/2)

Default configuration file has four YAML blocks

- ▶ **Global** : Contains global settings for controlling the Prometheus server's behaviour
- **Scrape Interval** : Specifies the interval between scrapes of any application or service
- **Evaluation Interval** : Tells Prometheus how often to evaluate its rules
- ▶ **Alerting** : Provided by an independent alert management tool called **Alertmanager**.

List each alertmanager used by the Prometheus server

```

1   global:
2     scrape_interval: 15s
3     evaluation_interval: 15s
4
5   alerting:
6     alertmanagers:
7       - static_configs:
8         - targets:
9           # -alertmanager:9093
10
11   rule_files:
12     #- "first.rules"
13     #- "second.rules"
14
15   scrape_configs:
16     - job_name: 'prometheus'
17       scrape_interval: 5s
18       static_configs:
19         - targets: ['172.17.8.104:9090']
20
21     - job_name: 'node1'
22       scrape_interval: 5s
23       static_configs:
24         - targets: ['172.17.8.104:9100']
25
26     - job_name: 'node2'
27       scrape_interval: 5s
28       static_configs:
29         - targets: ['172.17.8.105:9100']

```

Prometheus Configuration file (2/2)

- ▶ **Rule file** : List of files that contain recording or alerting rules.
- **Recording rules** : Allow you to precompute frequent and expensive expressions and to save their result as derived time series data
- **Alerting rules** : Allow you to define alert conditions. Prometheus will re-evaluate these rules every 15 seconds (**#evaluation_interval**).
- ▶ **Scrape Configuration** : Specifies all the targets that Prometheus will scrape

```

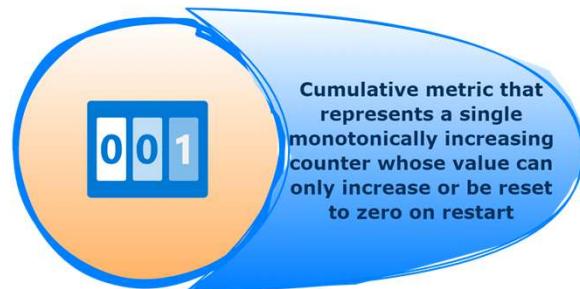
1   global:
2     scrape_interval: 15s
3     evaluation_interval: 15s
4
5   alerting:
6     alertmanagers:
7       - static_configs:
8         - targets:
9           # -alertmanager:9093
10
11  rule_files:
12    #- "first.rules"
13    #- "second.rules"
14
15  scrape_configs:
16    - job_name: 'prometheus'
17      scrape_interval: 5s
18      static_configs:
19        - targets: ['172.17.8.104:9090']
20    - job_name: 'node1'
21      scrape_interval: 5s
22      static_configs:
23        - targets: ['172.17.8.104:9100']
24    - job_name: 'node2'
25      scrape_interval: 5s
26      static_configs:
27        - targets: ['172.17.8.105:9100']
28
29

