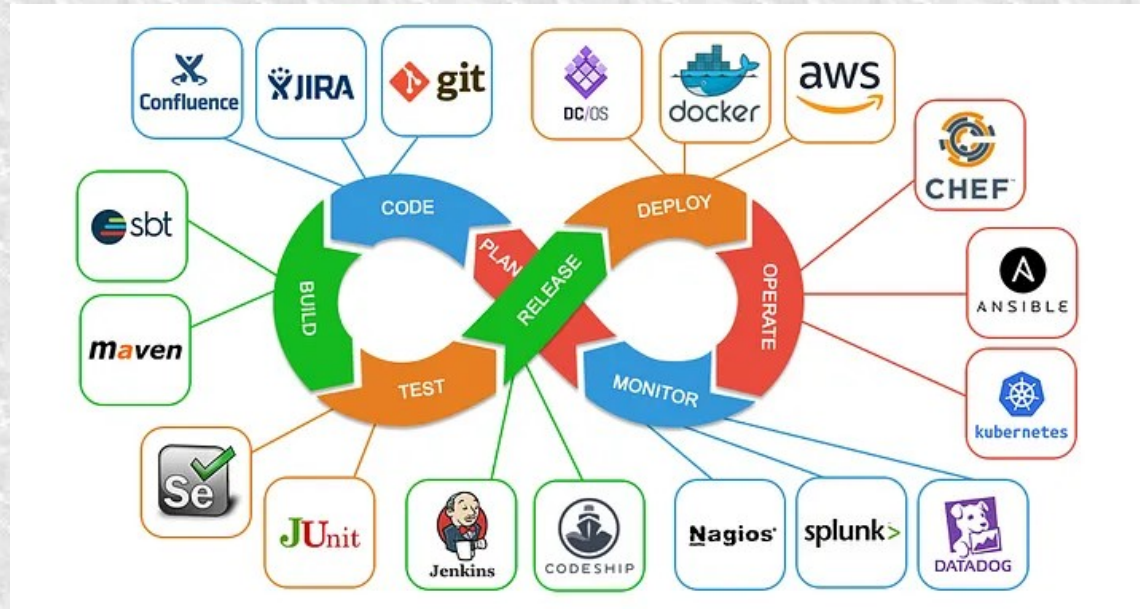
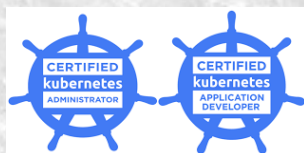


DevOps Tools Engineers



Brahim HAMDI

brahim.hamdi.consult@gmail.com



Avril 2024

Plan

- **Module 1** : **Modern Software Development**
- **Module 2** : **Source code management**
- **Module 3** : **System image creation and VM Deployment**
- **Module 4** : **Container usage**
- **Module 5** : **Container Deployment and Orchestration**
- **Module 6** : **Ansible and configuration management tools**
- **Module 7** : **CI / CD with Jenkins**
- **Module 8** : **IT monitoring**
- **Module 9** : **Log management and analysis**

DevOps Tools Engineers

Module 1
Modern Software
Development

Plan

- Agile
- Service based applications
- RESTful APIs
- IaaS, CaaS & PaaS

Agile

What is Agile ?

- Software development methodology.
- A set of values and principles
- Evolutionary and iterative development
- Continuous improvement
- Rapid and flexible response to change
- Scrum is the most widely used Agile method.
- Others agile methods and practices :
 - Kanban
 - Extreme Programming (XP)
 - Feature-Driven Development (FDD)
 - Test-Driven Development (TDD)
 - DevOps

Agile

The Manifesto for Agile Software Development

- Individuals and Interactions more than processes and tools.
- Working Software more than comprehensive documentation.
- Customer Collaboration more than contract negotiation.
- Responding to Change more than following a plan.

Agile

Agile vs DevOps

| Agile | DevOps |
|---------------------------|--|
| Feedback from customer | Feedback from self |
| Smaller release cycles | Smaller release cycles, immediate feedback |
| Focus on speed | Focus on speed and automation |
| Not the best for business | Best for business |

Service based applications

Application architecture

- Why does application architecture matter?
 - Build a product can scale.
 - To distribute.
 - Helps with time to market
- Application architectures:
 - Monolithic Architecture
 - SOA Architecture
 - Microservices Architecture

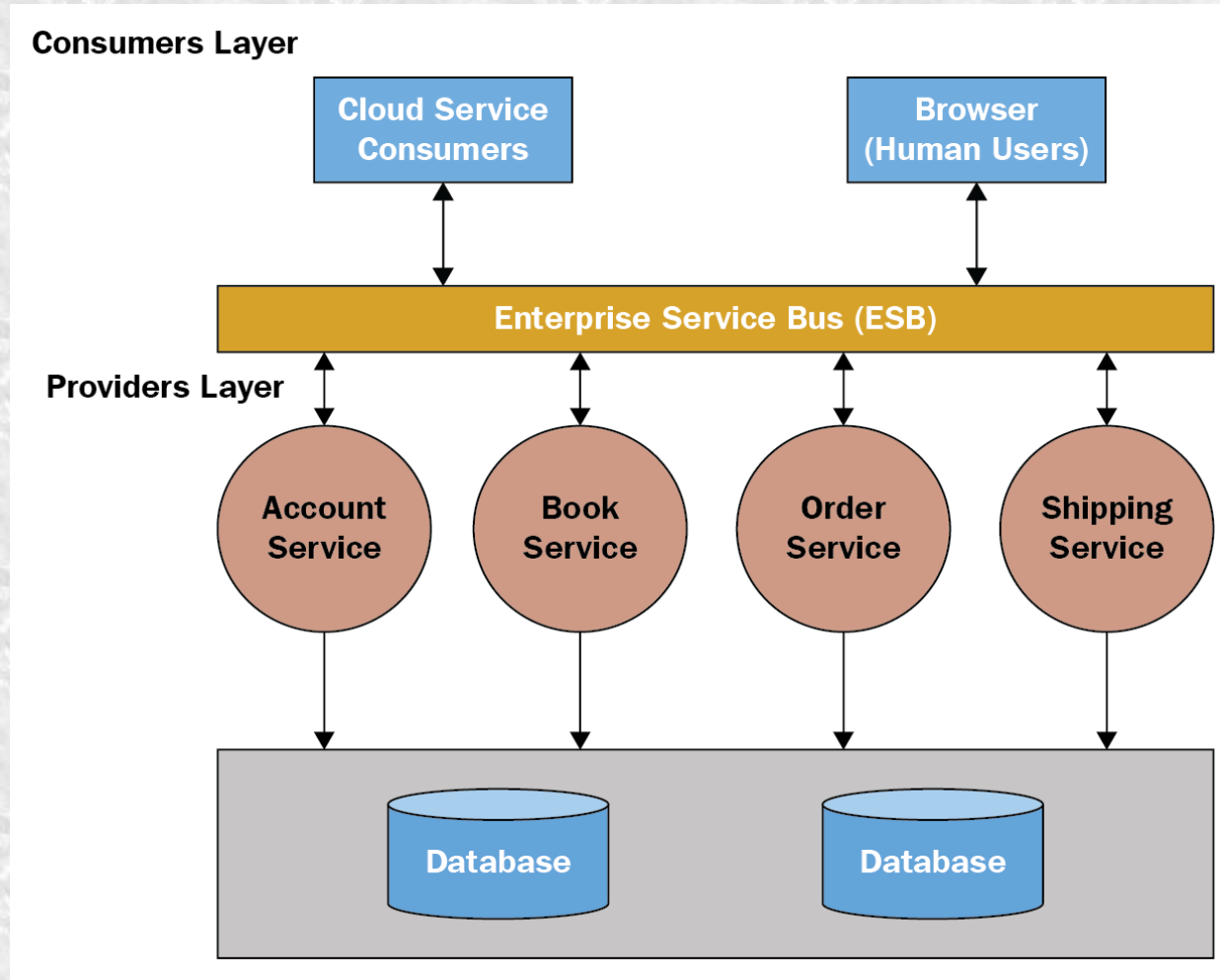
Service based applications

Monolithic vs. SOA vs. Microservices

- Monolithic:
 - Single Unit
 - Tightly coupled
- Service Oriented Architecture:
 - Coarse-grained
 - Loosely coupled
- Microservices:
 - Fine-grained
 - Loosely coupled

Service based applications

SOA architecture



Service based applications

SOA properties

- A service has four properties:
 - Logically represents a business activity with a specified outcome
 - Autonomous
 - Black box for its consumers
 - May consist of other underlying services

