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.


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

I am FONGAN TOUSSIDO Gilbert, Cloud DevOps Engineer with five (05) years of experience across AWS and Azure Cloud platforms. I work with companies in the integration of value-added services in the Cloud while guaranteeing performance efficiency, operational excellence, security, reliability and cost optimization in their infrastructure. I am holder of several certifications in the IT field such like AWS, Azure, LPI DevOps and Hashicorp Terraform. I am also an active member of the LPI (Linux Professional Institute),
PART I: SOFTWARE ENGINEERING
Module 1: Modern Software Development
Module 2: Standard Components and Platforms for Software
Module 3: Source Code Management
Module 4: Continuous Integration and Continuous Delivery

PART II: MACHINE DEPLOYMENT
Module 1: Virtual Machine Deployment
Module 2: Cloud Deployment
Module 3: System Image Creation

PART III: CONTAINER MANAGEMENT
Module 1: Container Usage
Module 2: Container Deployment and Orchestration
Module 3: Container Infrastructure

PART IV: CONFIGURATION MANAGEMENT
Module 1: Ansible
Module 2: Other Configuration Management Tools

PART V: SERVICE OPERATIONS
Module 1: IT Operations and Monitoring
Module 2: Log Management and Analysis
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.
PART I. Software Engineering
MODULE 1 : Modern Software Development
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
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
The **Waterfall model** was the first process Model to be used widely in Software Engineering. The waterfall Model illustrates the software development process in a linear sequential flow. This means that any 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.

The following illustration is a representation of the different phases:
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.
Advantage and disadvantages are:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple and easy to understand. Works well for smaller projects where requirements are very well understood</td>
<td>High amounts of risk an uncertainly</td>
</tr>
<tr>
<td>Highly-disciplined model and Phases are completed one at a time</td>
<td>Not a good model for complex and object-oriented projects</td>
</tr>
<tr>
<td>Clearly defined stages. Process and results are well documented</td>
<td>Difficulty to go back and change the functionnality in testing stage</td>
</tr>
</tbody>
</table>
Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made, and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time.

The following illustration is a representation of the different phases:
Advantage and disadvantages are:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple and easy to understand. Works well for smaller projects where requirements are very well understood</td>
<td>High amounts of risk an uncertainly</td>
</tr>
<tr>
<td>Highly-disciplined model and Phases are completed one at a time</td>
<td>Not a good model for complex and object-oriented projects</td>
</tr>
<tr>
<td>Clearly defined stages. Process and results are well documented</td>
<td>Difficulty to go back and change the functionality in testing stage</td>
</tr>
</tbody>
</table>
The **V-model** is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**. The V-Model is an 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:
Advantage and disadvantages are:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>highly-disciplined model and Phases are completed one at a time</td>
<td>High risk and uncertainty.</td>
</tr>
<tr>
<td>Simple and easy to understand and use.</td>
<td>Not a good model for complex and object-oriented projects</td>
</tr>
<tr>
<td>Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.</td>
<td>Poor model for long and ongoing projects.</td>
</tr>
</tbody>
</table>
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. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas.

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

The following illustration is a representation of the different phases:
Advantages and disadvantages are:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Promotes teamwork and cross training and Suitable for fixed or changing requirements</td>
<td>An overall plan, an agile leader and agile PM practice is a must without which it will not work.</td>
</tr>
<tr>
<td>Delivers early partial working solutions and Minimal rules, documentation easily employed.</td>
<td>Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.</td>
</tr>
<tr>
<td>Enables concurrent development and delivery within an overall planned context.</td>
<td>Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.</td>
</tr>
</tbody>
</table>
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, but these are not enough to realize the benefits of DevOps. Even with the best tools, DevOps is just another buzzword if you don't have the right culture.
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.
The benefits of DevOps are:

- Reliability
- Faster updates
- Reduced time to recover
- Better user experience
- Efficiency
- Fewer failures
- Lower risk
- Higher quality
- Shorter dev cycle
- Faster product delivery
- Stability
- Cost savings
The following show the difference between Agile and DevOps:

<table>
<thead>
<tr>
<th>Parameter/Software Development Technologies</th>
<th>Agile</th>
<th>DevOps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>Manage complex projects</td>
<td>Manage end-to-end engineering processes</td>
</tr>
<tr>
<td><strong>Task</strong></td>
<td>Focuses on constant changes</td>
<td>Focuses on constant testing and delivery</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>Possible within a range of tactical frameworks (sprint, safe and scrum)</td>
<td>Collaboration (doesn't have any commonly accepted framework)</td>
</tr>
<tr>
<td><strong>Feedback</strong></td>
<td>By the customer</td>
<td>By the internal team</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>Gap between customer need and development &amp; testing teams</td>
<td>Gap between development + testing and Ops</td>
</tr>
<tr>
<td><strong>Advantage</strong></td>
<td>Shorter development cycle and improved defect detection</td>
<td>Supports Agile’s release cycle</td>
</tr>
<tr>
<td><strong>Tools used</strong></td>
<td>JIRA, Kanboard</td>
<td>Ansible, Jenkins, Git</td>
</tr>
</tbody>
</table>
The term **API** is an acronym that stands for **Application Programming Interface**.

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. When you specify the menu items you want, the restaurant’s kitchen does the work and provides you with finished pizzas. **You don’t know exactly how the restaurant prepares this food, and you don’t really need to.**
API Concepts and Standards

Check the menu and make the order

User

Waiter

Take the order to the kitchen

Bring the pizza from the kitchen

Kitchen

Deliver the pizza to the table

Request

Mobile Application

Response

API

Server (Back-end System)
**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.
REST defines 6 architectural constraints which make any web service – a truly RESTful API.

- **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
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:

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

- **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. You could allow the return format to 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. Even if it's not an interesting hacking target, a badly behaved client could send thousands of requests every second and crash your server.

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.
A CSRF token is a unique, secret, unpredictable value that is generated by the server-side application and 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.
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

---

**Non-functional requirements**

**Functional requirements**
Modern Software Architecture / Types

Monolithic Architecture

Microservices Architecture

Service Oriented Architecture (SOA)

Serverless 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 Diagram]
Advantage and disadvantages are:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy development and deployment</td>
<td>Slower development speed and not adapted to change for deployment</td>
</tr>
<tr>
<td>High performance task</td>
<td>Not scalability (for individual components) and not reliability</td>
</tr>
<tr>
<td>Simplified and fast testing</td>
<td>Lack of flexibility</td>
</tr>
<tr>
<td>Easy debugging</td>
<td>Barrier to technology adoption</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service reusability</td>
<td>High overhead</td>
</tr>
<tr>
<td>Easy maintenance</td>
<td>High investment</td>
</tr>
<tr>
<td>Platform independent</td>
<td>Complex service management</td>
</tr>
<tr>
<td>Availability, reliability and scalability</td>
<td>Complex maintenance</td>
</tr>
</tbody>
</table>
**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:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increased agility</td>
<td>Increased complexity</td>
</tr>
<tr>
<td>Improved workflows</td>
<td>More expensive</td>
</tr>
<tr>
<td>Decreased the amount of time it takes to improve production</td>
<td>Greater security risks</td>
</tr>
<tr>
<td>Ability to scale horizontally</td>
<td>Different programming languages</td>
</tr>
</tbody>
</table>
Comparison of Architectures

Companies that have evolved from a monolithic approach to microservices:

- Netflix
- Amazon
- Twitter
- PayPal
- eBay
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 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:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to deploy</td>
<td>Security dependent on the service provider</td>
</tr>
<tr>
<td>Enhanced scalability</td>
<td>Privacy &amp; Vendor lock-in: Shared resources in Cloud</td>
</tr>
<tr>
<td>Lower costs</td>
<td>Not for long-term tasks</td>
</tr>
<tr>
<td>Accuracy on function development</td>
<td>Complexity of troubleshooting</td>
</tr>
</tbody>
</table>
Which Architecture?

Which architecture to choose?

**Monolithic**
- Startups, Small Apps
- Small resource base

**Microservices**
- Complex large-scale systems
- Multiple skilled teams

**SOA**
- Enterprise apps with complex operations

**Serverless**
- Client-heavy apps
- Fast-growing and rapidly changing apps
- High-latency background tasks
MODULE 2 : Standard Components and Platforms for Software
PLAN

INFRASCTURE:
- 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:

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

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

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

- **Structured Data** is data that adheres to a pre-defined data model and is therefore straightforward to analyze. It's conforming to a tabular format with relationship between the different rows and columns.

- **Unstructured Data** is information that is not organized in a pre-defined manner. Unstructured information is typically text-heavy, but may contain data such as dates, numbers, and facts as well.
Storage Use Case

- Native cloud applications
- Big data analysis
- Internet of Things
- Storage and distribution of rich media content
- Backup and archiving
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 reads receive the most recent write or an error.
- **Availability**: All reads contain data, but it might not be the most recent.
- **Partition tolerance**: The system continues to operate despite network failures (i.e., dropped partitions, slow network connections, or unavailable network connections between nodes.)
**ACID** describe the set of properties of database transactions that guarantee data integrity despite errors, system failures, power failures, or other issues.
There are following types of databases:

- **Relational Database**: These databases are categorized by a set of tables where data gets fit into a predefined category. The table consists of rows and columns where the column has an entry for data for a specific category and rows contains instance for that data defined according to the category. The **Structured Query Language (SQL)** is the standard user and application program interface for a relational database.