```

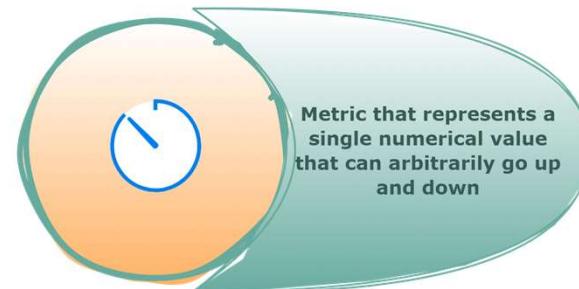
Prometheus Metric Types

Prometheus client libraries offer four core metric types

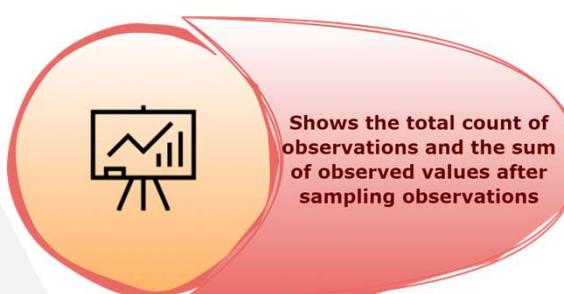
COUNTER



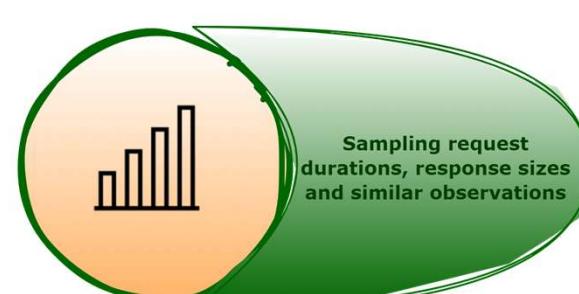
GAUGE



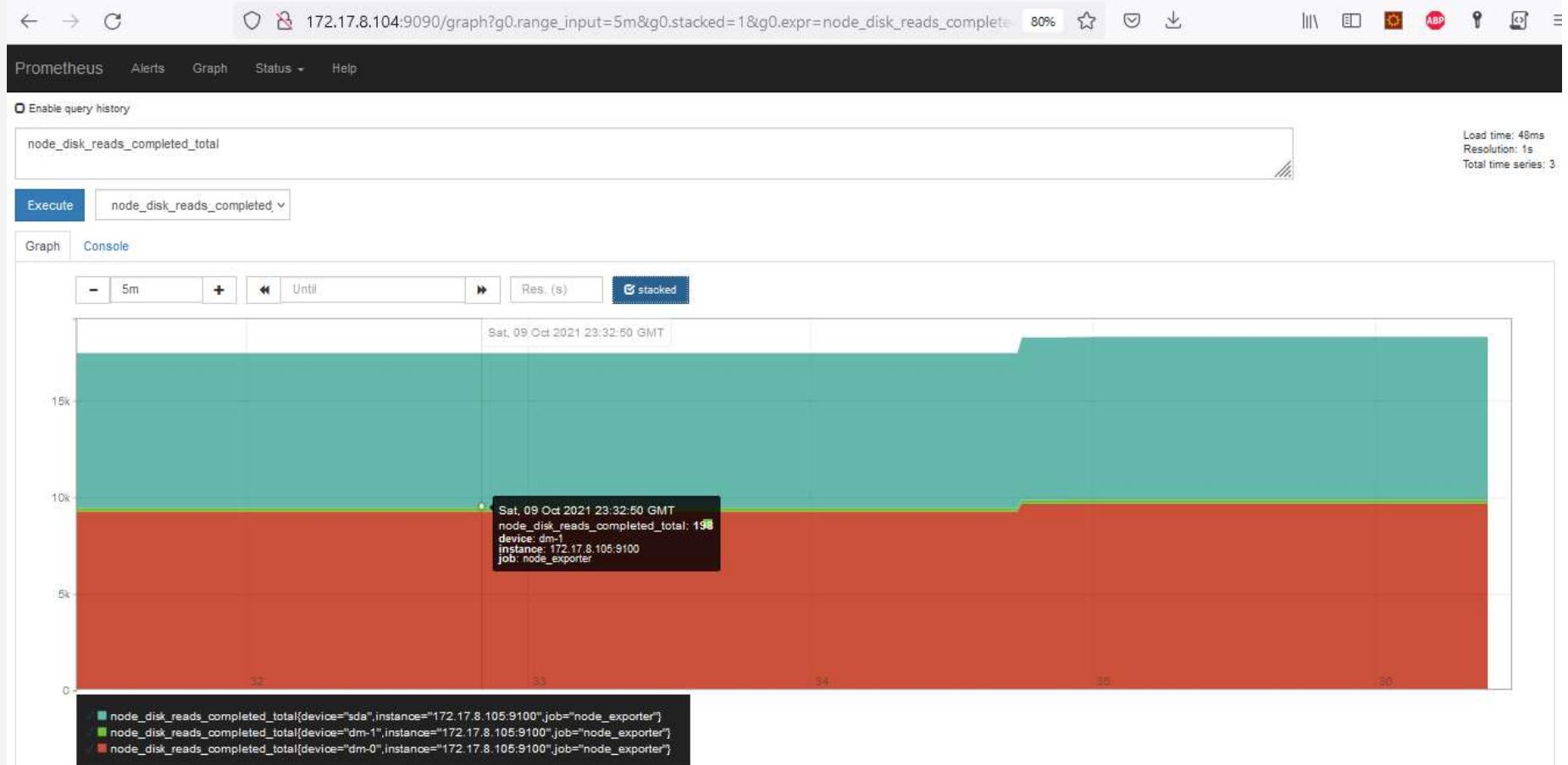
SUMMARY



HISTOGRAM



Prometheus Dashboard



Exporter

- ▶ There are several libraries and servers which help in exporting existing metrics from third-party systems as Prometheus metrics and maintained as part of the official Prometheus GitHub
 - **Databases (e.g., MySQL server exporter-Official)**
 - **Hardware (e.g., Node exporter-Official)**
 - **Trackers and CI (e.g., Jenkins exporter)**
 - **Messaging systems (e.g., Kafka exporter)**
 - **HTTP (e.g., Apache exporter)**
 - ▶ Software exposing Prometheus metrics
 - **Ansible Tower (AWX)**
 - **Kubernetes (direct)**
 - ▶ Other utilities
 - **Java/JVM (EclipseLink metrics collector)**
 - **Node.js (swagger-stats)**

Data Visualization

- ▶ **Prometheus** scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs
- ▶ Its stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts
- ▶ An API consumers like **Grafana** can be used to visualize the collected data
- **Used to query and visualize metrics**
- **Support multiple backend (Prometheus, MySQL, Datadog)**
- **Combine data from different sources**
- **Dashboard visualization fully customizable**
- **Each panel has a wide variety of styling and formatting options**
- **Supports templates and collection of add-ons and pre-built dashboards**