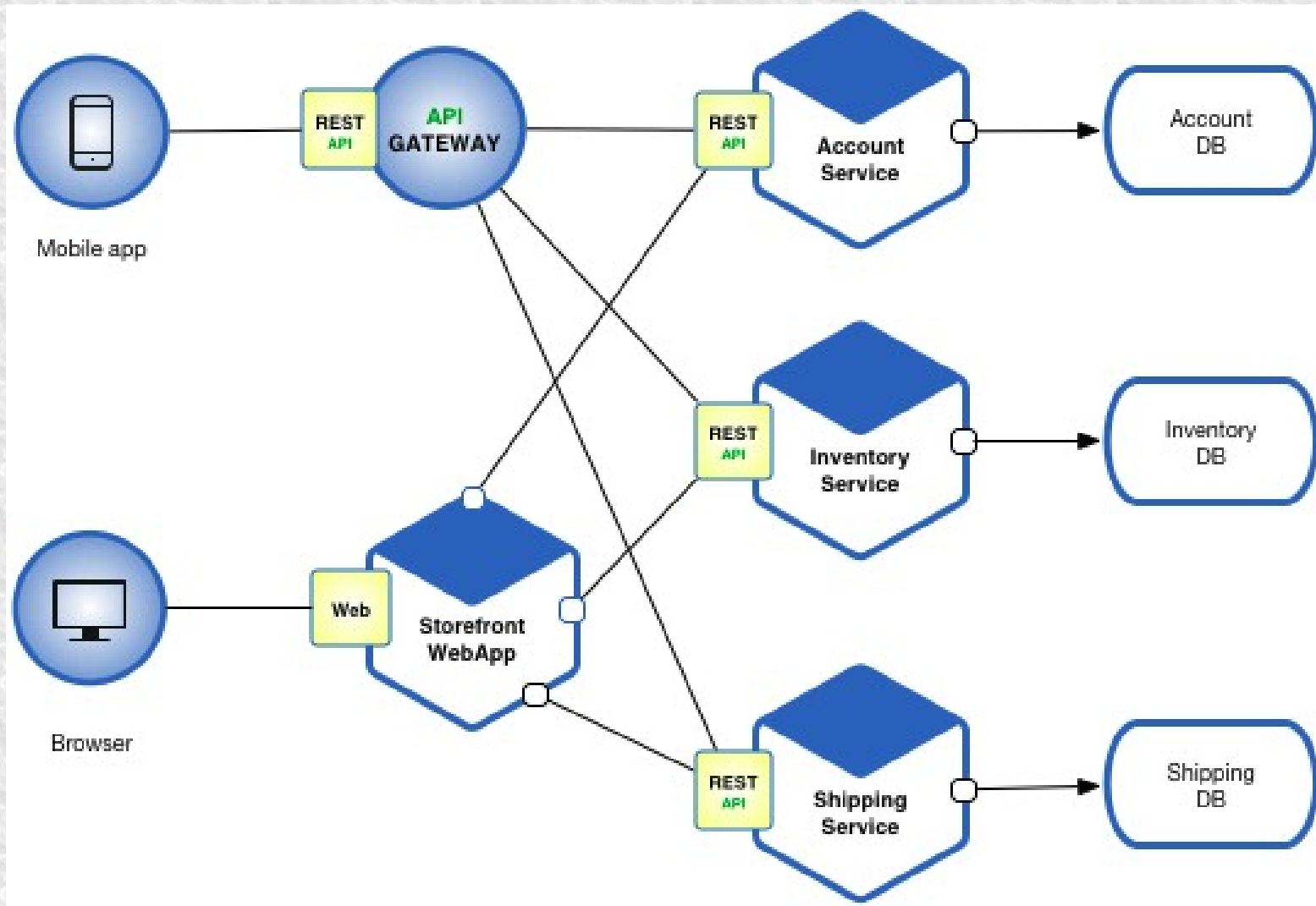
Service based applications

What are Microservices?

- Microservices architecture is the opposite of monolithic architecture
- Microservices are small & loosely coupled
- Should be fine-grained
- Protocols should be lightweight

Service based applications

Microservices architecture



Service based applications

What does microservices look like?

- Separate codebase
- Built independently
- Deployed independently
- Independant Running process
- Scaled independently

Service based applications

Why use Microservices?

- Modularity
- Flexibility : take advantages of different technologies
- Scalability
- Maintainability
- Suited Cloud-native deployment
- Small & autonomous teams
- Enable continuous integration and delivery

Service based applications

Choosing application architecture

- Monolithic architecture :
 - new product with limited resources and programming talent.
- Microservices architecture :
 - Run multiple copies of the application on multiple machines in order to satisfy scalability and availability requirements
 - Take advantage of emerging technologies (frameworks, programming languages, etc).
 - Support a variety of different clients : desktop browsers, mobile browsers, ...
 - Integrate with other applications
 - Exchanging messages with other systems; and returning a HTML/JSON/XML response.

RESTful APIs

What is REST ?

- REpresentational State Transfer
- Separation of client and server
- Statelessness
- Communication between clients and servers

RESTful APIs

Requests and Responses

- REST requires that a client make a request to the server
- Send a Request:
 - HTTP verb
 - Header
 - Resource path
 - Message Body (optional)
- Get a Response:
 - Content Type
 - Response Code

RESTful APIs

HTTP verbs

- 4 basic HTTP verbs:
 - GET - reads data and doesn't change application state
 - POST - creates resources
 - PUT - updates resources
 - DELETE - removes resources

RESTful APIs

Headers

- Provide information about the request or response, such as :
 - Content-Type
 - Accept
 - Authentication token or credentials for the request
 - User-Agent
 - ...

RESTful APIs

Paths

- Requests must contain a path to a resource
- Path should be the plural:
 - /customers
- Append an id to the path when accessing a single resource:
 - /customers/:id
 - /customers/:id/orders/:id

RESTful APIs

Status codes

- 200: OK: This is a successful request.
- 201: Created: A resource has been created.
- 202: Accepted: The request has been accepted but it hasn't been completed.
- 204: No Content: Successful HTTP requests, where nothing is being returned in the response body.
- 400: Bad Request: The request wasn't understood by the server, due to malformed syntax.
- 401: Unauthorized: Either the authentication header is missing, or it contains invalid credentials.
- 403: Forbidden: The client does not have permission to access this resource.

RESTful APIs

Status codes (Cont.)

- 404: Not Found: A resource matching the request doesn't exist.
- 405: Method Not Allowed: The requested operation is not supported on the specified Artifact type by the Services API.
- 500: Internal Server Error: An unhandled exception occurred on the server.
- 502 : Bad Gateway : The server was acting as a gateway or proxy and received an invalid response from the upstream server

RESTful APIs

Verbs and status codes

- GET: return 200 (OK)
- POST: return 201 (CREATED)
- PUT: return 200 (OK)
- DELETE: return 204 (NO CONTENT)

RESTful APIs

REST example

Request:

GET /customers/123

Accept: application/json

Response:

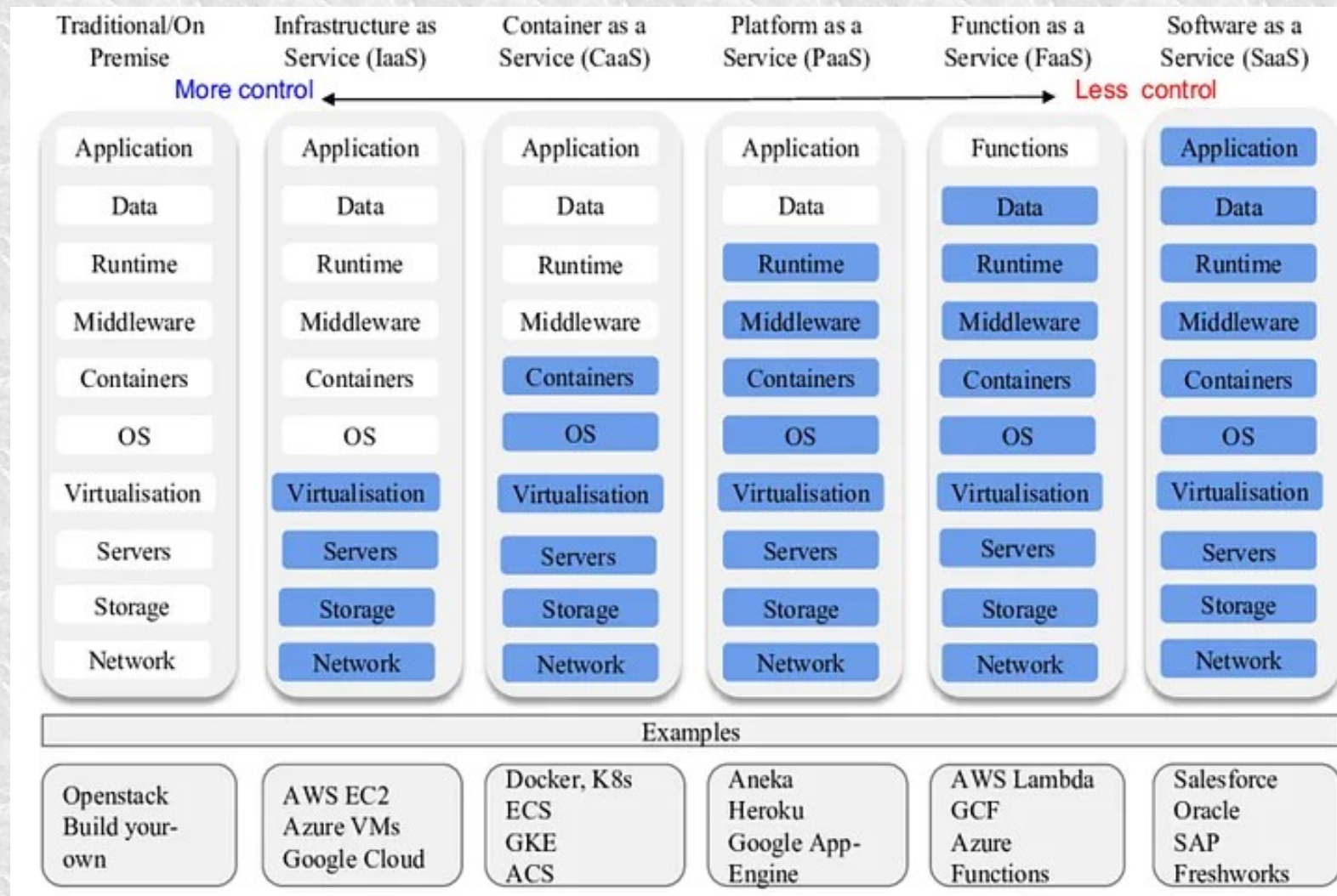
Status Code: 200 (OK)

Content-type: application/json

```
{  
  "customer": {  
    "id": 123,  
    "first_name": "Brahim",  
    "last_name": "Hamdi",  
    "email": "brahim.hamdi.consult@gmail.com"  
  }  
}
```

IaaS, CaaS & PaaS

Cloud services



LPI DevOps Tools Engineers

Module 2

Source code management

Plan

- SCM solutions
- Git concepts and repository structure
- Git data transport commands
- Other Git commands
- Git merge conflicts

SCM solutions

Source Code Management

- SCM – Source Code Management
- Software tools SCM are sometimes referred to as :
 - "Source Code Management Systems" (SCMS)
 - "Version Control Systems" (VCS)
 - "Revision Control Systems" (RCS)
 - or simply "code repositories"
- Tracking modifications assists development and collaboration by :
 - Providing a running history of development
 - helping to resolve conflicts when merging contributions from multiple sources.

SCM solutions

SCM types

- Centralized version control :
 - 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
 - Solutions : CVS, SVN (Subversion)
- Distributed version control
 - Version control is mirrored on every user's computer.
 - Allows branching and merging to be managed automatically.
 - Ability to work offline (Allows users to work productively when not connected to a network)
 - Solutions : Git, Mercurial.

Git concepts and repository structure

What is Git ?