- **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.
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
- Example: MySQL (MariaDB), Oracle, Postgres, IBM DB, ...
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
Database/Difference between SQL and NoSQL
<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>Standardized schema</td>
<td>Hardware</td>
</tr>
<tr>
<td></td>
<td>Large user community</td>
<td>Data normalization</td>
</tr>
<tr>
<td></td>
<td>No code required</td>
<td>Rigidity</td>
</tr>
<tr>
<td></td>
<td>ACID compliance</td>
<td>Resource-intensive scaling</td>
</tr>
<tr>
<td>NoSQL</td>
<td>Continuous availability</td>
<td>No standardized language</td>
</tr>
<tr>
<td></td>
<td>Query speed</td>
<td>Smaller user community</td>
</tr>
<tr>
<td></td>
<td>Agility</td>
<td>Inefficiency with complex queries</td>
</tr>
<tr>
<td></td>
<td>Cost</td>
<td>Data retrieval inconsistency</td>
</tr>
</tbody>
</table>
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 sends it to queue which stores the message if consumer is busy.
- Consumer retrieves the message from the queue and starts processing it.
- Queue temporarily locks the message to prevent it from being read by another consumer.
- Consumer deletes the message from the queue after it completes the message processing.
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
- On-demand service
- Measurable and billable service
- Resource pooling
- 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

- **Traditional IT**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking

- **Infrastructure (as a Service)**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking

- **Platform (as a Service)**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking

- **Software (as a Service)**
  - Applications
  - Data
  - Runtime
  - Middleware
  - Operating System
  - Virtualization
  - Servers
  - Storage
  - Networking
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

- **Private**: Used for a single organization; can be internally or externally hosted.
- **Community**: Shared by several organizations; typically externally hosted, but may be can be internally hosted by one of the organizations.
- **Hybrid**: Composition of two or more clouds (private, community or public) that remain unique entities but are bound together, offering the benefits of multiple deployment models; is internally & externally hosted.
- **Public**: Provisioned for open use for the public by a particular organization who also hosts the service.
Cloud Computing/Stakeholders

- AWS
- Azure
- Google Cloud Platform
- Oracle Cloud
- IBM Cloud
- Alibaba Cloud
MODULE 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 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.
In a **Distributed system** every user has a local copy of the repository in addition to the central repository on the server side.
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

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Super-fast and efficient performance, cross-platform</td>
<td>Complex and bigger history log difficult to understand</td>
</tr>
<tr>
<td>Code changes easy and clearly tracked</td>
<td>Does not support keyword expansion</td>
</tr>
<tr>
<td>Easy maintainable and robust</td>
<td>Does not support timestamp preservation</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name</td>
<td><code>$ git config --global user.name &quot;GilbertFongan&quot;</code></td>
</tr>
<tr>
<td>User email</td>
<td><code>$ git config --global user.email gilbert.fongan@yourdomain.com</code></td>
</tr>
<tr>
<td>Notepad++ as Default editor</td>
<td><code>$ git config --global core.editor &quot;notepad++.exe -multiInst -nosession&quot;</code></td>
</tr>
<tr>
<td>List global configuration</td>
<td><code>$ git config --global --list</code></td>
</tr>
</tbody>
</table>
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 add
- git commit
- git push
- git fetch
- git pull
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

1. **Repository**
   - **Clone operation**
     - **Working copy**
       - Edit, Add, Move files
       - **Modify working copy**
         - **Update operation**
           - Status and Diff operation
             - **Review changes**
               - Commit and Push operation
                 - **Commit changes**
                   - Amend and Push operation
                     - **Fix mistakes**
Git Workflow

- **Initialize**
  - `git init`
  - `git clone <url of remote repository>`

- **Update**
  - `git pull`
  - `git fetch`

- **Changes**
  - `git add`
  - `git commit -a`
  - `git commit`
  - `git push`

- **Revert**
  - `git checkout head`
  - `git checkout`

- **Diff**
  - `git diff`
### Git command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>git init</strong></td>
<td>Convert an existing unversioned project workspace to git repository or to create a new empty git repository. &quot;git&quot; subdirectory will be created</td>
</tr>
<tr>
<td><strong>git clone</strong></td>
<td>Download an existing git repository to your local computer. <code>git clone -b branch_name &lt;git url&gt;</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.</td>
</tr>
<tr>
<td><strong>git status</strong></td>
<td>Current branch</td>
</tr>
<tr>
<td><strong>git add</strong></td>
<td>Add changes in the workspace to the staging area. <code>git add &lt;file-name&gt;</code> or <code>git add .</code> to add all files</td>
</tr>
<tr>
<td><strong>git commit</strong></td>
<td>Add changes in the staging area to the local Git repository. <code>git commit -m 'commit message'</code></td>
</tr>
<tr>
<td><strong>git pull</strong></td>
<td>Update local git repository from the corresponding remote git repository. <code>git pull &lt;remote&gt; &lt;local&gt;</code>: Local git repository ← Remote git repository</td>
</tr>
<tr>
<td><strong>git push</strong></td>
<td>Add changes in the local git repository to the remote repository. <code>git push &lt;remote&gt; &lt;local&gt;</code>: Local git repository → Remote git repository</td>
</tr>
<tr>
<td><strong>git branch</strong></td>
<td>List all local branches. <code>git branch -a</code>: List all remote branches as well</td>
</tr>
<tr>
<td><strong>git checkout</strong></td>
<td>Navigate between different branches. <code>git checkout &lt;branch&gt;</code></td>
</tr>
<tr>
<td><strong>git merge</strong></td>
<td>Integrate changes from multiple branches into one. <code>git merge &lt;branch&gt;</code></td>
</tr>
</tbody>
</table>
## Git command

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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.  
`git remote -v`: List all remote connections  
`git remote add <name> <url>`: Create a new remote connection  
`git remote rm <name>`: Delete a connection to a remote repository  
`git remote rename <old name> <new name>`: 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.  
`git fetch <remote> <local>`  
`git checkout <remote>/<local>`: To view the change  
`git fetch vs git pull: git pull = git fetch + git merge` |
| **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.  
`git reset HEAD~ --hard` 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

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

2. Example commit, 3 files
   - Commit history shown with SHA-1 hashes.
   - Each commit links to the previous commit.

3. After 3 commits
   - Snapshots A, B, and C are updated with new commits.

4. Default branch is master
   - Master branch moves forward automatically when you commit.
Git branching

1. Add a branch
   - 98ca9
   - 34ac2
   - master
   - f30ab
   - testing

   **cmd:** git branch testing

   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
   - 98ca9
   - 34ac2
   - f30ab
   - testing
   - master
   - HEAD

   **HEAD = pointer to current branch**

3. Use checkout to change
   - 98ca9
   - 34ac2
   - f30ab
   - testing
   - master
   - HEAD

   **cmd:** git checkout testing

4. Make changes on that branch
   - 98ca9
   - 34ac2
   - f30ab
   - c2b9e
   - testing
   - master

   **edit test.rb (e.g., vim test.rb)**
   **cmd:** git commit -a -m 'made a change'
Git branching

1. **Switch back to master**

```
<table>
<thead>
<tr>
<th>HEAD</th>
<th>master</th>
</tr>
</thead>
<tbody>
<tr>
<td>86dc3</td>
<td>master</td>
</tr>
<tr>
<td>31ac0</td>
<td>5f9ab</td>
</tr>
<tr>
<td>52be0</td>
<td>testing</td>
</tr>
</tbody>
</table>
```

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

So edit 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**

```
<table>
<thead>
<tr>
<th>HEAD</th>
<th>master</th>
</tr>
</thead>
<tbody>
<tr>
<td>86dc3</td>
<td>master</td>
</tr>
<tr>
<td>31ac0</td>
<td>5f9ab</td>
</tr>
<tr>
<td>52be0</td>
<td>testing</td>
</tr>
</tbody>
</table>
```

edit test.rb again

```
cmd: git commit -a -m 'made other changes'
```
Git basic merging

1. A branch/merge example
2. Update branch
3. Need to switch to an urgent fix
4. Now merge hotfix and master

Start with master branch

cmd: `git checkout -b iss53`
NOTE: This creates branch and switches to it

This will be a “fast forward” merge – because branch was directly upstream, git just moves pointer forward.
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
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 the feature branch.
- **Rebase** add all the changes in the 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 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 the latest version of the code from the central repository.
Solution for problems faced in Traditional Integration

Step 01: Split the entire chunk of codes into segments

Step 02: Keep small segments of manageable code

Step 03: Integrate the segmented code, multiples times a day

Step 04: Adopt a continuous integration methodology to coordinate with your team
Benefits of CI/CD

**Reduction of delivery risk**
- Reduced chance of humans not following the process
- Reduced chance of miscommunication on executing the change