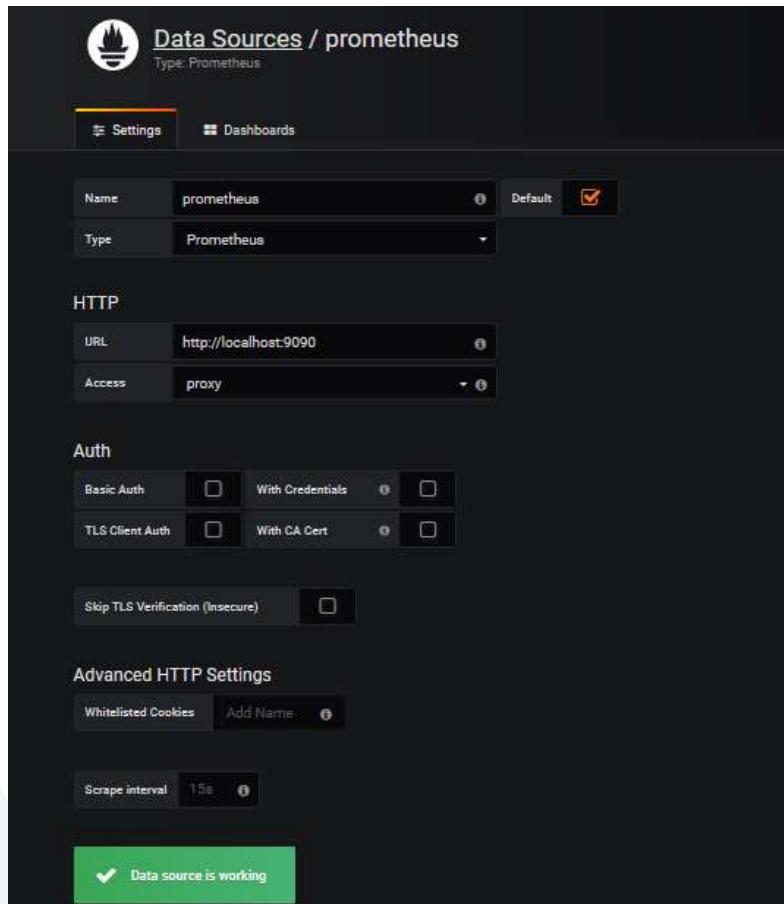


402

Grafana

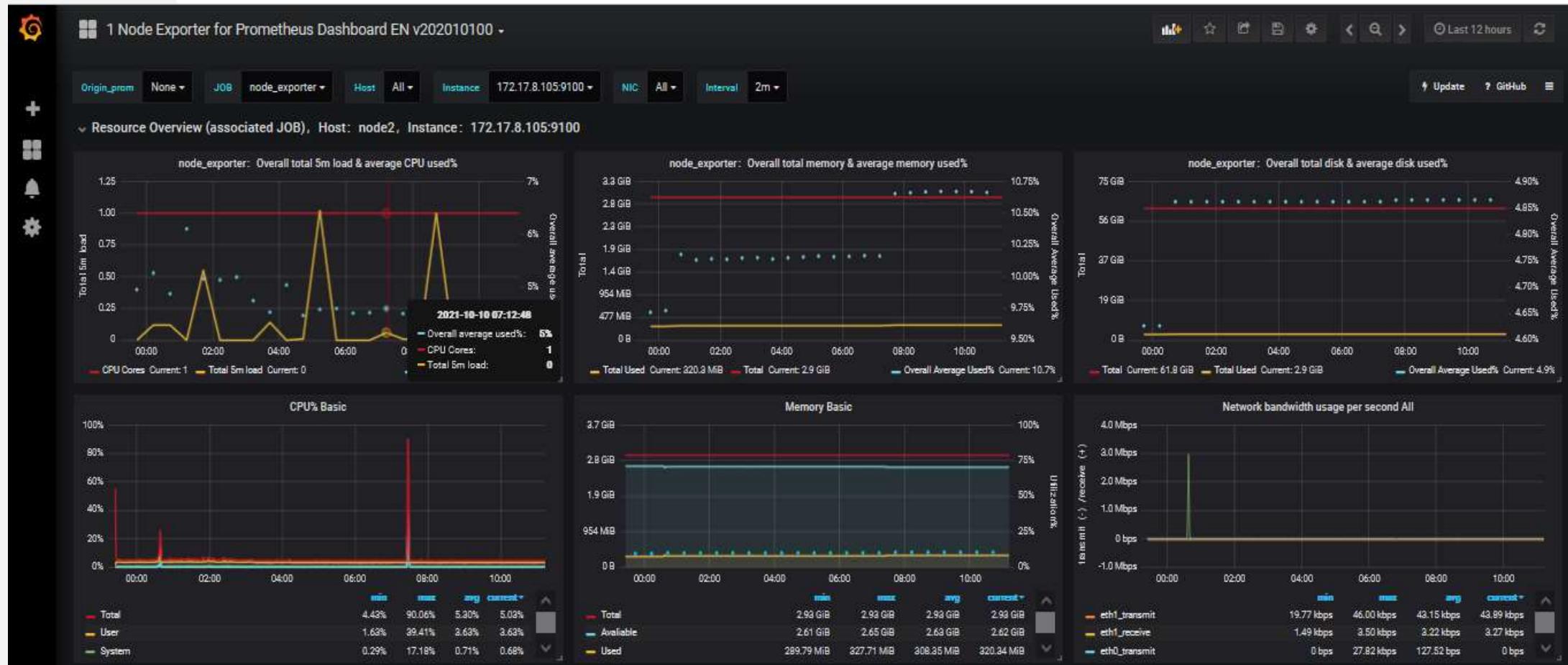
The screenshot shows the Grafana interface. On the left, the login screen displays the Grafana logo and a form with fields for 'admin' and 'password'. A 'Log In' button is present, along with a 'Forgot your password?' link. On the right, the 'Home Dashboard' is visible, featuring a dark header with a navigation menu icon and a 'Home' dropdown. Below the header, there are several interactive buttons: 'Install Grafana' (green), 'Add data source' (green), 'Create your first dashboard' (grey), 'Invite your team' (grey), and 'Install apps & plugins' (grey). The main content area includes sections for 'Starred dashboards' and 'Recently viewed dashboards' (both empty), and a sidebar for 'Installed Apps', 'Installed Panels', and 'Installed Datasources', all of which are currently empty.

Grafana Data Sources



404

Grafana Dashboard



MODULE V-1

Grafana Dashboard



MODULE V-2 : Log Management and Analysis

PLAN

- ▶ LOGS
- ▶ ELASTICSEARCH
- ▶ LOGSTASH
- ▶ KIBANA



Logs

- ▶ Most of applications, containers and Virtual Machines constantly generate information about **numerous events**
- ▶ These events can be anything from **severe errors** to a **simple notice**
- ▶ Collecting and analyzing this log data become challenging in a dynamic architecture or microservice environment
- ▶ Lots of **users, systems** (Routers, firewalls, servers) and **logs** (Netflow, syslogs, access logs, service logs and audit logs)
- ▶ **Elastic stack** (Logstash, Elasticsearch and Kibana) is used as a reference implementation.

Why Logs analysis

- ▶ Monitor all above given issues as well as process operating systems
- ▶ Helps DevOps teams to **gain insights** into their applications that allow them to better identify areas for improvement
- ▶ Helps to map patterns of user behavior
- ▶ Reducing **Customer Churn**
- ▶ Accelerating Releases
- ▶ Optimizing Production Infrastructure Costs

ELK Stack

- ▶ **ELK Stack** is a collection of three (03) open-source products (Elasticsearch, Logstash, and Kibana).
 - **Elasticsearch** : used for storing logs
 - **Logstash** : used for both shipping as well as processing and storing logs
 - **Kibana** : used for visualization of data through a web interface
- ▶ **ELK stack** provides centralized logging in order to identify problems with servers or applications
- ▶ It allow you to search all the logs in a single place
- ▶ Helps to find issues in multiple servers by connecting logs during a specific time frame