- Git is a distributed SCM system.
- Initially designed and developed by Linus Torvalds for Linux kernel development.
- A free software distributed under GNU General Public License version 2.
- Advantages :
 - Free and open source
 - Fast and small
 - Implicit backup
 - Secure : uses SHA1 to name and identify objects.
 - Easier branching : copy all the codes to new branch.

Git concepts and repository structure

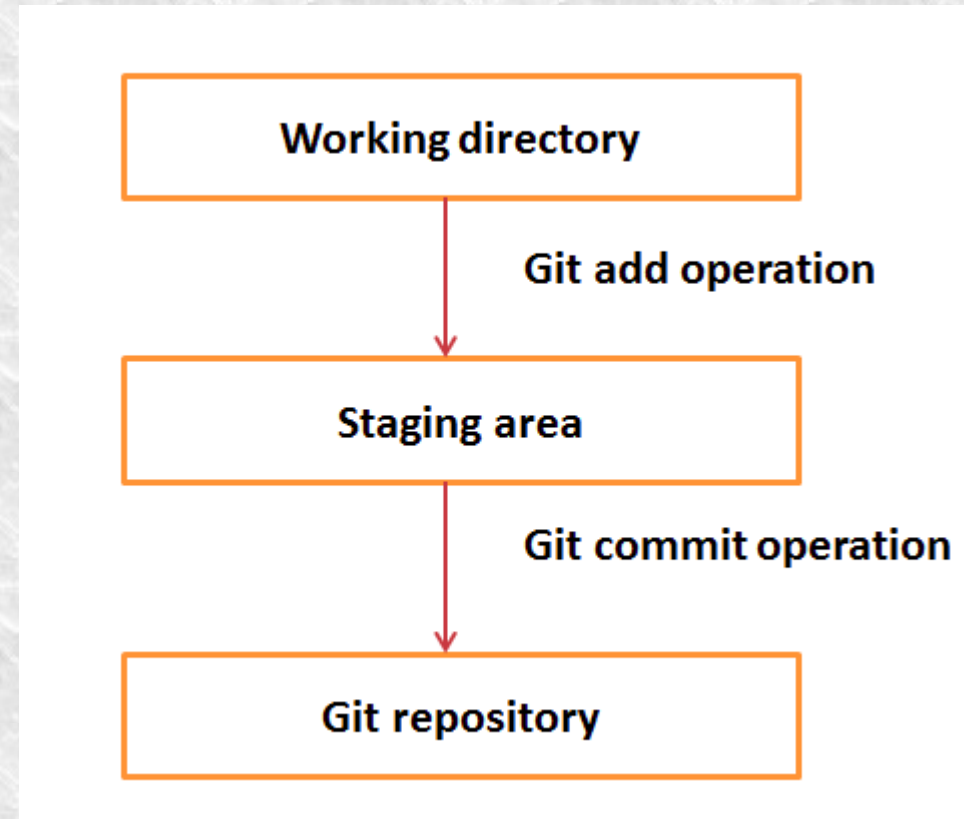
Git repository

- Local Repository : Typically is on user's computer.
 - User make changes in his private workplace
 - after commit, these changes become a part of a local repository.
 - Users can perform many operations with this repository
 - add file
 - remove file
 - rename file
 - move file
 - commit changes
 - and many more...

Git concepts and repository structure

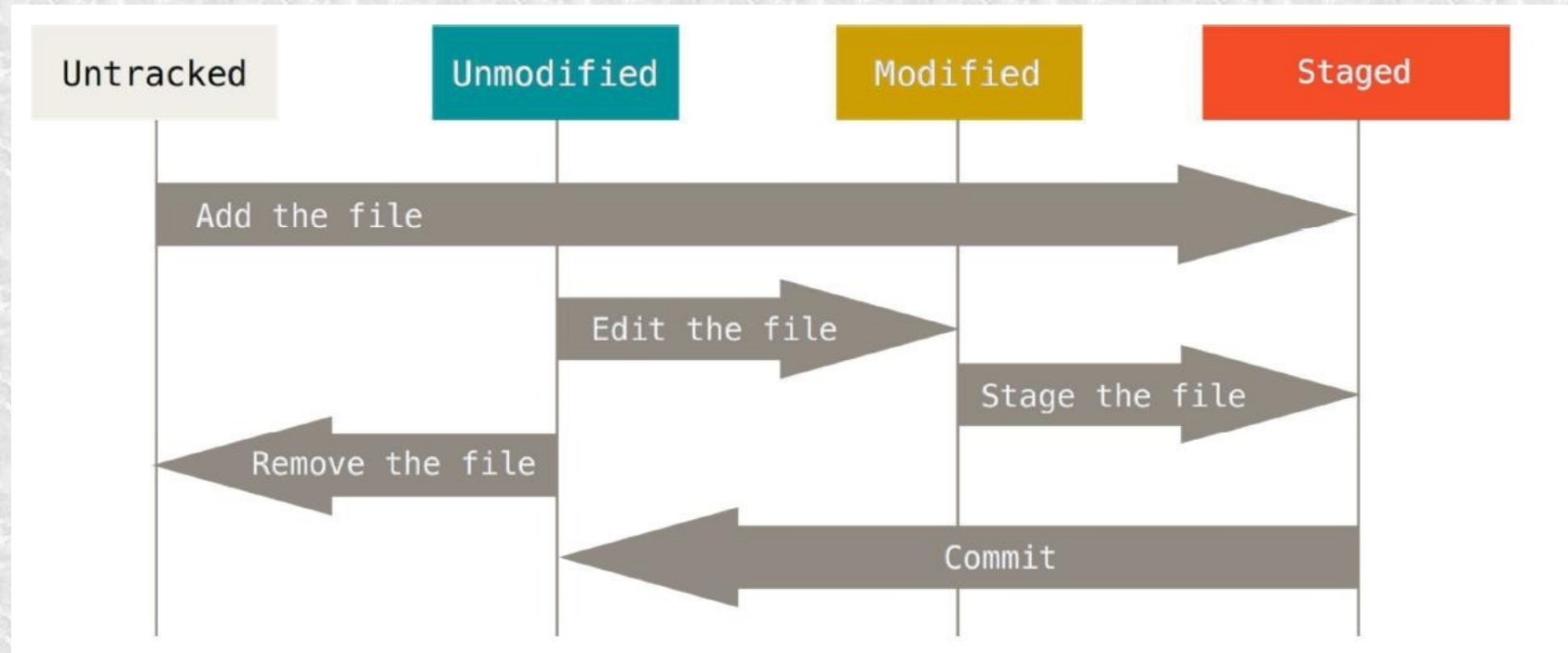
Working Directory and Staging Area or Index

- Basic workflow of Git.
 - Step 1 : modify a file from the working directory.
 - Step 2 : add these files to the staging area.
 - Step 3 : perform commit operation that moves the files from the staging area.



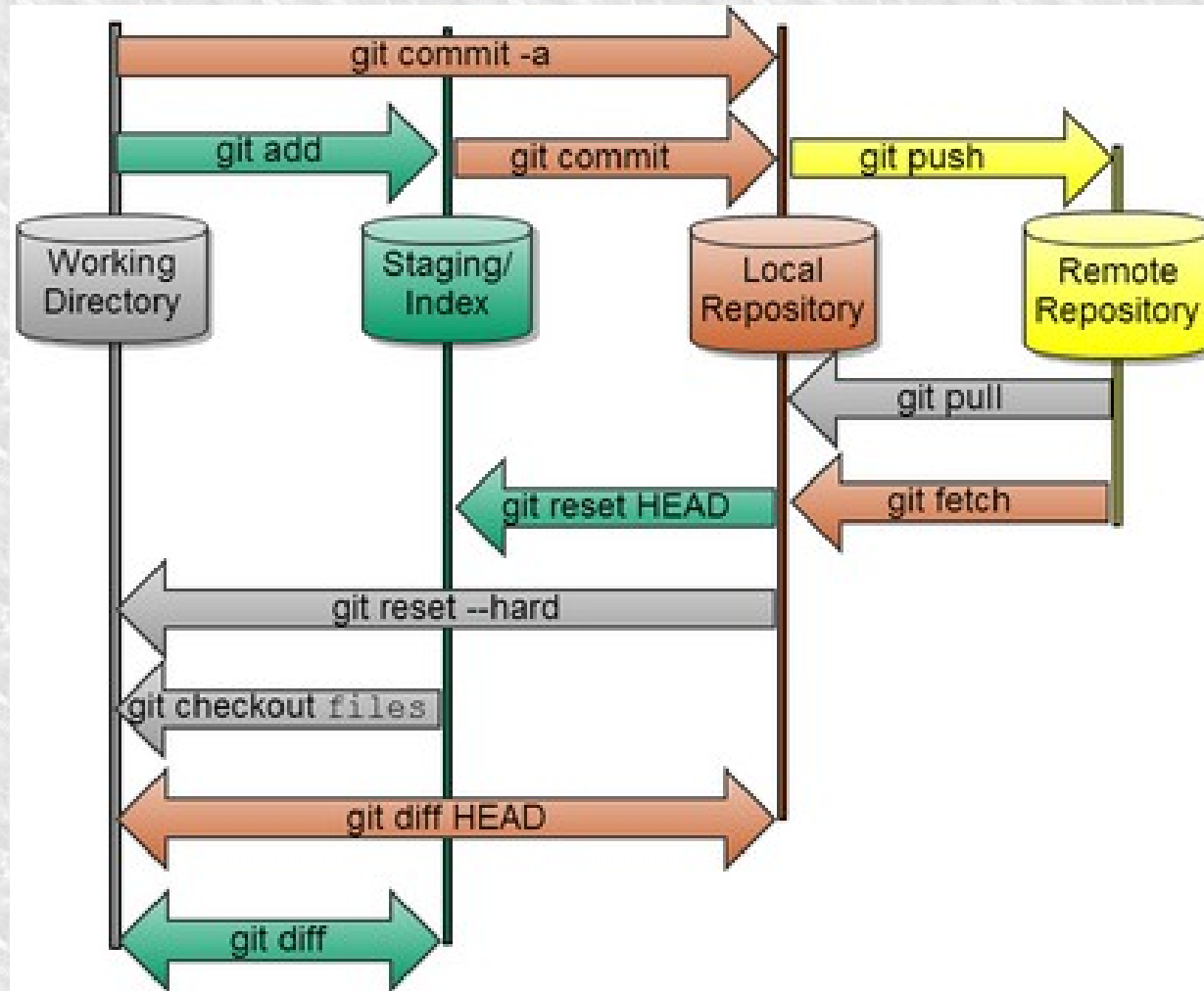
Git concepts and repository structure

git status



Git commands

main commands



Git merge conflict

Understanding merge conflicts

- Conflicts generally arise when :
 - Two peoples have changed the same lines in a file
 - If one user deleted a file while another developer was modifying it.
- Git cannot automatically determine what is correct.
- Conflicts only affect the user conducting the merge, the rest of the team is unaware of the conflict.
- Git will mark the file as being conflicted and halt the merging process.
- It is then the user's responsibility to resolve the conflict.