**Better visibility on change**
- Systems and tools are version controlled
- Diff the system states with confidence

**Increased efficiency and delivery options**
- Enables to deliver things with reduced effort
- Deploy changes more frequently
- Getting feedback faster
- Leads to smaller batch sizes

**The process is known**
- All the tests pass
- All the steps in deployment
- What is stopping us from releasing

**Opens up more avenues for review**
- and increased audit compliance

**Enhanced learning from failure**
- Test to cover issue or failure
- Execute test every time
- Decreases risk of this issue occurring again
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

1. Reduced Integration Risk
2. Higher Code Quality
3. The Code in Version Control Works
4. Reduced Friction between Team Members
5. The Quality of Life Improvement for Testers
6. Less Deployment Time
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.
There are mainly three approaches to Software Testing:

**Approaches to Software Testing**

- **White Box Testing**
- **Black Box Testing**
- **Grey Box Testing**

**Functional Testing**

**Non-Functional Testing**
## Software Testing / Approaches

<table>
<thead>
<tr>
<th>Testing</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| **White Box Testing (Clear box testing/Glass box testing)** | - Performed at the initial stages  
- More in depth  
- Find hidden defects  
- Helps in code optimization | - Complicated  
- Requires highly skilled resources  
- Tools may not be readily available |
| **Black Box Testing (Behavioral testing)** | - Exposes inconsistencies in specifications  
- No need to understand programming | - Test may be tough to design  
- Many bugs can go undetected |
| **Grey Box Testing (Mixture of white-box and black-box)** | - Develop more intelligent tests  
- Clear goals while testing  
Overall quality of the software is enhanced | - Complicated  
- Hard to detect bugs |
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

There are mainly four 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.
Types of Software Testing

- Manual
  - White Box Testing
    - Functional
      - Unit Testing
      - Integration
      - System Testing
      - User Acceptance Testing
  - Black Box Testing
    - Functional
    - Non-Functional
      - Compatibility Testing
      - Performance Testing
      - Usability Testing
Today, many software testing tools are of great importance, especially for automation testing.
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
Continuous delivery lets your team automatically build, test, and prepare code changes for release to production so that your software delivery is more efficient and rapid.

Improve Developer Productivity
These practices help your team be more productive by freeing developers from manual tasks and encouraging behaviors that help reduce the number of errors and bugs deployed to customers.

Find and Address Bugs Quicker
Your team can discover and address bugs earlier before they grow into larger problems later with more frequent and comprehensive testing. Continuous delivery lets you more easily perform additional types of tests on your code because the entire process has been automated.

Deliver Updates Faster
Continuous delivery helps your team deliver updates to customers faster and more frequently. When continuous delivery is implemented properly, you will always have a deployment-ready build artifact that has passed through a standardized test process.
Continuous Deployment:

- Strategy for software releases wherein any code commit that passes the **automated testing** phase is automatically released into the **production environment**, making changes that are visible to the software's users. 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** and thorough testing, and when they apply sophisticated, real-time monitoring in production to discover any issues with new releases.
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

01. Version Control
   Maintains different versions of the code
   Source Code Management

02. Continuous Integration
   Compile, validate, Code Review, Unit Testing, Integration Testing
   Continuous Build

03. Continuous Delivery
   Deploying the build application to test servers, Performing UAT
   Continuous Testing

04. Continuous Deployment
   Deploying the tested application on the prod server for release.
   Configuration Management and Containerization

Continuous Monitoring
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.
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
- GitHub
- GitLab
- circleci
- Travis CI
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

**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.
Jenkins / Building stages

- Getting the sources
  - Which SCM?
- Build triggers
  - Remote triggers (hook github, gitlab)
  - Following another Build
  - Periodically
- Build Environment
- Post-Build actions
- Script for the Build
  - Environment variables
  - Sonar configuration
  - Workspace cleanup rules
- Publication of reports
- Send notifications (Email...)
- Publication of javadoc
- Maven, gradle, ant scripts
- Shell scripts
Jenkins / User interface

- Configuration Panel
- Header
- Job Table
- Build Queue
- Executor Status Panel
Jenkins workflow
Jenkins / Getting the sources

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

- Possibility to add behaviors
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

Projet tp2-persistance

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
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

Give a name for the Node, Choose the Permanent Agent option and click on OK
Jenkins / Setup Master and Slaves

Configure
Build History
Load Statistics
Script Console
Log
Monitor Information
Disconnect

Build Executor Status
1. Idle
2. Idle

# of executors: 2
Remote root directory: C:\Users\Admin\Phrases\Desktop\slave_srv
Labels:
slave1
Usage:
Launch agent by connecting it to the master
Custom Workspace path:
C:\Users\Admin\Phrases\Desktop\slave_srv
Internal data directory:
remoting
Fail if workspace is missing

Availability:
Keep this agent online as much as possible
Mode Properties:
Disable network isolation on this node

Jenkins

Nodes

Build Queue:
No builds in the queue

Build Executor Status
1. Idle
2. Idle
Slave1
1. Idle
2. Idle

Online
Nodes

Back to Dashboard
Manage Jenkins
New Node
Configure Clouds
Node Monitoring

<table>
<thead>
<tr>
<th>Name</th>
<th>Architecture</th>
<th>Check Difference</th>
<th>Free Disk Space</th>
<th>Free Temp Space</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>Windows 11 (x64)</td>
<td>in sync</td>
<td>385.77 GB</td>
<td>4.77 GB</td>
<td>360.77 GB</td>
</tr>
<tr>
<td>Slave1</td>
<td>Windows 11 (x64)</td>
<td>in sync</td>
<td>335.77 GB</td>
<td>4.77 GB</td>
<td>305.77 GB</td>
</tr>
</tbody>
</table>

Data collected: 8.6 sec 8.5 sec 8.5 sec 7.6 sec 8.5 sec 8.5 sec

Network status
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.
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**

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

```groovy
tools {
    maven "Maven 3.3.9"
    jdk "Oracle JDK 8u40"
}
```
The **Environment** section.

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

**Example**

```groovy
environment {
    FOO = "bar"
    BAZ = "faz"
}
```
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.
Jenkins Declarative Pipeline

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

```groovy
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'
    }
}
```
pipeline {
  tools {
    maven 'Maven 3.3.9'
    jdk 'oracle JDK 8u40'
  }
  // run on any executor
  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." )
    }
  }
}

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 1 : Virtual Machine deployment
PLAN

VIRTUAL MACHINE:

VAGRANT:
- Features
- Architecture
- Workflow

VAGRANTFILE:
- Configure
- Options
- Providers
- Provisioners
- Boxes

VAGRANT COMMAND
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

(a)

(b)

processes

programming interface

kernel

hardware

kernel

VM1

VM2

VM3

virtual-machine implementation

hardware
Advantage and disadvantages are:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Familiar Interfaces</td>
<td>Difficulty in direct access to hardware</td>
</tr>
<tr>
<td>Isolation of OS and Reduction of cost</td>
<td>RAM degradation with the creation of new VMs</td>
</tr>
<tr>
<td>High Availability and Scalability</td>
<td>Disk Space degradation with the creation of new VMs</td>
</tr>
<tr>
<td>Backup with Fast Recovery</td>
<td>Less efficient than physical Machine</td>
</tr>
</tbody>
</table>
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** 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** 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

```ruby
# 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]
  end
  config.vm.provision :shell, path: "bootstrap.sh"
end
```
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
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.
  ```ruby
  config.vm.network "forward_port", guest:80, host:8080
  ```

- **Static IP**: Assign a private IP address to the Virtual Machine
  ```ruby
  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 (Local Area Network)
  ```ruby
  config.vm.network "public_network", ip: "10.0.0.1"
  ```
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

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "geerlingguy/centos7"
  config.vm.hostname = "nodejs.test"
  config.vm.network :private_network, ip: "192.168.55.55"
  config.ssh.insert_key = false
  config.vm.synchronized_paths "./", "~/vagrant", disabled: true

  config.vm.provider :virtualbox do |v|
    v.memory = 512
  end

  # Ansible provisioner.
  config.vm.provision :ansible do |ansible|
    ansible.playbook = "provisioning/playbook.yml"
  end
end
```
Vagrant provisioners