ELK Architecture



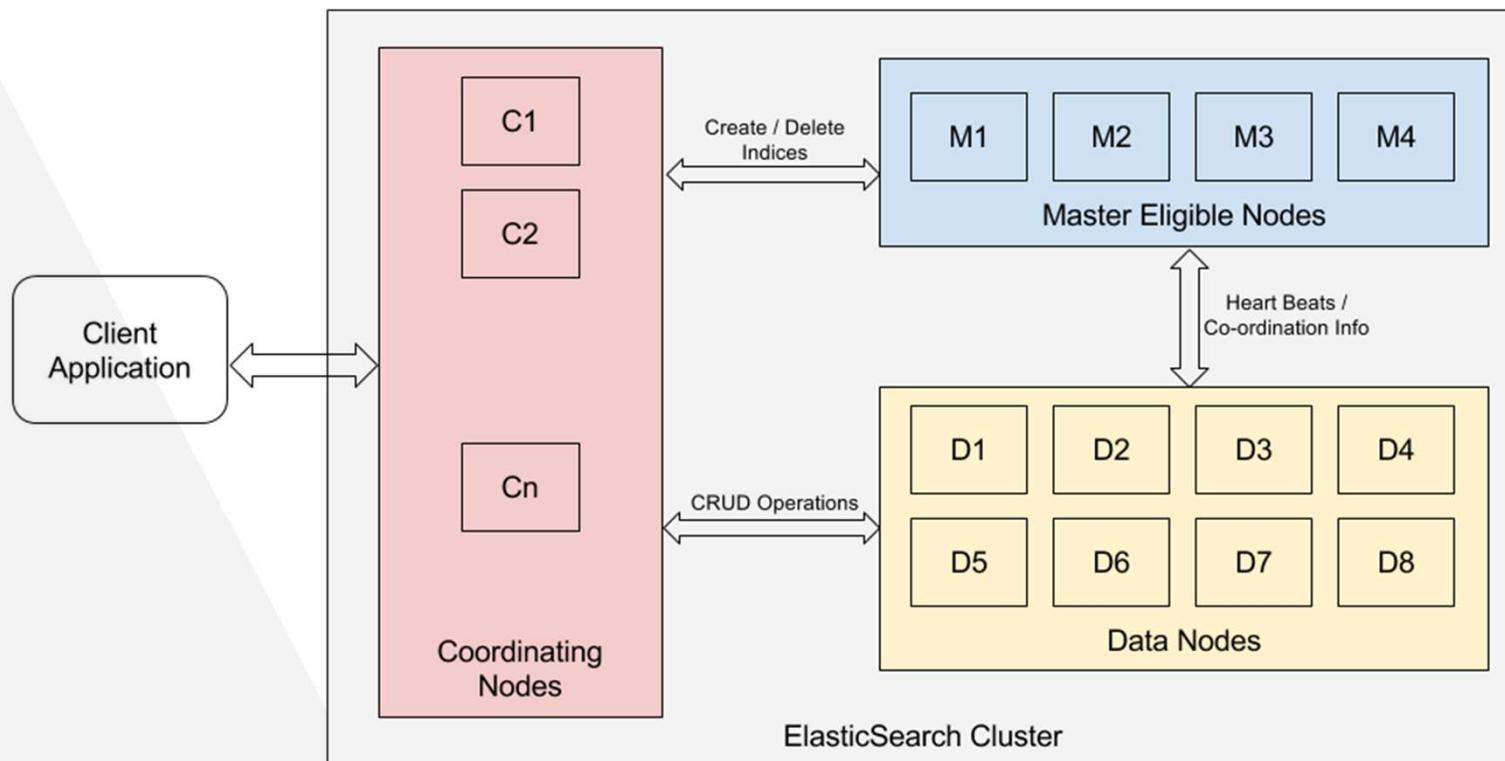
Elasticsearch

Elasticsearch is an open-source, extensively distributable, promptly adaptable, web search tool that is available through a broad and expounds API.

Elasticsearch can control incredibly quick searches that help your information revelation applications

- ▶ **NoSQL database built with RESTful API**
- ▶ **Simplifies data ingest, visualization, and reporting**
- ▶ **It helps in fast and Incisive search against large volumes of data**
- ▶ **Real-time data and real-time analytics**
- ▶ **Wide set of features like scalability, Hight availability and multi-tenant**

Elasticsearch Architecture (1/5)



Elasticsearch Architecture (2/5)

► **Cluster (Elasticsearch cluster)**

One or more servers collectively providing indexing and search capabilities node

► **Node**

Single physical or virtual machine that holds full or part of your data and provides computing power for **indexing** and **searching** your data

A node must accomplish several duties such as :

- Storing the data
- Performing operations on data (indexing, searching, aggregation, etc.)
- Maintaining the health of the cluster

Elasticsearch Architecture (3/5)

Based on the responsibilities, the following are different types of nodes that are supported

- **Data Node**

Node that has storage and compute capability. Participate in the CRUD, search, and aggregate operations

- **Master Node**

Nodes are reserved to perform administrative tasks. It track the availability/failure of the data nodes. They are responsible for creating and deleting the indices

- **Coordinating-Only Node**

Act as a smart **load balancers**. They are exposed to end-user requests. It appropriately redirects the requests between data nodes and master node

Elasticsearch Architecture (4/5)

▶ **Index**

It is a container to store data like a database in the relational databases. An index contains a collection of documents that have similar characteristics or are logically related (e.g customer data product catalog).

▶ **Document**

Document is the piece indexed by Elasticsearch and it is represented in the JSON format

▶ **Mapping types**

Mapping types is needed to create different types in an index and specified during index creation

Elasticsearch Architecture (5/5)

► Shards

Shard is a full-featured subset of an index and help with enabling Elasticsearch to become horizontally scalable (An index can store millions of documents and occupy terabytes of data, this can cause problems with performance, scalability, and maintenance).

► Replication

To ensure fault tolerance and high availability, Elasticsearch provides this feature to replicate the data (Shards can be replicated)

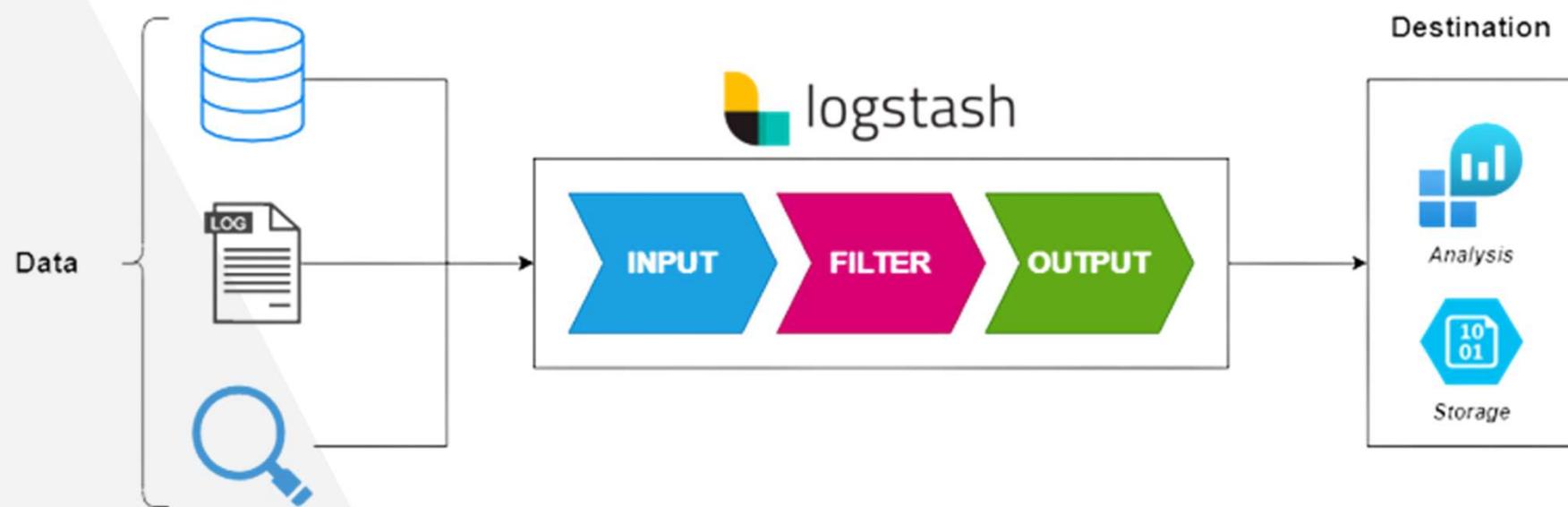
- High Availability : Data can be available through the replica shard even if the node failed
- Performance : search queries will be executed parallelly across the replicas