LPI DevOps Tools Engineers

Module 3

System image creation and VM Deployment

Plan

- Vagrant
- Vagrantfile
- Vagrantbox
- Packer

Vagrant

What's vagrant

- Create and configure lightweight, reproducible, and portable development environments.
- A higher-level wrapper around virtualization software such as VirtualBox, VMware, KVM.
- Wrapper around configuration management software such as Ansible, Chef, Salt, and Puppet.
- Public clouds e.g. AWS, DigitalOcean can be providers too.

Vagrantbox contents

- A tarred, gzip file containing the following:
 - VM artifacts (required) : VM image and other artifacts in the format accepted by the provider the box is intended for.
 - For example, a box targeting the VirtualBox provider might have a *.ovf* file and some *.vmdk* files.
 - metadata.json (required) : Contains information about the box
 - info.json : JSON document that can provide additional information about the box that displays when a user runs *vagrant box list -i*
 - Vagrantfile : The Vagrantfile embedded in the Vagrant box will provide some defaults for users of the box

Vagrantbox commands

```
devops@lpic:/sauvegarde2/vagrant_VM/centos7$ vagrant box
```

```
Usage: vagrant box <subcommand> [<args>]
```

```
Available subcommands:
```

```
add  
list  
outdated  
prune  
remove  
repackage  
update
```

```
For help on any individual subcommand run `vagrant box <subcommand> -h`
```

```
devops@lpic:/sauvegarde2/vagrant_VM/centos7$
```

Vagrantbox

Tools to create vagrantbox

- Use tools like packer.io, imagefactory etc.
- Build a Vagrantbox manually
 - Or use “vagrant package” command for Virtualbox.
- Modify base boxes and reuse them

Packer

What is Packer

- Open source tool for creating identical machine images :
 - for multiple platforms
 - from a single source configuration.
- Advantages of using Packer :
 - Fast infrastructure deployment
 - Multi-provider portability
 - Stability
 - Identity

Packer

Commands

```
devops@lpic:/sauvegarde2/vagrant_VM/centos7$ packer
usage: packer [--version] [--help] <command> [<args>]
```

Available commands are:

| | |
|----------|--|
| build | build image(s) from template |
| fix | fixes templates from old versions of packer |
| inspect | see components of a template |
| push | push a template and supporting files to a Packer build service |
| validate | check that a template is valid |
| version | Prints the Packer version |

```
devops@lpic:/sauvegarde2/vagrant_VM/centos7$ █
```


Packer

Templates

- The JSON configuration files used to define/describe images.
- Templates are divided into core sections:
 - variables (optional)
 - builders (required)
 - provisioners (optional)
 - post-processors (optional)

Packer Builders

- Responsible for creating machines and generating images for various platforms.
- Popular supported builders by Packer :
 - Amazon EC2
 - Azure
 - Google Cloud
 - OpenStack
 - VirtualBox
 - Docker
 - Hyper-V

Packer

Provisioners

- Responsible for preparing and configuring the operating system.
- Popular supported provisioners by Packer:
 - Ansible
 - Puppet
 - Chef
 - Salt
 - Shell
 - PowerShell

LPI DevOps Tools Engineers

Module 4

Container usage

Plan

- What is a Container and Why?
- Docker and containers
- Docker command line
- Connect container to Docker networks
- Manage container storage with volumes
- Create Dockerfiles and build images

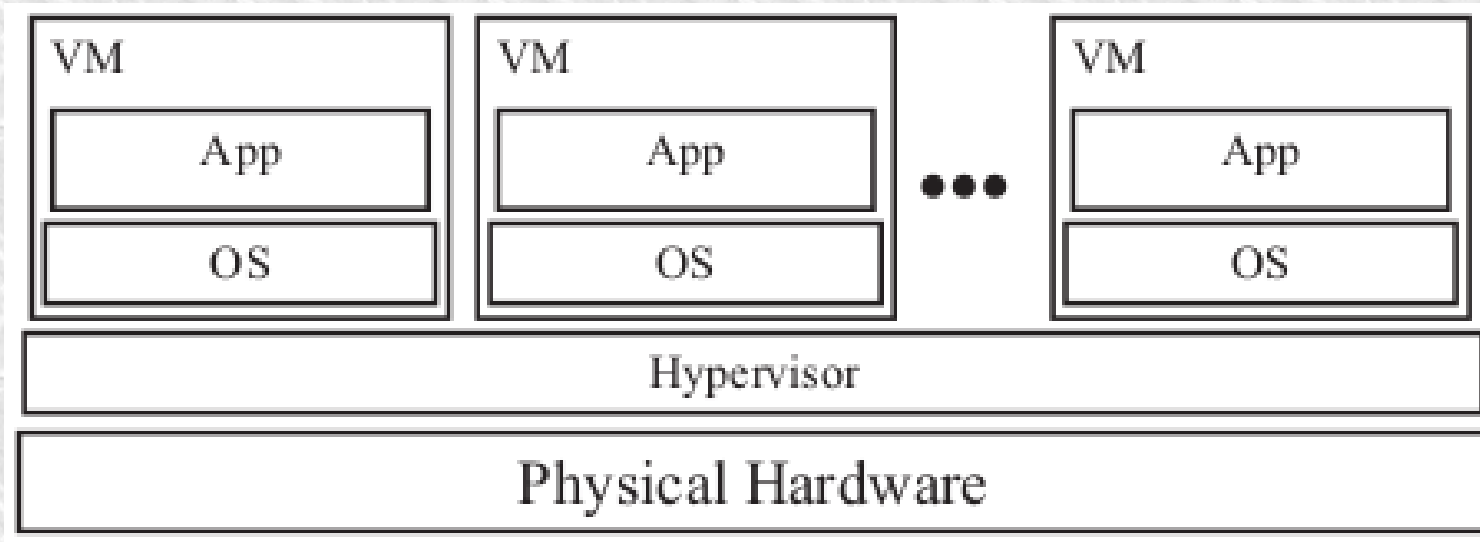
What is a Container and Why?

Advantages of Virtualization

- Minimize hardware costs.
- Multiple virtual servers on one physical hardware.
- Easily move VMs to other data centers.
- Conserve power
- Free up unused physical resources.
- Easier automation.
- Simplified provisioning/administration of hardware and software.
- Scalability and Flexibility: Multiple operating systems

What is a Container and Why?

Virtualization problems



- Each VM requires an operating system (OS)
 - Each OS requires a licence
 - Each OS has its own compute and storage overhead
 - Needs maintenance, updates, ...

What is a Container and Why?

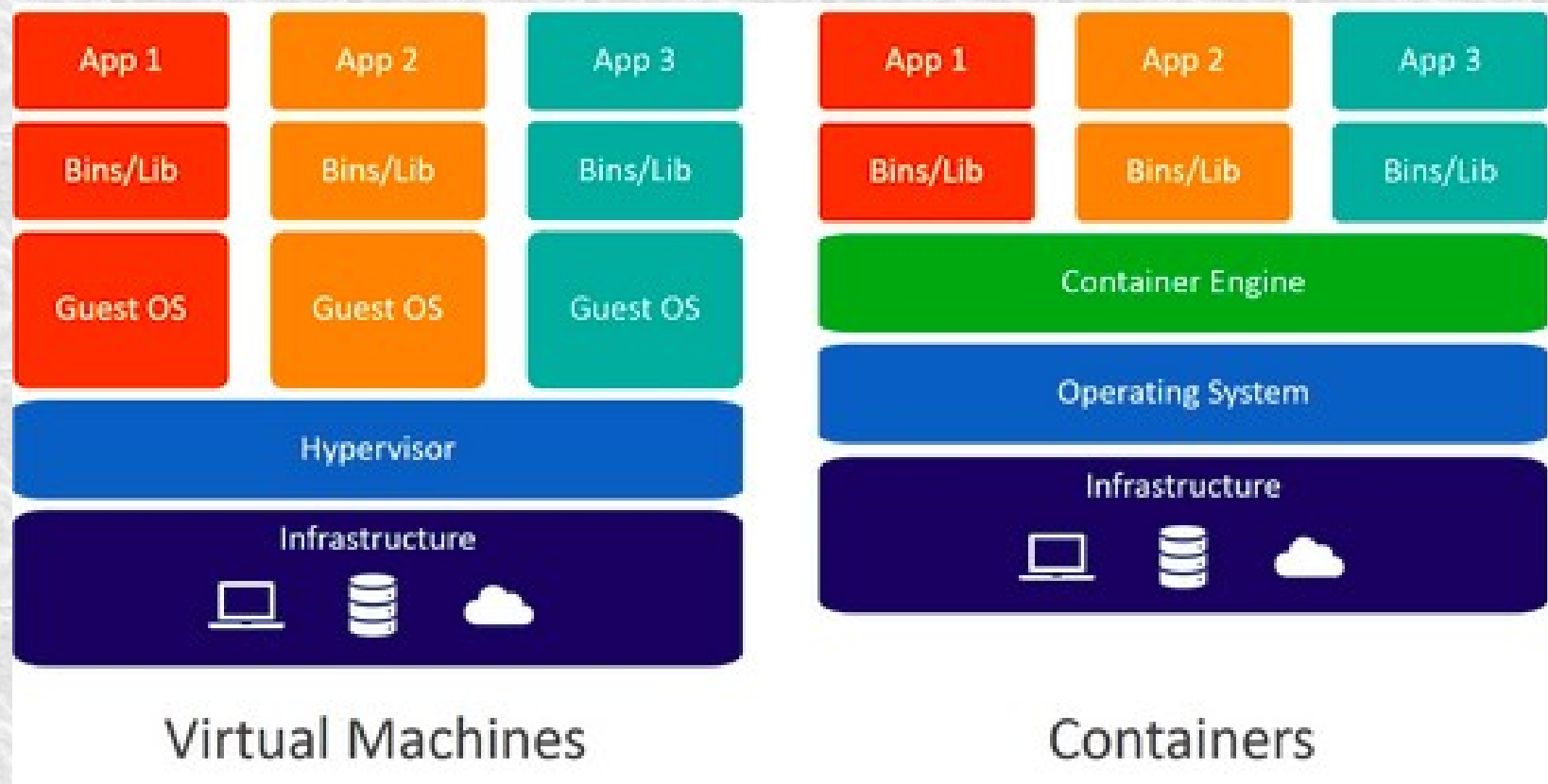
Solution: Containers

- Run many apps on the same physical/virtual machine
 - These apps share the OS (kernel) and its overhead
 - But these apps can't interfere with each other
 - Can't access each other's resources without explicit permission.
 - Like apartments in a complex

⇒ **Containers**

What is a Container and Why?