- 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:

```ruby
# Shell provisioning example
Vagrant.configure("2") do |config|
  #...other configuration
  config.vm.provision :shell do |s|
    s.inline = "echo hello Pongan"
    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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vagrant box list</code></td>
<td>See a list of all installed boxes on your computer</td>
</tr>
<tr>
<td><code>vagrant box add &lt;name&gt;&lt;url&gt;</code></td>
<td>Download a box image to your computer</td>
</tr>
<tr>
<td><code>vagrant box outdated</code></td>
<td>Check for updates vagrant box update</td>
</tr>
<tr>
<td><code>vagrant boxes remove &lt;name&gt;</code></td>
<td>Deletes a box from the machine</td>
</tr>
<tr>
<td><code>Vagrant package</code></td>
<td>Packages a running VirtualBox environment in a reusable box</td>
</tr>
</tbody>
</table>
Vagrant command (1/3)

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vagrant init - m</code></td>
<td>Create a minimal Vagrantfile (no comments or helpers)</td>
</tr>
<tr>
<td><code>vagrant init - f</code></td>
<td>Create a new Vagrantfile, overwriting the one at the current path</td>
</tr>
<tr>
<td><code>vagrant init -box-version</code></td>
<td>Create a Vagrantfile, locking the box to a version constraint</td>
</tr>
<tr>
<td><code>Vagrant init &lt;boxpath&gt;</code></td>
<td>Initialize Vagrant with a specific box. To find a box, go to the public Vagrant box catalog. For example, <code>vagrant init ubuntu/trusty64</code></td>
</tr>
</tbody>
</table>

Starting a VM

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vagrant up</code></td>
<td>Starts vagrant environment (also provisions only on the FIRST vagrant up command)</td>
</tr>
<tr>
<td><code>vagrant resume</code></td>
<td>Resume a suspended machine (vagrant up works just fine for this as well)</td>
</tr>
<tr>
<td><code>vagrant provision</code></td>
<td>Forces re-provisioning of the vagrant machine</td>
</tr>
<tr>
<td><strong>Vagrant reload</strong></td>
<td>Restarts vagrant machine, loads new Vagrantfile configuration</td>
</tr>
<tr>
<td><strong>Vagrant reload --provision</strong></td>
<td>Restart the virtual machine and force provisioning</td>
</tr>
</tbody>
</table>
## Vagrant command (2/3)

### Getting into a VM

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vagrant ssh</code></td>
<td>Connects to machine via SSH</td>
</tr>
<tr>
<td><code>vagrant ssh &lt;boxname&gt;</code></td>
<td>If you give your box a name in your Vagrantfile, you can ssh into it with. Boxname works from any directory</td>
</tr>
</tbody>
</table>

### Stopping a VM

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vagrant halt</code></td>
<td>Stops the Vagrant machine</td>
</tr>
<tr>
<td><code>vagrant suspend</code></td>
<td>Suspends a Virtual Machine (remembers state)</td>
</tr>
</tbody>
</table>

### Saving Progress

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Vagrant snapshot save [options][vm-name] &lt;name&gt;</code></td>
<td>Allows us to save the VM so that we can roll back at a later time.</td>
</tr>
</tbody>
</table>
### Other tips

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vagrant -v</code></td>
<td>Get vagrant version</td>
</tr>
<tr>
<td><code>vagrant status</code></td>
<td>Outputs status of the Vagrant machine</td>
</tr>
<tr>
<td><code>vagrant global-status</code></td>
<td>Outputs status of all vagrant machines</td>
</tr>
<tr>
<td><code>vagrant global-status --prune</code></td>
<td>Outputs status of all vagrant machines, but prunes invalid entries</td>
</tr>
<tr>
<td><code>Vagrant provision --debug</code></td>
<td>Use the debug flag to increase the verbosity of the output</td>
</tr>
<tr>
<td><code>Vagrant push</code></td>
<td>Vagrant can be configured to deploy code on remote central registry</td>
</tr>
<tr>
<td>`Vagrant up --provision</td>
<td>tee provision.log`</td>
</tr>
</tbody>
</table>
MODULE 2 : Cloud deployment
CLOUD DEPLOYMENT MODEL:
- Private
- Public
- Hybrid
- Community

CLOUD SERVICE MODEL:
- Cloud Foundry
- Openshift
- Openstack

CLOUD INIT:
- Syntax
- Example
**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,
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 firewall.

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

Boundary perimeter

Databases
Legacy systems
Hardware
Applications

New & third party users
Access control

Owner’s datacenter

Users who access the cloud from within the perimeter

Public users
Blocked access
Cloud Service Model

Cloud Service Models

- **SaaS**
  - End Users
  - Application Developers
  - Infrastructure & Network Architects

- **PaaS**
  - OS & Application Stack
  - Server Storage Network

- **IaaS**
  - Packaged Software
  - OS & Application Stack
  - Server Storage Network
**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

**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

**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 CLoud 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
Run 'apt-get upgrade' on first boot

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

Enable `byobu` by default for all system users

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

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

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

Run a few commands on first boot

```yaml
#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"

```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 log of 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 exe my-testinit --sudo cat /var/log/cloud-init-output.log
```
Cloud-init Example on Azure VM

1. Advanced

2. Custom data and cloud init
   - Pass a cloud-init script, configuration file, or other data into the virtual machine while it is being provisioned. The data will be saved on the VM in a known location. Learn more about custom data for VMs.

   Custom data:
   - `package_upgrade: true`
   - `packages: - nginx`

   Custom data on the selected image will be processed by cloud-init. Learn more about custom data for VMs.

User data
- Pass a script, configuration file, or other data that will be accessible to your applications throughout the lifetime of the virtual machine. Don’t use user data for storing your secrets or passwords. Learn more about user data for VMs.

Enable user data

Review + create

< Previous Next : Tags >
MODULE 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 platforms are:

- Amazon EC2
- CloudStack
- DigitalOcean
- OpenStack
- VMware
- 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
  ```
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

```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.
Pack Provisio

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

```json
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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packer init</td>
<td>Install required plugins</td>
</tr>
<tr>
<td>Packer plugins required</td>
<td>List plugins that will be installed by “packer init”</td>
</tr>
<tr>
<td>Packer plugins installed</td>
<td>List installed plugins</td>
</tr>
<tr>
<td>Packer build</td>
<td>Build image from template. Takes a template and runs all the builds within it in order to generate a set of artifacts. Use –force to forces a builder to run</td>
</tr>
<tr>
<td>Packer fmt</td>
<td>Format HCL2 configuration files to a canonical format and style</td>
</tr>
<tr>
<td>Packer validate</td>
<td>Check that a template is valid</td>
</tr>
<tr>
<td>Packer inspect</td>
<td>See components of a template</td>
</tr>
<tr>
<td>Packer hcl2_upgrade</td>
<td>Convert JSON to HCL2</td>
</tr>
</tbody>
</table>
PART III.
Container Management
...
MODULE 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 is a software that runs on Linux and Windows environments. It creates, manages, 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)"

A 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

- Develop application and its supporting components using containers
- Test the application
- Docker provides a platform for the user to:
  - Deploy the application into production environment, as a container or an orchestrated service

Docker Properties

- Provides fast delivery of the applications
- Is deployable and scalable
- Has high density and runs more workloads
- Aids in quick deployment for easy management
Benefits of Docker

- Application isolation and security management
- Multitenancy
- Simplified configuration
- Rapid deployment
- Server consolidation
- Increased productivity
Docker Architecture
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.

- **Namespaces**: Kernel namespaces are at the very heart of containers. Multiple isolated operating systems.
- **Cgroups**: If namespaces are about isolation, control groups (cgroups) are about setting limits. Cgroups let us set limits so that a single container cannot use all of the CPU, RAM, or Storage I/O of the host.
- **Capabilities**: It's a bad idea to run containers as root. Root is all-powerfull and therefore very dangerous.
## Containers Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Docker container run --it ubuntu /bin/bash</code></td>
<td>Start an Ubuntu container in the foreground, and tell it to run the Bash shell</td>
</tr>
<tr>
<td><code>[Ctrl + PQ]</code></td>
<td>Detach your shell from the terminal of a container and leave the container running (UP) in the background</td>
</tr>
<tr>
<td><code>Docker container ls</code></td>
<td>Lists all containers in the running (UP) state. -a flag you will also see containers in the stopped (Exited) state</td>
</tr>
<tr>
<td><code>Docker container exec --it &lt;container-name or container-id&gt; bash</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>Docker container stop &lt;container-name or container-id&gt;</code></td>
<td>Stop a running container and put it in the Exited(0) state</td>
</tr>
<tr>
<td><code>Docker container start &lt;container-name or container-id&gt;</code></td>
<td>Restart a stopped (Exited) container</td>
</tr>
<tr>
<td><code>Docker container rm &lt;container-name or container-id&gt;</code></td>
<td>Delete a stopped container</td>
</tr>
<tr>
<td><code>Docker container inspect &lt;container-name or container-id&gt;</code></td>
<td>Show detailed configuration and runtime information about a container</td>
</tr>
</tbody>
</table>
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](https://hub.docker.com)).
- The Docker client is opinionated and defaults to using Docker Hub.
- Image registries contain multiple image repositories.
Docker Hub also has the concept of **official repositories** and **unofficial repositories**

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

- **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
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