Logstash

Logstash is a free and open-source, server-side data processing pipeline that can be used to ingest data from multiple sources, transform it, and then send it to further processing or storage

- ▶ Data collection pipeline tool that collect data inputs and feeds into Elasticsearch
- ▶ Gathers all types of data from different sources and makes it available for further use
- ▶ Unify data from disparate sources and normalize the data into a desired destination
- ▶ Consists of three components :
 - **Input** : passing logs to process them into machine understandable format
 - **Filters** : It is a set of conditions to perform a particular action or event
 - **Output** : Decision maker for processed event or log

Logstash Architecture



Logstash Data Inputs

Inputs are the starting point of Logstash configuration. By default, **Logstash** will automatically create a **stdin input** if there are **no inputs** defined.

Some of the more commonly-used inputs are :

- ▶ **File** : Reads from a file on the filesystem, much like the UNIX command tail -DF
- ▶ **Syslog** : Listens on the well-known port 514 for syslog messages and parses.
- ▶ **Redis** : Reads from a redis server, using both redis channels and redis lists
- ▶ **Beats** : Processes events sent by Beats

Logstash Data Inputs Example

► MySQL log file

```
input {  
    file {  
        path => "/var/log/mysql/mysql.log", "/var/log/mysql/mysql-show.log", "/var/log/mysql/mysql-error.log"  
        type => "mysql"  
    }  
}
```

► Syslog

```
input {  
    syslog {  
        port => 514  
        codec => cef  
        syslog_field => "syslog"  
        grok_pattern => "<%{POSINT:priority}>%{SYSLOGTIMESTAMP:timestamp} CUSTOM GROK HERE"  
    }  
}
```

► Redis

```
input {  
    redis {  
        id => "my_plugin_id"  
    }  
}
```

Logstash Filters

Filters are intermediary processing devices in the Logstash pipeline. They can be combined with conditionals to perform an action on an event if it meets certain criteria.

Some useful filters are :

- ▶ **GROK** : Parse unstructured log data into something structured and queryable
- ▶ **MUTATE** : Perform general transformations on event fields. Help to rename, remove, replace, and modify fields in your events
- ▶ **DROP** : Drop everything that gets to the filter
- ▶ **CLONE** : Make a copy of an event, possibly adding or removing fields
- ▶ **GEOIP** : Add information about geographical location of IP addresses (Displays amazing charts in Kibana)

Logstash Filters Example

► DROP

```
filter {
    if [loglevel] == "debug" {
        drop { }
    }
}
```

► GROK (from http request log 55.3.244.1 GET /index.html 15824 0.043)

```
filter {
    grok {
        match => { "message" => "%{IP:client} %{WORD:method} %{URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration}" }
    }
}
```

► MUTATE

```
filter {
    mutate {
        split => ["hostname",",",""]
        add_field => {"shortHostname"=>"%{hostname[0]}"}
    }
}
```

Logstash Outputs

Outputs are the final phase of the logstash pipeline. An event can pass through multiple outputs, but once all output processing is complete, the event has finished its execution

Some useful outputs are :

- ▶ **Elasticsearch** : Store logs in Elasticsearch
- ▶ **File** : Writes events to files on disk
- ▶ **Graphite** : Writes metrics to Graphite
- ▶ **Http** : Sends events to a generic HTTP/HTTPS endpoint
- ▶ **Statsd** : Sends metrics using the Statsd network daemon
- ▶ **S3** : Sends Logstash events to the Amazon Simple Storage Service

Logstash Filters Example

▶ Elasticsearch

```
output {  
    elasticsearch {  
        hosts => "hostname"  
        data_stream => "true"  
    }  
}
```

▶ FILE

```
output {  
    file {  
        path => "C:/devops-book-labs/logstash/bin/log/output.log"  
        codec => line { format => "custom format: %{message}" }  
    }  
}
```

▶ Statsd

```
output {  
    statsd {  
        host => "statsd.example.org"  
        count => {  
            "http.bytes" => "%{bytes}"  
        }  
    }  
}
```

Logstash Complete Example

```
01-apacheserver.conf 481 bytes
```

```
1 input {
2   file {
3     path => "/var/log/httpd/access_log"
4     type => "apache-access"
5     start_position => "beginning"
6   }
7 }
8 filter {
9 if [type] == "apache-access"
10 {
11   grok {
12     match => { "message" => "%{COMBINEDAPACHELOG}" }
13   }
14 }
15 date {
16   match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
17 }
18 }
19 output {
20   elasticsearch {
21     hosts => ["localhost:9200"]
22   }
23   stdout { codec => rubydebug }
24 }
```

Kibana

Kibana is a data visualization and exploration tool used for log and time-series analytics, application monitoring, and operational intelligence use cases

- ▶ Dashboard offers various **interactive diagrams**, geospatial data, and graphs to visualize complex queries
- ▶ It can be used for search, view, and interact with data stored in Elasticsearch directories
- ▶ It helps users to perform advanced data analysis and visualize their data in a variety of tables, charts, and maps.
- ▶ Integrates different methods for performing searches on data

Kibana



ELK

- ▶ **Centralized logging** can be useful when attempting to identify problems with servers or applications
- ▶ **ELK** is a collection of three open-source tools Elasticsearch, Logstash and Kibana.
- ▶ Elasticsearch is a NoSQL database, Logstash is the data collection pipeline tool and Kibana is a data visualization platform
- ▶ **Netflix, LinkedIn, Tripware, Medium** all are using ELK stack for their business
- ▶ ELK works best when logs from various Apps of an enterprise converge into a single ELK instance