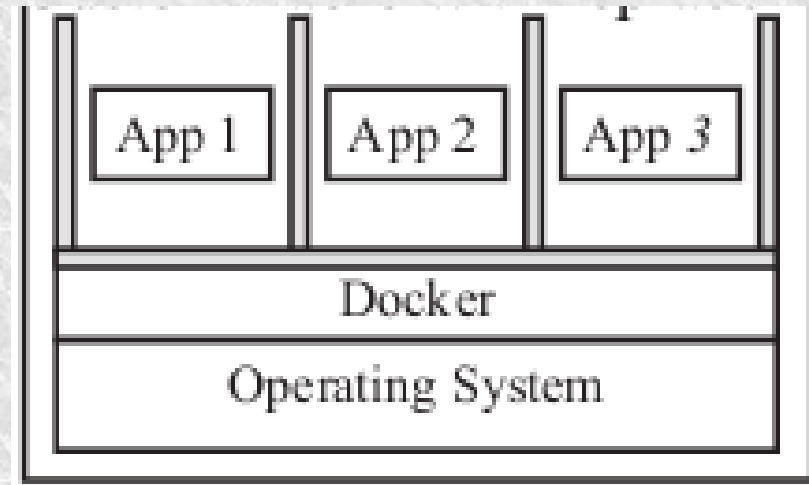
VM vs Containers



Docker and containers

Docker

- Provides the isolation among containers
- Helps them share the OS
- Docker = Dock worker \Rightarrow Manage containers
- Developed initially by Docker.com
- Downloadable for Linux, Windows, and Mac from Docker.com



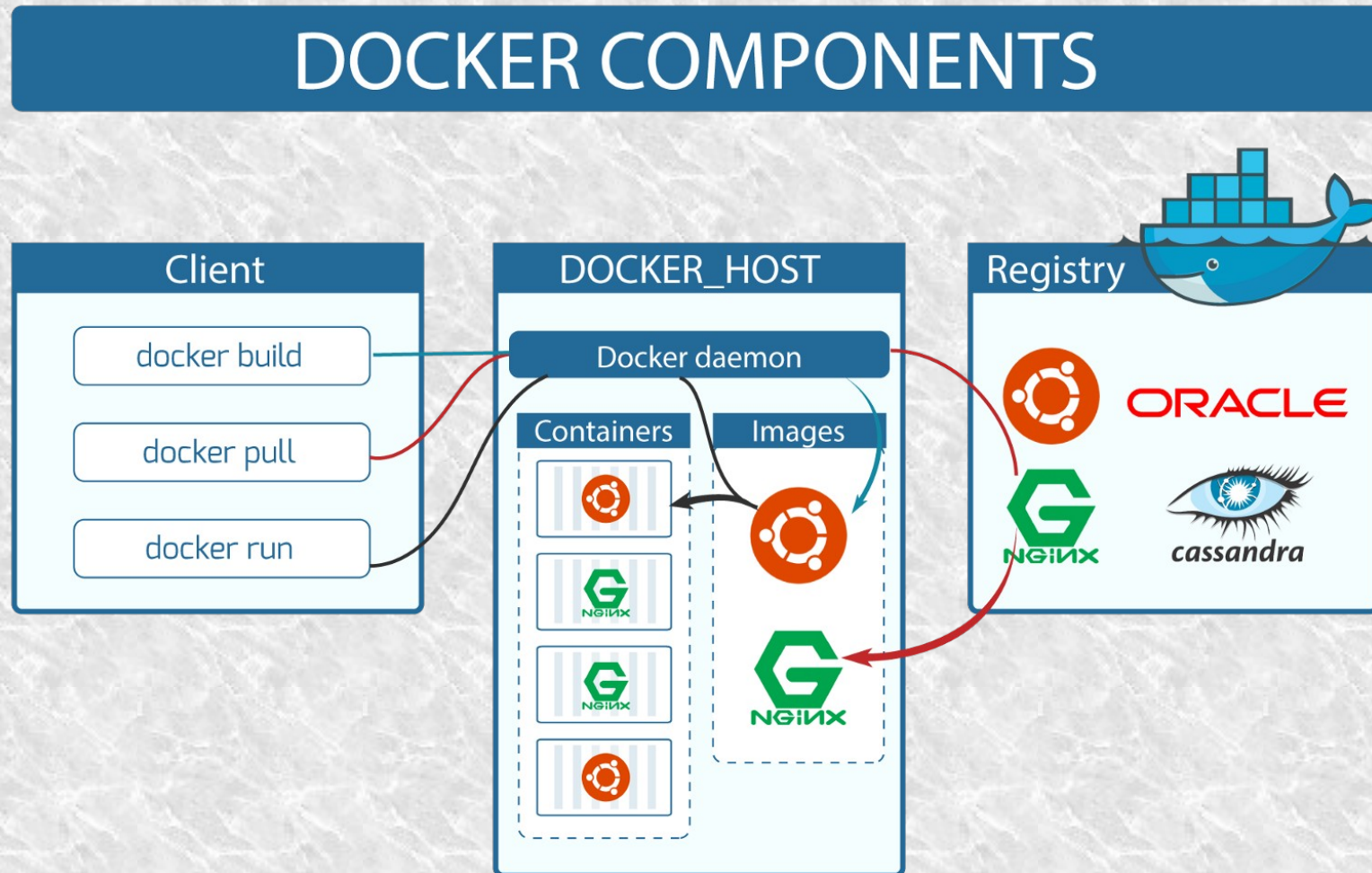
Docker and containers

Docker

- Docker Engine: Runtime.
- Two Editions:
 - Community Edition (CE): Free for experimentation.
 - Enterprise Edition (EE): For deployment with paid support.
- Written in “Go” programming language from Google.

Docker and containers

Docker Components



Docker and containers

Image Registries

- Containers are built from images and can be saved as images
- Images are stored in registries
 - Local registry on the same host
 - Docker Hub Registry: Globally shared
 - Private registry
- Any component not found in the local registry is downloaded from specified location.
- Three image types:
 - Official images vetted by Docker
 - Unofficial images verified by Docker
 - Unofficial images not verified by Docker (Use with care)
- Each image has several tags, e.g., v2, latest, ...
- Each image is identified by its 256-bit hash

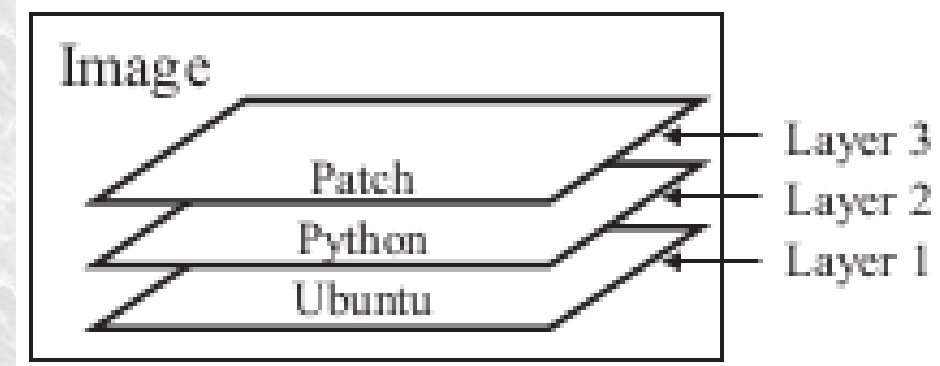
Docker and containers

Image layers

- Each image has many layers
- Image is built layer by layer
- Layers in an image can be inspected by Docker commands
- Each layer has its own 256-bit hash

- For example:

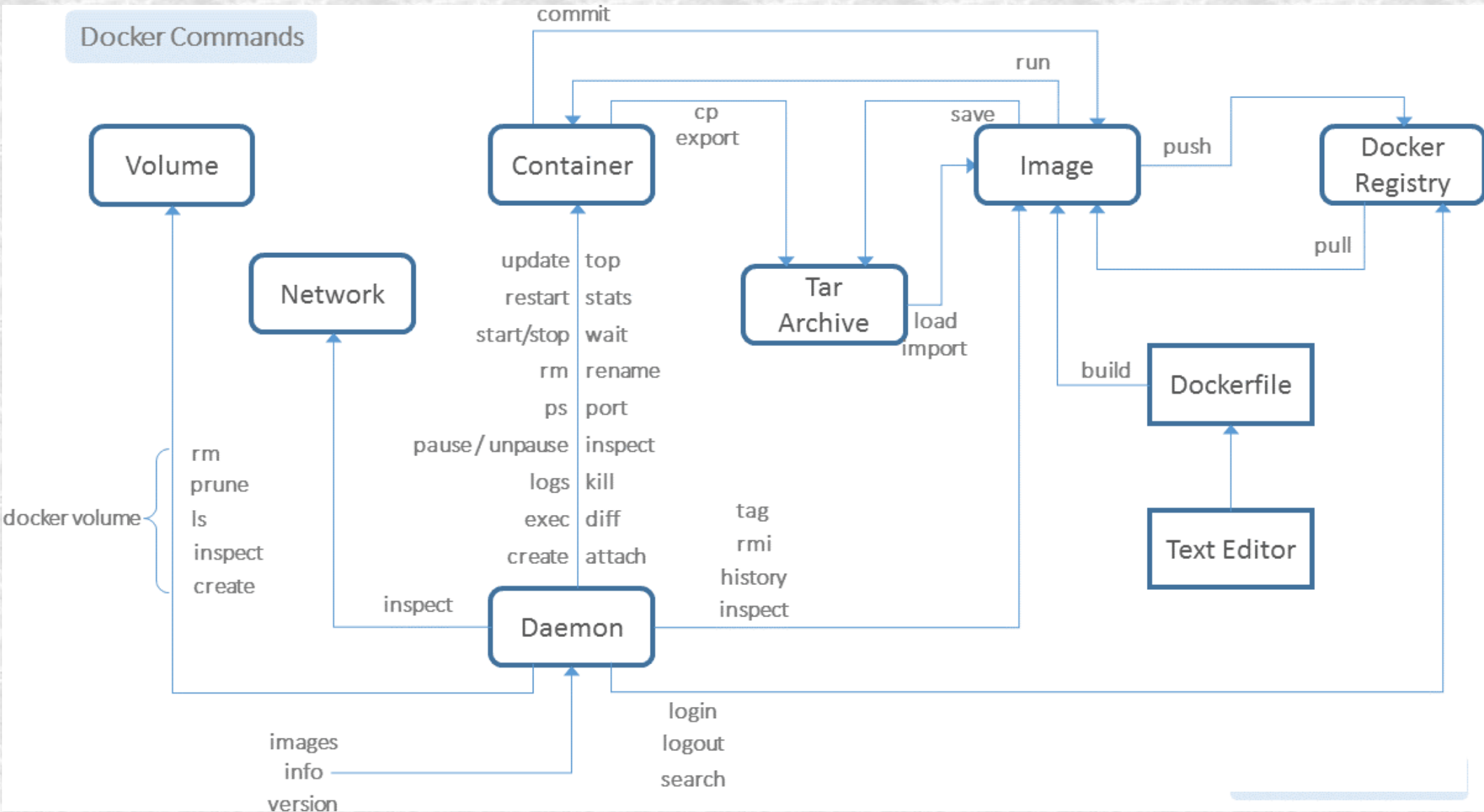
- Ubuntu OS is installed, then
- Python package is installed, then
- a security patch to the Python is installed



- Layers can be shared among many containers

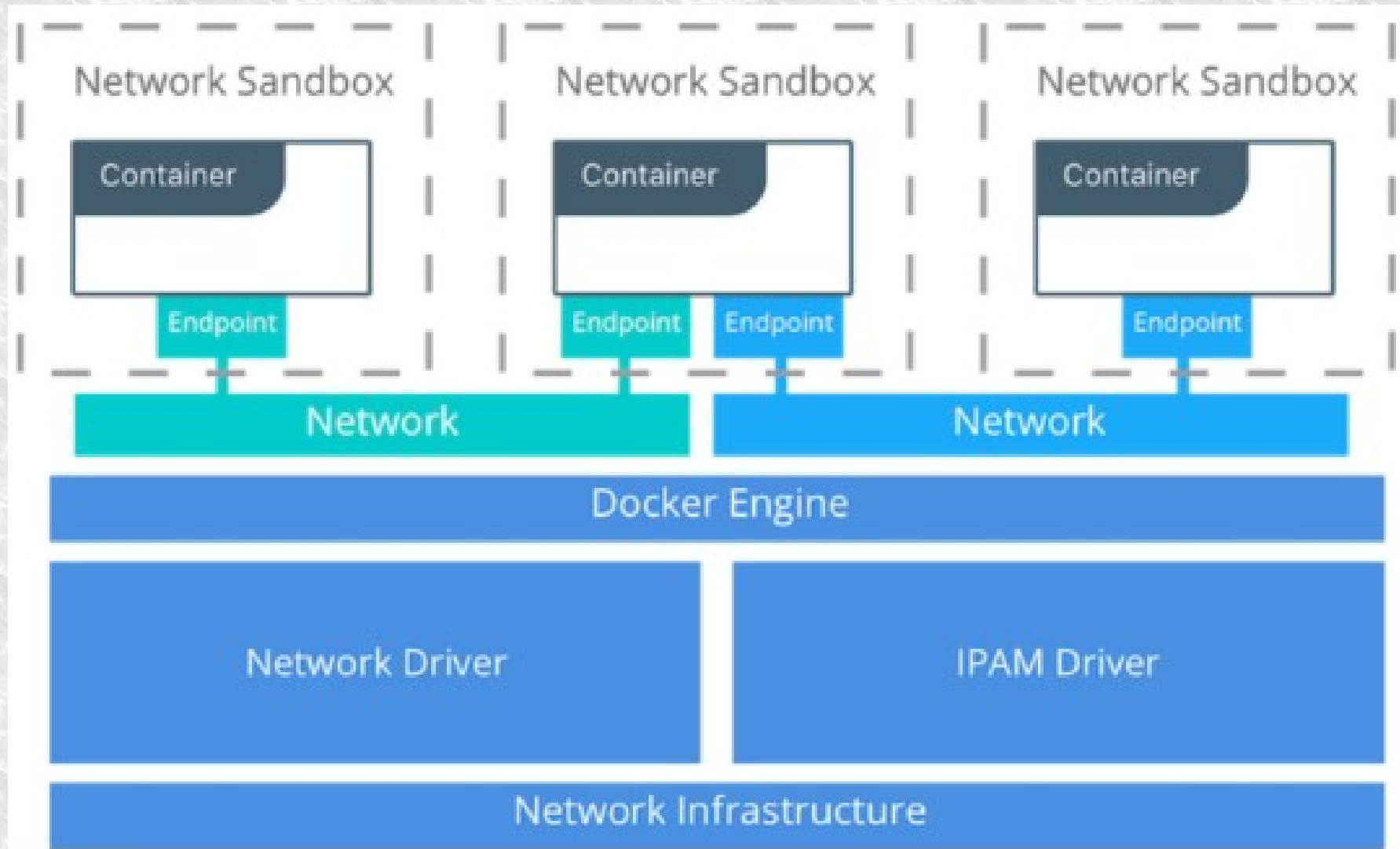
Docker command line

Available commands



Connect container to Docker networks

Container Networking Model (CNM)



Connect container to Docker networks

Container Networking Model (CNM)

- A standard proposed by Docker.
 - There is also CNI : container networking standard proposed by CoreOS.
- Designed to support the Docker runtime engine only.
- Sandbox : contains the configuration of a container's network stack. This includes
 - management of the container's interfaces
 - routing table
 - DNS settings.
- Endpoint: enable connection to the outside world, from a simple bridge to a complex overlay network
- Network driver: possibility to use Docker solution or third party
- IPAM : IP address management - DHCP and the like.

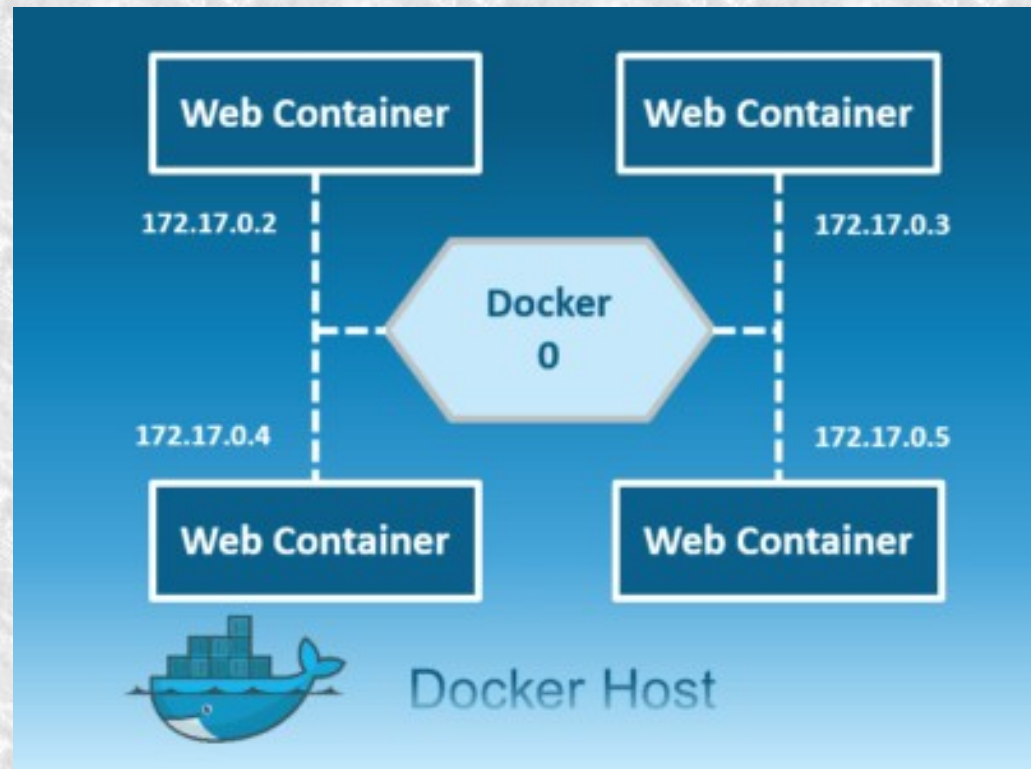
Connect container to Docker networks

Network drivers

- To list all docker networks
docker network ls
- 5 network drivers:
 - bridge: The default network driver – scope local
 - host: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly.
 - overlay: Connect multiple Docker daemons together and enable swarm services to communicate with each other – scope swarm
 - macvlan: Allow to assign a MAC address to a container, making it appear as a physical device on network
 - none: Disable all networking. Usually used in conjunction with a custom network driver.
- 3 pre-defined networks (cannot be removed) : bridge, host and none