```bash
$ docker image inspect ubuntu:latest
```
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

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

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>First instruction in Dockerfile and it identifies the image to inherit from</td>
</tr>
<tr>
<td>MAINTAINER</td>
<td>Provides visibility and credit to the author of the image</td>
</tr>
<tr>
<td>RUN</td>
<td>Executes a Linux command for configuring and installing</td>
</tr>
<tr>
<td>ENTRYPOINT</td>
<td>The final script or application used to bootstrap the container, making it an executable application</td>
</tr>
<tr>
<td>CMD</td>
<td>Provide default arguments to the ENTRYPOINT using a JSON array format</td>
</tr>
<tr>
<td>LABEL</td>
<td>Name/value metadata about the image</td>
</tr>
<tr>
<td>ENV</td>
<td>Sets environment variables</td>
</tr>
<tr>
<td>COPY</td>
<td>Copies file into the container</td>
</tr>
<tr>
<td>ADD</td>
<td>Alternative to copy</td>
</tr>
<tr>
<td>WORKDIR</td>
<td>Sets working directory for RUN, CMD, ENTRYPOINT, COPY, and/or ADD instructions</td>
</tr>
<tr>
<td>EXPOSE</td>
<td>Ports the container will listen on</td>
</tr>
<tr>
<td>VOLUME</td>
<td>Creates a mount point</td>
</tr>
<tr>
<td>USER</td>
<td>User to run RUN, CMD, and/or ENTRYPOINT instructions</td>
</tr>
</tbody>
</table>
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 alpine
LABEL maintainer= "gilbert.toussido@gmail.com"
RUN apk add -update nodejs nodejs-npm
COPY . /src
WORKDIR /src
RUN npm install
EXPOSE 8080
ENTRYPOINT ["node", "/app.js"]
```
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

# 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
----> 9b5e2e22af1be
Step 2/5 : RUN apt-get update & apt-get install -y python3 pip
----> Using cache
----> 7fd0380c8e30
Step 3/5 : RUN pip install flask
----> Using cache
----> 9b7f2228bd9b
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

```bash
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)

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

Now we can push it to Docker Hub

```bash
root@node1:/home/vagrant# docker push gilbertfongan/newflaskapp
Using default tag: latest
b0fee53fe4bf: Pushing ........................................... 3.578MB/4.707MB
b0fee53fe4bf: Pushing ........................................... 4.085MB/4.707MB
```
Docker Push image

Docker is updating and extending our product subscriptions. Please read our blog for more information.

Advanced Image Management
View all your images and tags in this repository, clean up unused content, recover untagged images. Available for Pro and Team accounts.

gilbertfongan/newflaskapp
This repository does not have a description

Last pushed: 3 hours ago

Docker commands
To push a new tag to this repository,

docker push gilbertfongan/newflaskapp:tagname
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.

```bash
# Dockerfile
# build stage
FROM buildbase as build
...
...
...
# production ready stage
FROM runbase
...
COPY --from=build /artifact /app
```
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

<table>
<thead>
<tr>
<th>Commands</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker image build (-t &lt;repository_name&gt;):&lt;tagname&gt; &lt;build_context&gt;)</td>
<td>Command that reads a Dockerfile and containerizes the application.</td>
</tr>
<tr>
<td></td>
<td>-t flag tags the image</td>
</tr>
<tr>
<td></td>
<td>-f flag lets you specify the name and location of the Dockerfile</td>
</tr>
<tr>
<td>Docker image push (&lt;repository_name&gt;:&lt;tagname&gt;)</td>
<td>Push containerized app to image registry (by default Docker Hub)</td>
</tr>
<tr>
<td>Docker login [OPTIONS][SERVER]</td>
<td>Log in into a Docker registry</td>
</tr>
<tr>
<td>Docker tag \SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]\</td>
<td>Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE</td>
</tr>
</tbody>
</table>
Docker workflow
Docker Usage Sheet

Commands Cheat Sheet

**Container Lifecycle**
- `docker create [IMAGE]`: create a container without starting it
- `docker rename [CONTAINER_NAME] [NEW_CONTAINER_NAME]`: rename a container
- `docker run [IMAGE]`: create and start a container
- `docker run --rm [IMAGE]`: remove a container after it stops
- `docker run -tid [IMAGE]`: start a container and keep it running
- `docker run -it [IMAGE]`: create, start the container, and run a command in it
- `docker run -it-rm [IMAGE]`: create, start the container, and run a command in it; after executing, the container is removed
- `docker rm [CONTAINER]`: delete a container if it isn't running
- `docker update [CONTAINER]`: update the configuration of a container

**Networking**
- `docker network ls`: list networks
- `docker network rm [NETWORK]`: remove one or more networks
- `docker network inspect [NETWORK] [CONTAINER]`: 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

**Image Lifecycle**
- `docker build [URL]`: create an image from a Dockerfile
- `docker build + [URL]`: build an image from a Dockerfile and tag it
- `docker pull [IMAGE]`: pull an image from a registry
- `docker push [IMAGE]`: push an image to a registry
- `docker import [URL/FILE]`: create an image from a tarball
- `docker commit [CONTAINER] [NEW_IMAGE_NAME]`: create an image from a container
- `docker rm [IMAGE]`: remove an image
- `docker load [TAR_FILE/STDIN_FILE]`: load an image from a tar archive as stdin
- `docker save [IMAGE] > [TAR_FILE]`: save an image to a tar archive stream to stdout with all parent layers, tags, and versions

**Start & Stop**
- `docker start [CONTAINER]`: start a container
- `docker stop [CONTAINER]`: stop a running container
- `docker restart [CONTAINER]`: stop 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

**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 resources usage statistics of containers
- `docker diff [CONTAINER]`: show changes to files (or directories) on a filesystem
- `docker images ls`: show all locally stored images
- `docker history [IMAGE]`: show history of an image
MODULE 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 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
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**
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)


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

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>CREATED</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter-app_web.fe</td>
<td>latest</td>
<td>67d24725fb77</td>
<td>3 minutes ago</td>
<td>51.7MB</td>
</tr>
<tr>
<td>gilbertfongan/newflaskapp</td>
<td>latest</td>
<td>12d37cae3628</td>
<td>31 hours ago</td>
<td>438MB</td>
</tr>
<tr>
<td>python</td>
<td>3.6-alpine</td>
<td>c5917c34066a</td>
<td>2 weeks ago</td>
<td>40.8MB</td>
</tr>
<tr>
<td>ubuntu</td>
<td>latest</td>
<td>fb52e22af1b0</td>
<td>3 weeks ago</td>
<td>72.8MB</td>
</tr>
<tr>
<td>redis</td>
<td>alpine</td>
<td>f6ff2956798e9</td>
<td>4 weeks ago</td>
<td>32.3MB</td>
</tr>
</tbody>
</table>

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.),

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>NAMES</th>
<th>COMMAND</th>
<th>CREATED</th>
<th>STATUS</th>
<th>PORTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5e9f2100d256b</td>
<td>redis:alpine</td>
<td>counter-app_redis_1</td>
<td>&quot;docker-entrypoint.sh&quot;</td>
<td>About a minute ago</td>
<td>Up</td>
<td>About a minute 6379/tcp</td>
</tr>
<tr>
<td>3f32b0f85b7b</td>
<td>counter-app_web-fe</td>
<td>python app.py</td>
<td>0-&gt;5000/tcp, :::5000-&gt;5000/tcp</td>
<td>About a minute ago</td>
<td>Up</td>
<td>About a minute 0.0.0.0:5000</td>
</tr>
</tbody>
</table>

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:

```bash
root@node1:/home/vagrant/counter-app# docker network ls
NETWORK ID       NAME                DRIVER   SCOPE
3ce814c2603d     bridge             bridge   local
615c65b43334     counter-app_counter-net bridge   local
17b2772f5e7      host               host     local
c364deaf90e8     none                null     local
root@node1:/home/vagrant/counter-app# docker volume ls
DRIVER VOLUME NAME
local  3a16754d5c9ac1588efb2342dd94965d6471bde703626bb0f6e828b6b7becb1f40
local  counter-app_counter-net
```

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

```
What's up Docker Deep Divers! You've visited me 1 times.
```

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

```bash
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

<table>
<thead>
<tr>
<th>COMMANDS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker-compose up</td>
<td>Deploy a Compose App. It expects the Compose file to be called <code>docker-compose.yml</code> or <code>docker-compose.yaml</code>, but you can specify a custom filename with the <code>-f</code> flag. It’s common to start the App in the background with the <code>-d</code> flag.</td>
</tr>
<tr>
<td>Docker-compose stop</td>
<td>Stop all containers in a Compose App without deleting them from the system.</td>
</tr>
<tr>
<td>Docker-compose restart</td>
<td>Restart a Compose App that has been stopped with <code>docker-compose stop</code>. 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.</td>
</tr>
<tr>
<td>Docker-compose ps</td>
<td>List each container in the Compose App. It shows the current state, the command each one is running, and network ports.</td>
</tr>
<tr>
<td>Docker-compose down</td>
<td>Stop and delete a running Compose App. It deletes containers and networks, but not volumes and images.</td>
</tr>
</tbody>
</table>
At the highest level, **Docker networking** comprises three major components:

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

---

**Diagram:**

- **CNM** (Design (DNA))
- **libnetwork** (Primitives (Control & Management plane))
- **Drivers** (Networks (Data plane))

---

**Docker Networking**

- **Sandbox**
  - **EP**
  - **Network**
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

Core networking
- Sandboxes
- Endpoints
- Networks
  - Service discovery
  - Load balancing

Libnetwork

Pluggable interface

Network specifics
- Bridge driver (Single-host) (a.k.a. NAT on Windows)
- Overlay driver (Multi-host) (Container-only)
- MACVLAN driver (Existing VLANs) (promiscuous mode)

Docker Engine

{API}
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 have 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
  
  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.
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.

Command to create a new single-host bridge network called “localnet”:
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)

The Docker Engine connects the container to the bridge network by default.