References/Modern Software Development

- <https://www.amazon.fr/dp/B0B5PK5P9V/>
- <https://www.lpi.org/blog/2018/01/16/devops-tools-introduction-02-modern-software-development>
- https://www.tutorialspoint.com/sdlc/sdlc_quick_guide.htm
- <https://martinfowler.com/bliki/DevOpsCulture.html>
- https://www.tutorialspoint.com/sdlc/sdlc_quick_guide.htm
- <https://about.gitlab.com/topics/devops/>
- <https://orangematter.solarwinds.com/2022/03/21/what-is-devops/>
- <https://razorops.com/blog/benefits-of-devops/>
- <https://www.guru99.com/agile-vs-devops.html>
- <https://www.manutan.com/blog/fr/lexique/api-definition-et-application-dans-les-achats>
- <https://slideplayer.com/slide/15761190/>
- <https://restfulapi.net/rest-architectural-constraints/>
- <https://reflectoring.io/complete-guide-to-csrf/>
- <https://portswigger.net/web-security/csrf/tokens>
- <https://www.wallarm.com/what/what-is-cross-site-request-forgery>
- <https://portswigger.net/web-security/csrf>
- <https://developer.mozilla.org/fr/docs/Web/HTTP/CORS>
- <https://bunny.net/academy/http/what-are-cross-origin-resource-sharing-cors-headers/>
- <https://www.ionos.fr/digitalguide/sites-internet/developpement-web/cross-origin-resource-sharing/>
- https://www.tutorialspoint.com/software_architecture_design/introduction.htm
- <https://medium.com/@concisesoftware/whats-the-difference-between-software-architecture-and-design-b705c2584631>
- <https://www.slideshare.net/arslantumbin/software-architecture-vs-design>
- <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>
- <https://www.bmc.com/blogs/microservices-architecture/>
- <https://www.techtarget.com/searchapparchitecture/definition/service-oriented-architecture-SOA>
- <https://www.infoworld.com/article/2071889/what-is-service-oriented-architecture.html>
- <https://subscription.packtpub.com/book/application-development/9781789133608/1/ch01lvl1sec04/service-oriented-architecture-soa>
- <https://www.geeksforgeeks.org/service-oriented-architecture/>
- <https://www.geeksforgeeks.org/service-oriented-architecture/>
- <https://microservices.io/>
- <https://smartbear.com/solutions/microservices/>
- <https://anceret-mathieu.fr/2021/07/larchitecture-micro-services/>
- <https://www.datadoghq.com/knowledge-center/serverless-architecture/>
- <https://www.okta.com/identity-101/serverless-computing/>
- <https://rubygarage.org/blog/monolith-soa-microservices-serverless>

References/ Standard Components and Platforms

- <https://www.bridge-global.com/blog/mutable-vs-immutable-infrastructure/>
- <https://k21academy.com/terraform-iac/why-terraform-not-chef-ansible-puppet-cloudformation/>
- <https://www.hpe.com/us/en/what-is/data-storage.html>
- <https://qumulo.com/blog/block-storage-vs-object-storage-vs-file-storage/>
- <https://www.techtarget.com/searchstorage/tip/Object-storage-vs-file-storage-for-cloud-applications>
- <https://www.bigdataframework.org/data-types-structured-vs-unstructured-data/>
- https://www.researchgate.net/figure/CAP-theorem-with-databases-that-choose-CA-CP-and-AP_fig1_282519669
- <https://www.bmc.com/blogs/cap-theorem/>
- <https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/>
- <https://hevodata.com/learn/relational-database-vs-nosql/>
- <https://www.tutorialspoint.com/Types-of-databases>
- <http://talimi.se/database/>
- https://www.researchgate.net/figure/SQL-vs-MongoDB-terms_fig4_340622952
- <http://développeurweb.com/sql-vs-nosql-avantages-et-inconvénients/>
- <https://www.cloudamqp.com/blog/what-is-message-queuing.html>
- <https://blog.devgenius.io/everything-about-distributed-message-queue-ae6597d84b36>
- <https://dachou.github.io/2018/09/28/cloud-service-models.html>
- <http://bigdatariding.blogspot.com/2013/10/cloud-computing-types-of-cloud.html>

References/ Source Code Management

- <https://www.lpi.org/blog/2018/01/30/devops-tools-introduction-04-source-code-management>
- <https://www.geeksforgeeks.org/version-control-systems/>
- <https://medium.com/@vemulasrinivas2505/version-control-systems-74375eb48961>
- <https://medium.com/version-control-system/types-of-version-control-system-766a6b656088>
- <https://devopsbuzz.com/centralized-vs-distributed-version-control-systems/>
- <https://medium.com/@derya.cortuk/version-control-software-comparison-git-mercurial-cvs-svn-21b2a71226e4>
- https://gite.lirmm.fr/common-docs/doc-git/-/wikis/presentation/initiation_git_gitlab.pdf
- <https://www.slideshare.net/KishorKumar/git-46566815>
- <https://slideplayer.fr/slide/15375846/>
- https://www.tutorialspoint.com/git/git_life_cycle.htm
- <https://medium.com/analytics-vidhya/git-most-frequently-used-commands-9df9f200c235>
- <https://git-scm.com/docs/git-revert>
- <https://www.miximum.fr/blog/enfin-comprendre-git/>
- <https://fr.slideserve.com/sakina/git-branching-powerpoint-ppt-presentation>
- <https://blog.developer.atlassian.com/pull-request-merge-strategies-the-great-debate/>

References/ CI/CD

- <https://www.lpi.org/blog/2018/02/06/devops-tools-introduction-05-continuous-delivery>
- <https://www.slideshare.net/stevemac/introduction-to-cicd>
- <https://medium.com/edureka/continuous-integration-615325cfeeac>
- <https://aws.amazon.com/devops/continuous-integration/>
- <https://www.guru99.com/software-testing-introduction-importance.html>
- <https://www.leewayhertz.com/software-testing-process/>
- <https://quodem.com/en/blog/what-is-software-testing-outsourcing/>
- <https://www.leewayhertz.com/software-testing-process/>
- <https://vivadifferences.com/difference-between-functional-and-non-functional-testing/>
- <https://www.gcreddy.com/tag/types-of-test-tools>
- https://aws.amazon.com/devops/continuous-delivery/?nc1=h_ls
- <https://www.spiceworks.com/tech/devops/articles/cicd-vs-devops/>
- <https://1902software.com/blog/automated-deployment/>
- <https://developer.cisco.com/learning/labs/jenkins-lab-01-intro/>
- <https://www.gocd.org/2017/07/25/blue-green-deployments.html>
- <https://www.gocd.org/2017/08/15/canary-releases/>
- <https://www.census.gov/fedcasic/fc2018/ppt/6CVazquez.pdf>
- <https://formation.cloud-gfi-nord.fr/assets/integrationContinue/CI-Jenkins.pptx>
- <https://subscription.packtpub.com/book/application-development/9781784390891/3/ch03lvl1sec20/the-jenkins-user-interface>
- <https://blog.devgenius.io/what-is-jenkins-pipeline-and-jenkinsfile-96f30f3a29c>
- <https://formation.cloud-gfi-nord.fr/assets/integrationContinue/CI-Jenkins.pptx>
- <https://www.edureka.co/blog/jenkins-master-and-slave-architecture-a-complete-guide/>
- <https://www.jenkins.io/doc/book/pipeline/syntax/>