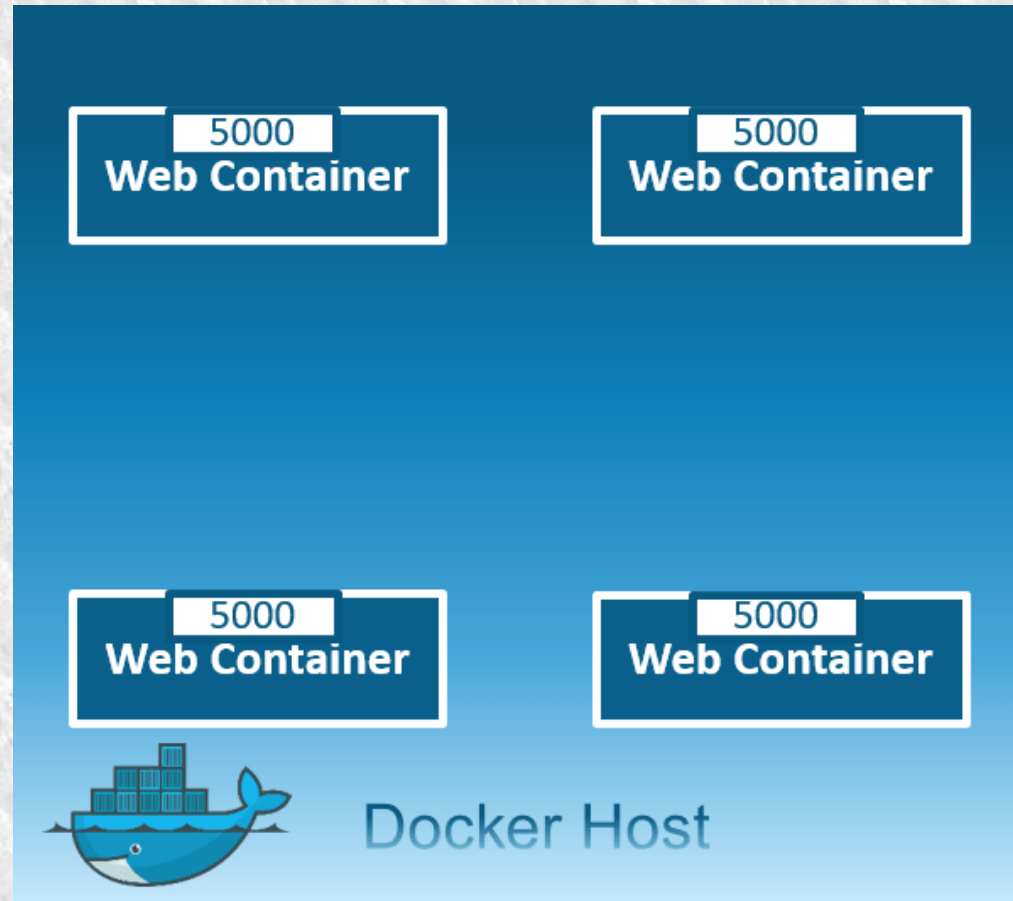
Connect container to Docker networks

Default bridge network



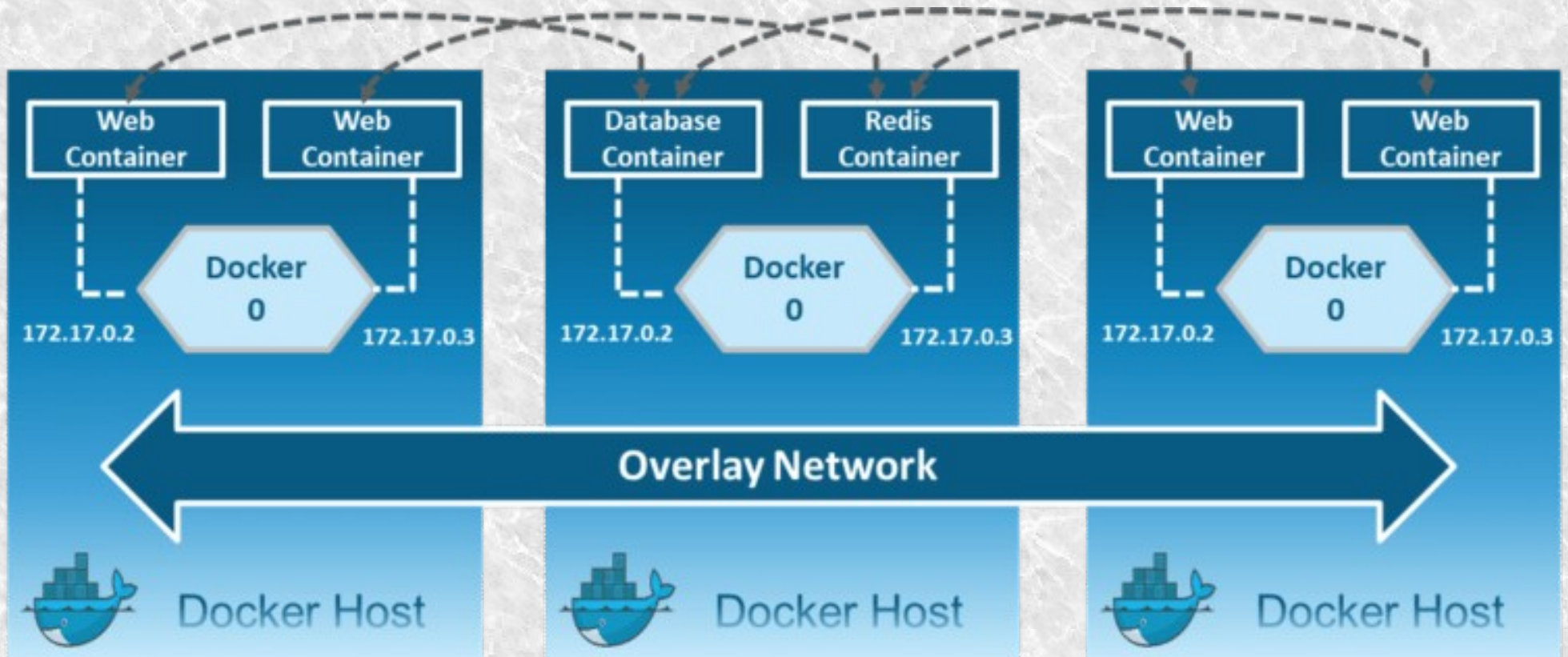
Connect container to Docker networks

Host network



Connect container to Docker networks

Overlay network



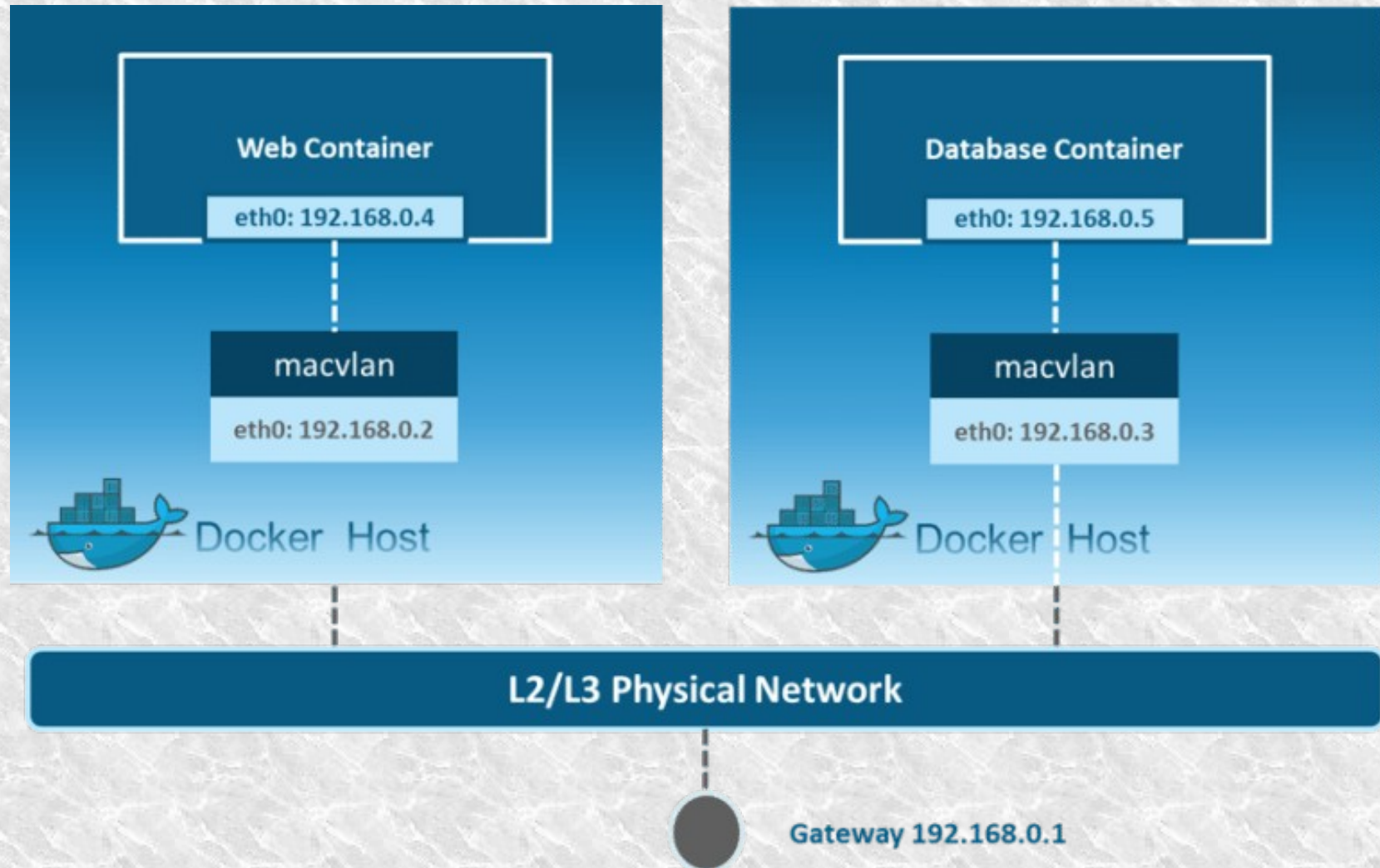
Connect container to Docker networks

overlay networks

- When initialize a swarm, two new networks are created on that Docker host:
 - an overlay network called ***ingress***, which handles control and data traffic related to swarm services.
 - a bridge network called ***docker_gwbridge***, which allows the containers to connect to the host that it is running on.
- You can create user-defined overlay networks using the command :
 - *docker network create -d overlay my-overlay*