*eth0* is created by the bridge driver and an address is given by the Docker native IPAM driver.

*veth* is a virtual ethernet interface which connects bridge to the *eth0* interface inside the container.

*docker0* is a Linux bridge that exists in the host network namespace.

The routing table connects *docker0* and *eth0* on the external network.
Single-host User-Defined Bridge Network

Containers

Virtual ethernet interfaces

A user-defined bridge that connects to two containers via two interfaces

The routing table connects `docker0`, `my_bridge`, and `eth0` on the external network
## Single-host Bridge Network Comparison

<table>
<thead>
<tr>
<th>Features</th>
<th>Default</th>
<th>User-Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better isolation and interoperability between containerized applications</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic DNS resolution between containers</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Attachment and detachment of containers on the fly</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Configurable bridge creation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Linked containers share environment variables</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
**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

These containers connect with each other using a *localhost* on C1.

The containers C1 and nginx are using the host network and share the same interface for *eth0*.
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.
Provisioning for an Overlay Network is automated by Docker Swarm Control Plane

![Diagram of Multi-Host Overlay Network](image)

- **Virtual ethernet interfaces**
- **An ovnet is a bridge with the overlay driver**
- **A docker_gwbridge is an egress bridge**
To create a new overlay network: 

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

To list all networks on each node: 

```
$ docker network ls
```

To attach a service to the Overlay Network: 

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

To inspect a created Overlay Network: 

```
$ docker network inspect devops
```
Multi-Host Overlay Network Use case

The application *pets* is deployed on *db* and *web*. These two containers are present on different hosts.

The overlay network, which is created by a Universal Control Panel, allows the *db* and *web* to connect to each other.
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.
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
Containers formed on `mvnet` network

An interface `eth0` on the host is bound to `mvnet (a MACVLAN network)`

An external gateway is required during MACVLAN network configuration

MAC VLAN (3/4)
The MACVLAN driver needs this 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

- "db and web are present on the same host and connected to different MACVLAN networks."
- "Two MACVLAN networks are created and joined to different sub-interfaces."
- "The external networks are specific for specific containers."
**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 adress 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 know the IP adress of container « c2 »

4. [Same Network] DNS resolver returns the IP adress of « c2 » to the local resolver in « c1 »

5. The ping command is sent to the corresponding target IP adress of « c2 »
## Docker Network Commands

<table>
<thead>
<tr>
<th>COMMANDS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker network ls</td>
<td>List all networks on the local Docker host.</td>
</tr>
<tr>
<td>Docker network create</td>
<td>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 specify with the –d flag</td>
</tr>
<tr>
<td>Docker network inspect</td>
<td>Provides detailed configuration information about a Docker Network</td>
</tr>
<tr>
<td>Docker network prune</td>
<td>Deletes all unused networks on a Docker host.</td>
</tr>
<tr>
<td>Docker network rm</td>
<td>Deletes specific networks on a Docker host</td>
</tr>
</tbody>
</table>
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 high 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 (m01tzqlfrus72zjki6s2v2z32) is now a manager.

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

  docker swarm join --token SHMTKN-1-336qe1yp4arv0eiy10t1616h6uwywh1g6b50ea2c4b41z5015uuj-1fqs9x4bayg31vd6ezfsbqqudt 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 XXXXXXXXXXXXX 172.17.8.104:2377 (Run from Worker)

root@node2:/home/vagrant# docker swarm join --token 5WMTKN-1-33Gqeiyp4arwOeyi0t161G6uJyw1gb50ea2yg3ivdGezfsbqqudt 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 XXXXXXX...XXXXXXXXX\n  172.17.8.4:2377 \n  --advertise-addr 172.17.8.40:2377 \n  --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
  m91tzq1fus72z1jk16dvz2s2 * node1    Ready  Active     Leader      20.10.8
  rh8n6xvpx4jvpqbgzg7rpi node2    Ready  Active
  ```
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
v88hrzo3dfc5wvu584e88c4ar
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
qhalf64kopc6xucxhldnypeo87
overall progress: 2 out of 2 tasks
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      IMAGE         NODES  REPLICAS  IMAGE          PORTS
v80hzoa8dfc5 hello-world replicated 1/1     alpine:latest *:6379->6379/tcp
qy2r644upk0   redis   replicated 2/2     redis:latest
```

$ docker service ps redis

```
root@node1:/home/vagrant# docker service ps redis
ID     NAME      IMAGE          NODES  REPLICAS  IMAGE          PORTS
m38vhq4vqrw redis.1 redis:latest node2 Running Running 2 minutes ago
=aeeqsh4gd redis.2 redis:latest node1 Running Running about a minute ago
```

$ docker service inspect redis

```
root@node1:/home/vagrant# docker service inspect redis

```

```

```

```
```
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     IMAGE         MODE     REPLICAS NODES DESIRED STATE CURRENT STATE
f3mlbyncfkm hello-world replicated 1/1 alpine:latest
znod5gqkur5k redis replicated 4/4 redis:latest *.8369->8369/tcp

$ docker service ps redis

root@node1:/home/vagrant# docker service ps redis
ID     NAME     IMAGE         NODE     DESIRED STATE CURRENT STATE
maqu7646m@og redis.1 redis:latest node2 Running Running 6 minutes ago
pslwult438k redis.2 redis:latest node3 Running Running 6 minutes ago
gfwardyvuv4ud redis.3 redis:latest node2 Running Running 4 minutes ago
pitpde1lhrot redis.4 redis:latest node1 Running Running about a minute ago
```
Swarm supports two 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
  # docker service create -d --name example2 
  --publish published=5001, target=8080, mode=host 
  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
  ```
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**

  ```bash
  $ docker network ls
  root@node1:/home/vagrant# docker network create -d overlay devops
  tkhh4vf5o1yyup9@mxwv4j7m7
  root@node1:/home/vagrant# docker network ls
  NETWORK_ID  NAME       DRIVER   SCOPE
  8d3e7895a564  bridge   bridge   local
  tkhh4vf5o1v  devops   overlay   swarm
  4a8eb79b81bd  docker_gwbridge  bridge   local
  17b2b772f5e7  host     host     local
  bxlxoa7r4m1  ingress   overlay   swarm
  c364deaf96e8  none     null     local
  ```

- **Deploy a new service and attach it to the network**

  ```bash
  $ 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
  12 gilbertfongan/demo:v1
  jh30w7meh4tkvihn22gchnt
  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           PORTS
  f3mbvncfkm hello-world replicated 1/1 alpine:latest *:80->80/tcp
  znod5gqyrS5k redis      replicated 1/1 redis:latest        *:6379->6379/tcp
  jh9y7b4hkv 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

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

```bash
$ 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

```bash
$ docker stack ls
<table>
<thead>
<tr>
<th>NAME</th>
<th>SERVICES</th>
<th>ORCHESTRATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>webapp</td>
<td>2</td>
<td>Swarm</td>
</tr>
</tbody>
</table>

$ docker stack services webapp

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>MODE</th>
<th>REPLICAS</th>
<th>IMAGE</th>
<th>PORTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>j0uprwm4oua9</td>
<td>webapp_mysql</td>
<td>replicated</td>
<td>1/1</td>
<td>mysql:latest</td>
<td></td>
</tr>
<tr>
<td>qg4nqt7smo3</td>
<td>webapp_web</td>
<td>replicated</td>
<td>1/1</td>
<td>nginx:latest</td>
<td>*:808</td>
</tr>
<tr>
<td>1-&gt;80/tcp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Stacks deploy example

**Task making up the services**

```bash
$ docker stack ps webapp
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>IMAGE</th>
<th>NODE</th>
<th>DESIRED</th>
<th>STATE</th>
<th>CURRENT STATE</th>
<th>ERROR</th>
<th>PORTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ow1buo3i4mfj</td>
<td>webapp_mysql.1</td>
<td>mysql:latest</td>
<td>node1</td>
<td></td>
<td>Running</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Running</td>
<td></td>
<td>12 minutes</td>
<td></td>
</tr>
<tr>
<td>jnv24yv3ekz</td>
<td>webapp_web.1</td>
<td>nginx:latest</td>
<td>node2</td>
<td></td>
<td>Running</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Running</td>
<td></td>
<td>12 minutes</td>
<td></td>
</tr>
</tbody>
</table>

**Delete Docker stacks**

```bash
$ docker stack rm webapp
```

Removing service webapp_mysql
Removing service webapp_web
Removing network webapp_default
## Docker Stacks Commands

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

Features of Kubernetes

- Pod
- Horizontal Scaling
- Self-Healing
- Persistent Storage
- Automated rollouts & rollbacks
- Automatic Bin Packing
- Service Discovery & load balancing
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)
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 internet connection
[preflight] You can also perform this action in beforehand using `kubeadm config images 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.default.svc kubelet.kubernetes.default.svc.cluster.local node01] and IPs [10.0.0.1 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 IPs [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 ach605.8mglx3nt027tcs  
--discovery-token-ca-cert-hash sha256:ebc51f8b4310372d3ef5330a67d52ad7685f968b3c04bd59fd6fb110b0
```
Join the cluster with a Worker

$ kubeadm join 10.10.1.120:6443 --token achk05.u8mglsx3nt027tcs --discovery-token-ca-cert-hash sha256:ebc61f8b4c310372d3e5f539a67d52ad76859468eb63cac4dda55d5eb110b0

Check all nodes from Master node

$ kubectl get nodes
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

Example:

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

- **ExternalName**: Exposes the Service using an arbitrary name (specified by `externalName` in the Spec) by returning a CNAME record with the name.
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