References/ Virtual Machine Deployment

- Practical DevOps Tools, Gilbert Fongan Toussido
<https://www.slideshare.net/NikunjDhameliya1/virtual-machine-69002899>
- <https://techgenix.com/linux-virtual-machine-vm/>
- <https://slideplayer.com/slide/13686844/>
- <https://www.javatpoint.com/what-is-vagrant>
- <https://www.agilequalitymadeeasy.com/post/vagrant-crash-course-development-environments-made-easy>
- <https://medium.com/happy-giraffe/vagrant-box-up-and-go-88f5ff5d09d1>
- <https://mariadb.com/kb/en/creating-a-vagrantfile/>
- <https://slideplayer.com/slide/13686844/>

References/ Cloud Deployment

- <https://www.lpi.org/blog/2018/01/23/devops-tools-introduction-03-cloud-components-and-platforms>
- <https://cloudcomputingtypes.com/>
- https://www.academia.edu/40026388/Cloud_Deployment_Model
- <https://www.spiceworks.com/tech/cloud/articles/what-is-private-cloud-storage/>
- <https://medium.com/@IDMdatasecurity/types-of-cloud-services-b54e5b574f6>
- <https://stackshare.io/stackups/cloud-foundry-vs-openstack>
- <https://www.cloudfoundry.org/>
- <https://cloudinit.readthedocs.io/en/latest/>
- <https://www.c-sharpcorner.com/article/what-is-cloud-init-and-boot-diagnostic-on-azure-vm/>

References/ System Image Creation

-  <https://www.packer.io/intro/use-cases>
- <https://learn.hashicorp.com/tutorials/packer/get-started-install-cli>
- <https://www.talentica.com/blogs/automation-of-ami-creation-using-packer/>
- <https://devopscube.com/packer-tutorial-for-beginners/>
- <https://lzone.de/ch>
- <https://developer.hashicorp.com/packer/docs/commands/buildsheet/pa>

References/Container Usage

- <https://www.lpi.org/blog/2018/02/13/devops-tools-introduction-06-container-basics>
- <https://www.slideteam.net/introduction-to-dockers-and-containers-powerpoint-presentation-slides.html#images-3>
- <https://www.citrix.com/fr-fr/solutions/app-delivery-and-security/what-is-containerization.html>
- <https://github.com/kaan-keskin/introduction-to-docker/blob/main/Introduction.md>
- <https://kaushal28.github.io/Docker-Basics/>
- <https://www.slideteam.net/introduction-to-dockers-and-containers-powerpoint-presentation-slides.html>
- <https://kaushal28.github.io/Docker-Basics/>
- <https://medium.com/@mccode/using-semantic-versioning-for-docker-image-tags-dfde8be06699>
- <https://github.com/kaan-keskin/introduction-to-docker/blob/main/ContainerizingAnApp.md>
- https://kapeli.com/cheat_sheets/Dockerfile.docset/Contents/Resources/Documents/index
- <https://iceburn.medium.com/dockerfile-cheat-sheet-9f52aa4a99b3>
- <https://docs.docker.com/engine/reference/commandline/build/>
- <https://aboullait.me/multi-stage-docker-java/>
- <https://www.padok.fr/en/blog/docker-image-multi-staging>
- <https://github.com/dockersamples/atsea-sample-shop-app/blob/master/app/Dockerfile>
- <https://blog.octo.com/en/docker-registry-first-steps/>
- <https://phoenixnap.com/kb/list-of-docker-commands-cheat-sheet>

References/ Container Deployment and Orchestration

- <https://www.lpi.org/blog/2018/02/20/devops-tools-introduction-07-container-orchestration>
- <https://docs.docker.com/compose/>
- <https://www.baeldung.com/ops/docker-compose>
- <https://k21academy.com/docker-kubernetes/docker-compose/>
- <https://net2.com/how-to-install-docker-compose-on-ubuntu-20-04/>
- <https://github.com/nigelpoulton/counter-app/blob/master/docker-compose.yml>
- <https://faizanbashir.me/practical-introduction-to-docker-compose-d34e79c4c2b6>
- <https://docs.docker.com/compose/compose-file/compose-file-v3/>
- <https://github.com/kaan-keskin/introduction-to-docker/blob/main/ContainerNetworking.md>
- <https://docs.docker.com/engine/tutorials/networkingcontainers/>
- <https://www.section.io/engineering-education/introduction-to-docker-swarm-in-container-orchestration/>
- <https://pt.slideshare.net/FawadKhaliq/microservices-architectures-with-docker-swarm-etcd-kuryr-and-neutron/6>
- <https://www.section.io/engineering-education/introduction-to-docker-swarm-in-container-orchestration/>
- <https://neo4j.com/blog/neo4j-container-orchestration-kubernetes-docker-swarm-mesos/>
- <https://github.com/kaan-keskin/introduction-to-docker/blob/main/DockerSwarm.md>
- <https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/>
- <https://www.slideshare.net/thomasch/docker-distributed-application-bundle-stack-overview>
- <https://buildvirtual.net/how-to-use-docker-stack-to-deploy-docker-containers/>
- <https://www.slideshare.net/rajdeep/introduction-to-kubernetes>
- <https://www.slideteam.net/understanding-kubernetes-architecture-with-diagrams-complete-deck.html>
- <https://kodekloud.com/blog/kubernetes-features-for-beginner/>
- <https://www.waytoeasylearn.com/learn/kubernetes-features/>
- <https://github.com/gopal1409/KubernetesDocker>
- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>
- <https://phoenixnap.com/kb/kubernetes-objects>
- <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>
- <https://iximiu.z.com/en/posts/kubernetes-api-structure-and-terminology/>
- <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>
- <https://blog.ineat-group.com/2019/10/presentation-de-kubernetes/>
- <https://blog.learncodeonline.in/kubernetes-core-concepts-services>
- <https://www.learnitguide.net/2020/05/kubernetes-services-explained-examples.html>
- <https://collabnix.com/3-node-kubernetes-cluster-on-bare-metal-system-in-5-minutes/>
- <https://medium.com/the-programmer/working-with-deployments-roll-back-and-rolling-updates-bb4299488e34>

References/Container Infrastructure

-  <https://www.lpi.org/blog/2018/02/27/devops-tools-introduction-08-container-infrastructure>
- <https://github.com/docker/machine>
- <https://devopscube.com/docker-machine-tutorial-getting-started-guide/>
- https://medium.com/@cnadeau/_/docker-machine-basic-examples-7d1ef640779b
- <https://lucanus cervus-notes.readthedocs.io/en/latest/Virtualization/Docker/Docker%20Machine/>
- <https://docs.ionos.com/docker-machine-driver-1/v/docker-machine-driver/usage/commands>
- <https://docs.docker.com/desktop/>

References/ Ansible

<https://www.lpi.org/blog/2018/03/13/devops-tools-introduction-10-ansible>
<https://datacientest.com/ansible>
<https://www.ansible.com/overview/how-ansible-works>
<https://spacelift.io/blog/ansible-vs-terraform>
https://docs.rockylinux.org/books/learning_ansible/01-basic/
<https://dev.to/rahulku48837211/ansible-architecture-and-setup-2355>
<https://www.fita.in/devops-tutorial/>
<https://www.ovnvirtualization.pro/red-hat-ansible-automation-architecture-and-features/>
<https://www.fita.in/devops-tutorial/>
<https://www.fita.in/devops-tutorial/>
Installation Guide – Ansible Documentation
https://docs.ansible.com/ansible/latest/reference_appendices/general_precedence.html#general-precedence-rules
<https://www.ovnvirtualization.pro/ansible-part-i-basics-and-inventory/>
<https://tekneed.com/creating-and-managing-ansible-configuration-file-linux/>
<https://www.c-sharpcorner.com/article/getting-started-with-ansible-part-3/>
<https://www.ssh.com/academy/ssh/copy-id>
<https://linuxhint.com/use-ssh-copy-id-command/>
https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#intro-inventory
https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html
<https://www.javatpoint.com/ansible-playbooks>
https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html
<https://digitalvarys.com/how-to-write-an-ansible-playbook/>
<https://www.slideshare.net/knoldus/introduction-to-ansible-81369741>
https://www.bogotobogo.com/DevOps/Ansible/Ansible_SettingUp_Webservers_Nginx_Install_Env_Configure_Deploy_App.php
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html
<https://www.ansible.com/overview/how-ansible-works>
https://linuxhint.com/ansible-with_item/
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_loops.html
<https://ericsysmin.com/2019/06/20/how-to-loop-blocks-of-code-in-ansible/>
<https://www.toptechskills.com/ansible-tutorials-courses/ansible-import-variables-tutorial-examples/>
<https://www.dasblinkenlichten.com/ansible-roles-and-variables/>
https://galaxy.ansible.com/docs/contributing/creating_role.html
<https://www.redhat.com/sysadmin/intro-ansible-tower>
<https://developer.arubanetworks.com/aruba-aoscx/docs/ansible-tower-overview>

References/ Other Configuration Management Tools

- <https://www.lpi.org/blog/2018/03/20/devops-tools-introduction-11-other-configuration-management-tools>
- <https://www.devopsschool.com/blog/what-is-chef-tools-benefits-of-using-chef-tools/>
- <https://github.com/Tikam02/DevOps-Guide/blob/master/Infrastructure-provisioning/Chef/chef-commands.md>
- https://www.tutorialspoint.com/chef/chef_overview.htm
- <https://www.digitalocean.com/community/tutorials/how-to-use-roles-and-environments-in-chef-to-control-server-configurations>
- https://docs.chef.io/data_bags/
- <https://www.devopsschool.com/blog/what-is-chef-tools-benefits-of-using-chef-tools/>
- <https://www.linode.com/docs/guides/install-a-chef-server-workstation-on-ubuntu-18-04/>
- <https://nabeelvalley.co.za/docs/automation/manage-node-with-server/>
- <https://severalnines.com/blog/how-automate-daily-devops-database-tasks-chef/>
- <https://docs.chef.io/cookbooks/>
- <https://medium.com/edureka/chef-tutorial-8205607f4564>
- https://docs.chef.io/chef_solo/
- <https://docs.chef.io/workstation/knife/>
- <https://www.digitalocean.com/community/tutorials/how-to-use-roles-and-environments-in-chef-to-control-server-configurations>
- <https://www.techtarget.com/searchitoperations/tip/What-is-the-Puppet-configuration-management-tool-and-how-does-it-work>
- <https://www.javatpoint.com/puppet-architecture>
- <https://github.com/Tikam02/DevOps-Guide/blob/master/Infrastructure-provisioning/Puppet/puppet-commands.md>

References/ IT Operations and Monitoring

-  <https://www.lpi.org/blog/2018/03/27/devops-tools-introduction-12-it-operations-and-monitoring>
- <https://www.crowdstrike.com/cybersecurity-101/observability/devops-monitoring/>
- <https://www.tigera.io/learn/guides/prometheus-monitoring/>
- <https://k21academy.com/docker-kubernetes/prometheus-grafana-monitoring/>
- <https://samirbehara.com/2019/05/30/cloud-native-monitoring-with-prometheus/>
- <https://sensu.io/blog/introduction-to-prometheus-monitoring>
- <https://prometheus.io/docs/instrumenting/exporters/>

References/ Log Management and Analysis

- ▶ <https://www.lpi.org/blog/2018/04/03/devops-tools-introduction-13-log-management-and-analysis>
- ▶ <https://www.mezmo.com/learn-log-management/what-is-log-analysis#:~:text=Log%20analysis%20also%20provides%20significant,map%20patterns%20of%20user%20behavior.>
- ▶ <https://devops.com/business-value-of-log-analysis/>
- ▶ <https://www.guru99.com/elk-stack-tutorial.html>
- ▶ <https://www.yobitel.com/single-post/2019/12/03/elk-kubernetes-stack-in-cloud-native-way-scope-about-retail-industry>
- ▶ <https://www.velotio.com/engineering-blog/elasticsearch-101-fundamentals-core-concepts>
- ▶ <https://www.oreilly.com/library/view/learning-elastic-stack/9781787281868/5a399bbb-a9d6-4338-814c-3058e1fa4e3f.xhtml>
- ▶ <https://www.bmc.com/blogs/logstash-using-data-pipeline/>
- ▶ <https://www.elastic.co/guide/en/logstash/current/input-plugins.html>
- ▶ <https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html>
- ▶ <https://www.elastic.co/guide/en/logstash/current/output-plugins.html>
- ▶ <https://www.elastic.co/guide/en/logstash/current/plugins-outputs-statsd.html>

THANKS!

Any questions?

You can find me at gilbert.toussido@gmail.com