```yaml
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

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

```yaml
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
```
<table>
<thead>
<tr>
<th>COMMANDS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kubectl create deployment nginx --image=nginx</code></td>
<td>Command used to create deployment based on the YAML file &quot;DEPLOY.yaml&quot;</td>
</tr>
<tr>
<td><code>kubectl apply -f DEPLOYMENT.yaml</code></td>
<td></td>
</tr>
<tr>
<td><code>kubectl get deployments</code></td>
<td>List all deployments in the namespace. With [--all-namespaces] to list in all namespaces</td>
</tr>
<tr>
<td><code>kubectl get deployment DEPLOYMENT</code></td>
<td>List a particular deployment</td>
</tr>
<tr>
<td><code>kubectl describe deployment DEPLOYMENT</code></td>
<td>Description of a Deployment</td>
</tr>
<tr>
<td><code>kubectl delete deployment DEPLOYMENT</code></td>
<td>Remove or delete a Deployment. Add [-l name=myLabel] to delete deployment with Label name=myLabel</td>
</tr>
<tr>
<td><code>kubectl set image deployment/DEPLOYMENT www=image:v2</code></td>
<td>Rolling update &quot;www&quot; containers of &quot;DEPLOYMENT&quot;, updating the image</td>
</tr>
<tr>
<td><code>kubectl rollout history deployment/DEPLOYMENT</code></td>
<td>Check the history of deployments including the revision</td>
</tr>
<tr>
<td><code>kubectl rollout undo deployment/DEPLOYMENT</code></td>
<td>Rollback to the previous deployment revision</td>
</tr>
<tr>
<td><code>kubectl rollout undo deployment/DEPLOYMENT --to-revision=2</code></td>
<td>Rollback to a specific revision</td>
</tr>
<tr>
<td><code>kubectl rollout status --w deployment/DEPLOYMENT</code></td>
<td>Watch rolling update status of &quot;DEPLOYMENT&quot; deployment until completion</td>
</tr>
<tr>
<td><code>kubectl rollout restart deployment/DEPLOYMENT</code></td>
<td>Rolling restart of the &quot;DEPLOYMENT&quot; deployment</td>
</tr>
<tr>
<td><code>kubectl autoscale deployment DEPLOYMENT --min=2 --max=10</code></td>
<td>Auto scale a deployment &quot;DEPLOYMENT&quot;</td>
</tr>
</tbody>
</table>
MODULE 3 : Container Infrastructure
PLAN

- DOCKER MACHINE
- DOCKER DESKTOP
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.
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-deprecated)**

  ```
  && 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} \n  --generic-ssh-key ${SSH_PUBLIC_KEY} \n  --generic-ssh-user ${SSH_USER} \n  new-machine
  ```
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
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

<table>
<thead>
<tr>
<th>COMMANDS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker-machine config</td>
<td>Print the connection config for machine</td>
</tr>
<tr>
<td>Docker-machine env</td>
<td>Display the commands to set up the environment for the Docker client</td>
</tr>
<tr>
<td>Docker-machine inspect</td>
<td>Inspect information about a machine</td>
</tr>
<tr>
<td>Docker-machine ip</td>
<td>Get the IP address of the machine</td>
</tr>
<tr>
<td>Docker-machine kill</td>
<td>Kill a machine</td>
</tr>
<tr>
<td>Docker-machine ls</td>
<td>List machines</td>
</tr>
<tr>
<td>Docker-machine provision</td>
<td>Re-provision existing machines</td>
</tr>
<tr>
<td>Docker-machine regenerate-certs</td>
<td>Regenerate TLS Certificates for a machine</td>
</tr>
<tr>
<td>Docker-machine restart</td>
<td>Restart a machine</td>
</tr>
<tr>
<td>Docker-machine start</td>
<td>Start a machine</td>
</tr>
<tr>
<td>Docker-machine status</td>
<td>Get the status of a machine</td>
</tr>
<tr>
<td>Docker-machine stop</td>
<td>Stop a machine</td>
</tr>
<tr>
<td>Docker-machine upgrade</td>
<td>Upgrade a machine to the latest version of Docker</td>
</tr>
</tbody>
</table>
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/](https://docs.docker.com/desktop/)
PART I.V. Configuration Management
MODULE 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® 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?

- Orchestration
- Provisionning
- Automation
- Config. Management
- App. Deployment
- Push-based (stateless)
Ansible features

- **Management Node**: Controls 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 stack 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)

Content of config file

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

[privelege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
[msleczek@vm0-net projekt A]$
```
Ansible uses SSH for authentication and assumes keys are in place (Controller)

```
$ ssh-keygen
```

Activate PubkeyAuthentication & ChallengeResponseAuthentication (Target)
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)
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)

```
[web servers]
mail.example.com
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:  
```
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.
Parallelism and shell commands

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

```bash
$ 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 machines in `webservers` group

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

- Creating a new directory

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

- Deleting directories (recursively) and files

```bash
$ 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
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.
Playbooks are a collection of one or more plays
Each play maps 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

-vagrant@ansible-controller ansible_demo]$ ansible-playbook ping.yml -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
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
    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*

  - 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
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"

Jinja2 expression

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

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

  notify:
    - openport80
    - startwebserver

  handlers:
    - name: openport80
      service: name=httpd state=started
    - name: startwebserver
      firewalld: port=80/tcp permanent=true state=enabled immediate=yes
Ansible Playbook roles & vars

Creating a role

$ 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
Ansible Playbook roles & vars

- **Edit tasks files**
  
  `$ sudo nano linuxwebserver/tasks/main.yml`

- **Edit Handlers files**
  
  `$ sudo nano linuxwebserver/handlers/main.yml`
Ansible Playbook roles & vars

### Write playbook which call the role created

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

### Edit Inventory files

```bash
$ sudo nano hosts
```

```yaml
[vagrant@node1 deploy with role]$ sudo nano hosts
[linuxservers]
anibleclient2 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
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
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

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

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

```yaml
---
- 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

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

```yaml
[linuxservers]
ansibleclient2 ansible_host=172.17.11.5

[linuxservers:vars]
```
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.
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
Despite having changed the packages to be installed and the server port on the inventory, their explicit definition via the runtime prevails.
Ansible Galaxy

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

Ansible Tower is a platform for software and hardware management that provides automation across infrastructure and cloud environments. It includes features such as Ansible Playbooks for configuration management, Ansible CLI & CI Systems for integrating with continuous integration and continuous deployment (CI/CD) pipelines, and tools for role-based access control and knowledge visibility.

Ansible Tower is designed to simplify the deployment and management of complex IT environments by allowing administrators to automate their infrastructure and applications. It supports a wide range of use cases, including provisioning, configuration management, app deployment, continuous delivery, security & compliance, and orchestration.

Key components of Ansible Tower include:
- **Ansible Playbooks**: A collection of scripts that define the configuration, application deployment, and infrastructure management tasks.
- **Ansible CLI & CI Systems**: Integration with CI/CD tools for automating the deployment and testing of applications.
- **Role-Based Access Control**: Security controls to manage user roles and permissions.
- **Knowledge & Visibility**: Tools for tracking and managing knowledge related to configurations and deployments.
- **Scheduled & Centralized Jobs**: Tools for scheduling and managing tasks to be executed at predefined times.
- **Simple User Interface**: A user-friendly interface for administrators and users.
- **Tower API**: An API for integrating Ansible Tower with other systems and tools.
- **Open Source Module Library**: A library of modules that can be used to perform specific tasks.
- **Plugins**: Additional features and functionalities that can be installed to extend the capabilities of Ansible Tower.
- **Python Codebase**: The underlying codebase that powers Ansible Tower.
- **Infrastructure**: Support for various operating systems and network devices, including Linux, Windows, Cisco, Juniper, and more.
- **Networks**: Support for network devices and protocols.
- **Containers**: Support for container technologies like Docker, LXC, and more.
- **Cloud**: Support for cloud environments such as AWS, Google Cloud, and Azure.
- **Services**: Support for various services, including databases, logging, source control management, and more.

Ansible Tower provides a comprehensive solution for managing IT environments, enabling administrators to automate and streamline their operations, ensuring efficiency, scalability, and security.
Ansible Tower (AWX) is a web-based solution that makes Ansible even more easy to use.
MODULE 2 : Other Tools
PLAN

- **CHEF**
  - Chef Expression
  - Chef Workstation
  - Chef Server
  - Chef Nodes
  - Chef Tools

- **PUPPET**
  - Architecture
  - Commands
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
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 create. 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.
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**
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 be 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
```
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

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

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

Chef-solo
(Stand-alone)
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

<table>
<thead>
<tr>
<th>COMMANDS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puppet facter</td>
<td>Show all facters</td>
</tr>
<tr>
<td>Puppet agent --enable</td>
<td>Enable puppet agent to run on node</td>
</tr>
<tr>
<td>Puppet resource package</td>
<td>Show all installed packages</td>
</tr>
<tr>
<td>Puppet resource</td>
<td>Show all managed resources</td>
</tr>
<tr>
<td>Puppet module list</td>
<td>List all installed module</td>
</tr>
<tr>
<td>Puppet config print all</td>
<td>Print all configuration settings</td>
</tr>
</tbody>
</table>
PART I.V. Service Operations
MODULE 1 : IT Operations and Monitoring
DEVOPS MONITORING
PROMETHEUS
- Architecture
- Configuration file
- Metric types
- Exporter

GRAFANA
- Data source
- Dashboard
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 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

- **Application (Client Library)**
  - Pull Metrics
  - Prometheus Server
    - Stores Data
    - Local Storage
    - PromQL
    - Push Alerts
      - Alertmanager
    - Find Targets
      - Service Discovery
    - Notifications
      - OpsGenie
      - pagerduty
  - Exporter
  - 3rd Party Application
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
- **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.

```yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

alerting:
  alertmanagers:
  - static_configs:
    - targets:
        # -alertmanager:9093

rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
- job_name: 'prometheus'
  scrape_interval: 5s
  static_configs:
  - targets: ['172.17.8.104:9090']
  - job_name: 'node1'
    scrape_interval: 5s
    static_configs:
    - targets: ['172.17.8.105:9100']
  - job_name: 'node2'
    scrape_interval: 5s
    static_configs:
    - targets: ['172.17.8.105:9100']
```
Prometheus Metric Types

Prometheus client libraries offer four core metric types

- **Counter**
  It is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart

- **Gauge**
  It is a metric that represents a single numerical value that can arbitrarily go up and down

- **Summary**
  Shows the total count of observations and the sum of observed values after sampling observations

- **Histogram**
  For sampling request durations, response sizes and similar observations. It usually counts in buckets and provides the sum of all observed values.
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)
Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs.

It 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 consumer 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
Grafana
Grafana Data Sources

Data Sources / prometheus

- Name: prometheus
- Type: Prometheus

HTTP
- URL: http://localhost:9090
- Access: proxy

Auth
- Basic Auth
- With Credentials
- TLS Client Auth
- With CA Cert
- Skip TLS Verification (precaution)

Advanced HTTP Settings
- Whitelisted Cookies
- Add Names
- Scrapes interval: 1s

Data source is working
Grafana Dashboard
MODULE 2 : Log Management and Analysis
PLAN

- LOGS
- ELASTICSEARCH
- LOGSTASH
- KIBANA
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 allows 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
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, High availability and multi-tenant
Elasticsearch Architecture
Elasticsearch Architecture

- **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

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

- **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

- **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 parrellelly across the replicas
**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
**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

```ruby
input {
  file {
    path => ["/var/log/mysql/mysql.log", "/var/log/mysql/mysql-show.log", "/var/log/mysql/mysql-error.log"
    type => "mysql"
  }
}
```

Syslog

```ruby
input {
  syslog {
    port => 514
    codec => cef
    syslog_field => "syslog"
    grok_pattern => "<%{POSINT:priority}>%{SYSTIMESTAMP:timestamp} CUSTOM GROK HERE"
  }
}
```

Redis

```ruby
input {
  redis {
    id => "my_plugin_id"
  }
}
```
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**

```ruby
filter {
  if [loglevel] == "debug" {
    drop { }
  }
}
```

**GROK (from http request log)**

```ruby
55.3.244.1 GET /index.html 15824 0.043
```

```ruby
filter {
  grok {
    match => { "message" => "%(IP:client) %(WORD:method) %(URIPATHPARAM:request) %(NUMBER:bytes) %(NUMBER:duration)" }  
  }
}
```

**MUTATE**

```ruby
filter {
  mutate {
    split => ["hostname","",""]
    add_field => {"shortHostname"=>"%{hostname[0]}"
  }
}
```
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**
```ruby
output {
  elasticsearch {
    hosts => "hostname"
    data_stream => "true"
  }
}
```

**FILE**
```ruby
output {
  file {
    path => "C:/devops-book-labs/logstash/bin/log/output.log"
    codec => line { format => "custom format: %{message}"}
  }
}
```

**Statsd**
```ruby
output {
  statsd {
    host => "statsd.example.org"
    count => {
      "http.bytes" => "%{bytes}"
    }
  }
}
```
Logstash Complete Example

```
input {
  file {
    path => "/var/log/httpd/access_log"
    type => "apache-access"
    start_position => "beginning"
  }
}

filter {
  if [type] == "apache-access"
  {
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
    date {
      match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
    }
  }
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
  }
  stdout { codec => rubydebug }
}
```
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
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/ Standard Components and Platforms

https://www.bridge-global.com/blog/mutable-vs-immutable-infrastructure/
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.bmc.com/blogs/cap-theorem/
https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/
https://hevodata.com/learn/relationa-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
https://www.cloudbampp.com/blog/what-is-message-queueing.html
https://blog.devgenius.io/everything-about-distributed-message-queue-ae6597d84b36
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://www.slideshare.net/KishrorKumar/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/
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.leewayhertz.com/software-testing-process/
https://quodem.com/en/blog/what-is-software-testing-outsourcing/
https://www.leewayhertz.com/software-testing-process/
https://www.greedy.com/tag/types-of-test-tools
https://aws.amazon.com/devops/continuous-delivery/?nc1=h_js
https://www.spiceworks.com/tech/devops/articles/cicd-vs-devops/
https://1902software.com/blog/automated-deployment/
https://www.gocd.org/2017/07/25/blue-green-deployments.html
https://www.gocd.org/2017/08/15/canary-releases/
https://formation.cloud-gfi-nord.fr/assets/integrationContinue/CI-Jenkins.pptx
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/
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://medium.com/happy-giraffe/vagrant-box-up-and-go-88f5ff5d09d1
https://slideplayer.com/slide/13686844/
References/ Cloud Deployment

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/
References/ System Image Creation

https://www.packer.io/intro/use-cases
https://learn.hashicorp.com/tutorials/packer/get-started-install-cli
https://devopskube.com/packer-tutorial-for-beginners/
https://lzone.de/ch
https://developer.hashicorp.com/packer/docs/commands/buildsheet/
References/Container Usage

https://www.slideteam.net/introduction-to-dockers-and-containers-powerpoint-presentation-slides.html#images-3
https://www.slideteam.net/introduction-to-dockers-and-containers-powerpoint-presentation-slides.html
https://medium.com/@mccode/using-semantic-versioning-for-docker-image-tags-dfde8be06699
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://aboullaite.me/multi-stage-docker-java/
https://www.padok.fr/en/blog/docker-image-multi-staging
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://docs.docker.com/engine/tutorials/networking-containers/
https://www.section.io/engineering-education/introduction-to-docker-swarm-in-container-orchestration/
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.slideshare.net/rajdeep/introduction-to-kubernetes
https://www.slideshare.net/rajdeep/introduction-to-kubernetes
https://www.slideshare.net/rajdeep/introduction-to-kubernetes
https://kodekloud.com/blog/kubernetes-features-for-beginner/
https://www.waytomeylabel.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://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
References/Container Infrastructure

https://www.lpi.org/blog/2018/02/27/devops-tools-introduction-08-container-infrastructure
https://github.com/docker/machine
https://medium.com/@cnadeau_/docker-machine-basic-examples-7d1ef640779b
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://datascientest.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/rahulkul48837211/ansible-architecture-and-setup-2355
https://www.fita.in/devops-tutorial/
https://www.openvirtualization.pro/red-hat-ansible-automation-architecture-and-features/
https://www.fita.in/devops-tutorial/
https://www.fita.in/devops-tutorial/
https://installation-guide-ansible-documentation
https://docs.ansible.com/ansible/latest/reference_appendices/general_precedence.html#general-precedence-rules
https://www.openvirtualization.pro/ansible-part-i-basics-and-inventory/
https://www.ansible.com/ansible/latest/user_guide/intro_inventory.html#intro-inventory
https://www.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/knolius/introduction-to-ansible-81369741
https://www.bogotobogo.com/DevOps/Ansible/Ansible_SettingUp_Webservers_Nginx_Install_Env_Configure_Deploy_App.php
https://www.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html
https://www.ansible.com/ansible/latest/playbook_guide/playbooks_loops.html
https://linuxhint.com/ansible-with_item/
https://www.ansible.com/ansible/latest/ansible-tower-overview
References/ Other Configuration Management Tools

https://www.tutorialspoint.com/chef/chef_overview.htm
https://docs.chef.io/data_bags/
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.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.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.mezmo.com/learn-log-management/what-is-log-analysis#text=Log%20analysis%20also%20provides%20significant%20information%20about%20user%20behavior.
https://www.oreilly.com/library/view/learning-elastic-stack/9781787281868/5a99bb-a9d6-4338-814c-3058e1fa4e3f.xhtml
THANKS!

Any questions?
You can find me at gilbert.toussido@gmail.com