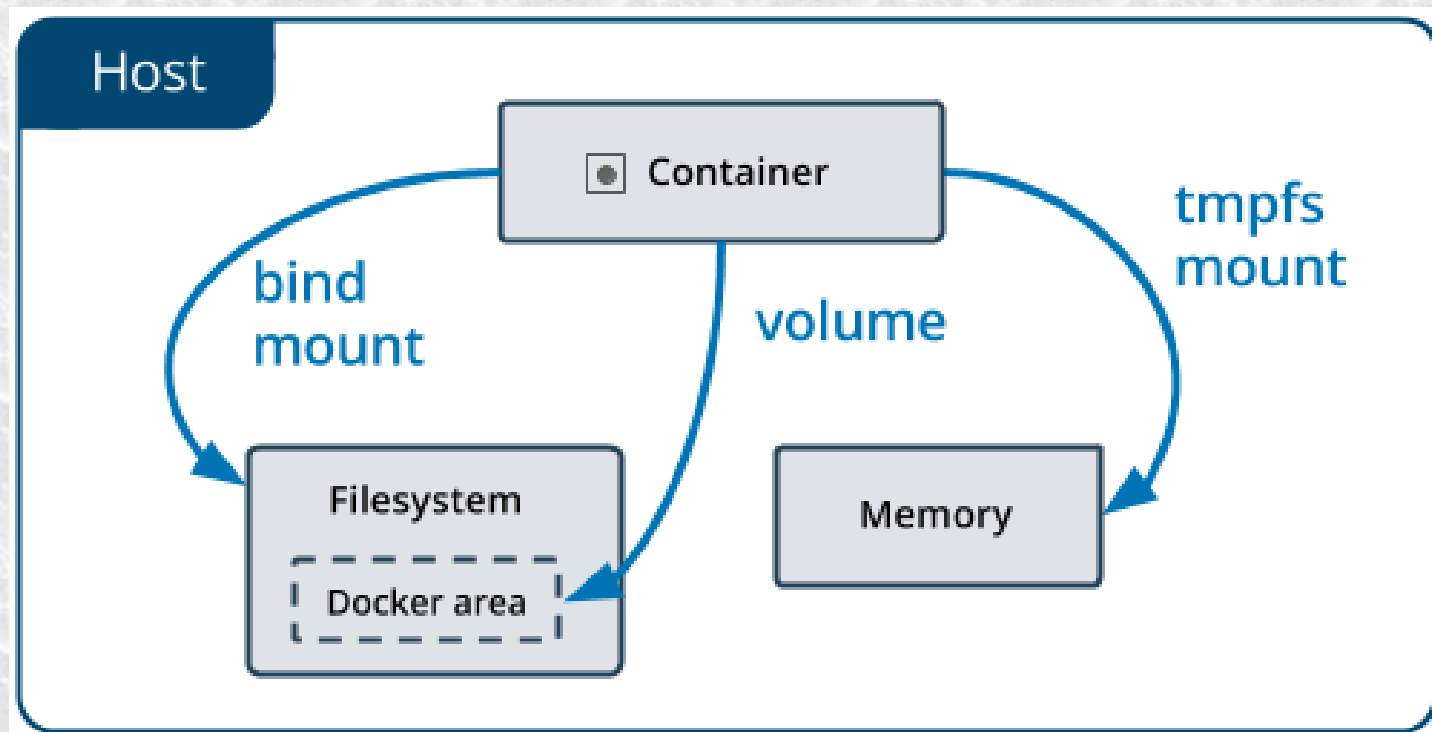
Connect container to Docker networks

Macvlan network



Manage container storage with volumes

Docker storage mechanisms



Manage container storage with volumes

Manage volumes

- Some use cases for volumes :
 - Sharing data among multiple running containers.
 - Store your container's data on a remote host or a cloud provider, rather than locally.
 - Back up, restore, or migrate data from one Docker host to another.

Create Dockerfiles and build images

dockerfile

- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- Docker can build images automatically by reading the instructions from a Dockerfile.
 - *docker build path .*
- Before the docker sends the context to the docker daemon, it looks for a file named « .dockerignore » in the root directory of the context. If this file exists, the CLI modifies the context to exclude files and directories that match patterns in it.
- The format of the Dockerfile:

Comment

INSTRUCTION arguments

Create Dockerfiles and build images

FROM, RUN instructions

- A Dockerfile must start with a `FROM` instruction
- FROM instruction specifies the Base Image from which you are building.
- The RUN instruction will execute any commands in a new layer on top of the current image and commit the results.
- RUN has 2 forms:
 - RUN <command> (shell form, the command is run in a shell, which by default is /bin/sh -c on Linux or cmd /S /C on Windows)
 - RUN ["executable", "param1", "param2"] (exec form)

Create Dockerfiles and build images

CMD & ENTRYPOINT instructions

- Provide defaults for an executing container.
- If you list more than one CMD/ENTRYPOINT then only the last will take effect.
- 2 forms:
 - Shell form
 - Exec form
- When running image CMD instruction can be overridden, but ENTRYPOINT no.
- CMD, ENTRYPOINT and other instructions are evaluated when a new container is created from an existing image built from the Dockerfile.

Create Dockerfiles and build images

COPY, ADD, WORKDIR instructions

- COPY : copies new files/directories from <src> to the filesystem of the container at the path <dest>.
 - files and directories will be interpreted as relative to the source of the context of the build.
- ADD : copies new files/directories or remote file URLs from <src> to the filesystem of the image at the path <dest>.
- WORKDIR: sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.

Create Dockerfiles and build images

MAINTAINER, EXPOSE, ENV instructions

- MAINTAINER : sets the Author field of the generated images.
- EXPOSE : informs Docker that the container listens on the specified network ports at runtime.
 - does not actually publish the port.
- ENV : sets the environment variable <key> to the value <value>.
 - This value will be in the environment for all subsequent instructions in the build stage.

Create Dockerfiles and build images

VOLUME instruction

- VOLUME : creates a mount point with the specified name and marks it as holding externally mounted.
- The *docker run* command initializes the newly created volume with any data that exists at the specified location within the base image.

LPI DevOps Tools Engineers

Module 5

Container Deployment and Orchestration

Plan

- Docker-compose
- Docker swarm
- Kubernetes

Docker-compose

What's docker-compose ?

- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a YAML file to configure your application's services.
- Then, with a single command, you create and start all the services from your configuration.
- Compose works in all environments: production, staging, development, testing, as well as CI workflows.

Docker-compose

Using compose

- Using Compose is basically a three-step process:
 - Define your app's environment with a Dockerfile so it can be reproduced anywhere.
 - Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.
 - Run *docker-compose up* and *Compose starts* and runs your entire app.

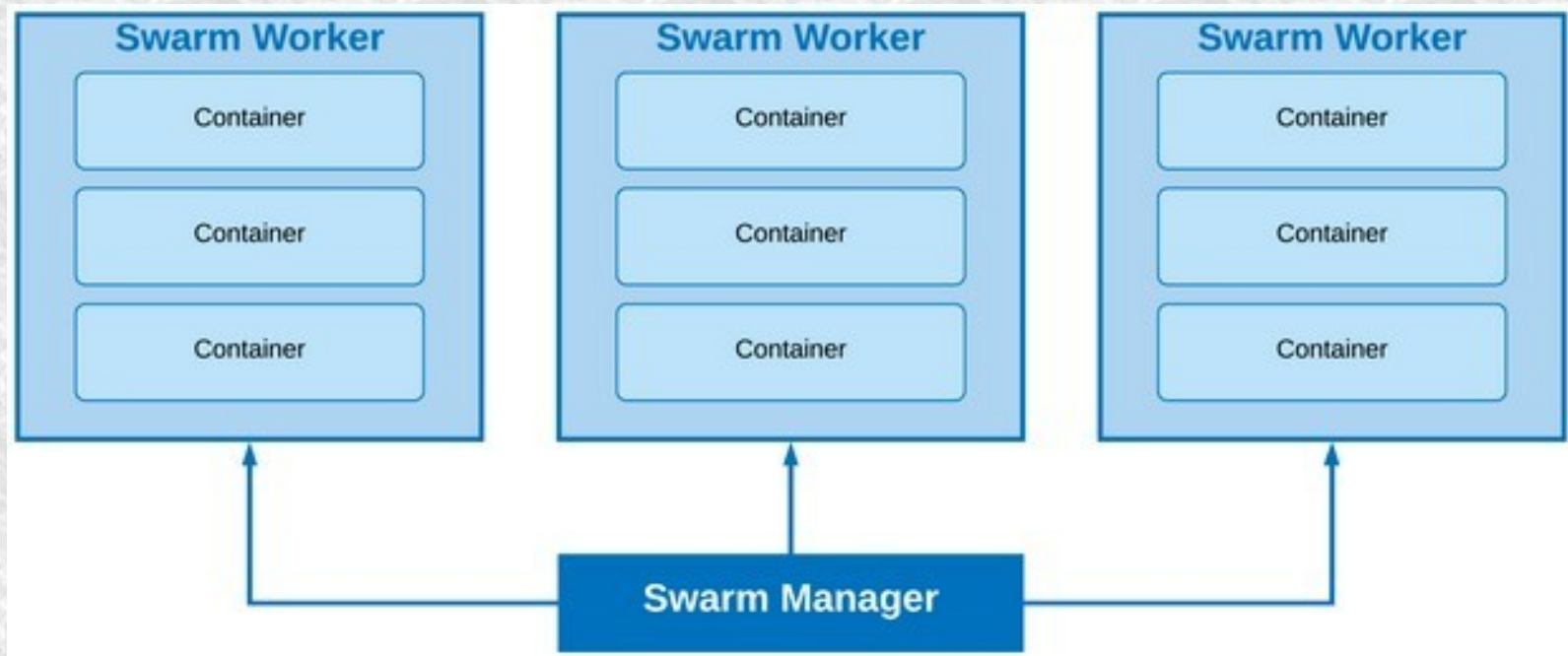
What's docker swarm mode

- A swarm consists of one or more nodes: physical or VM running Docker Engine 1.12 or later in swarm mode.
- Swarm mode refers to cluster management and orchestration features embedded in Docker Engine.
- When initialize a new swarm (cluster) or join nodes to a swarm, the Docker Engine runs in swarm mode.

Docker swarm

How nodes work

- There are two types of nodes: **managers** and **workers**.



Docker swarm

Swarm Services networks

- The following three networks concepts are important to swarm services:
 - **Overlay network** : manage communications among the Docker daemons participating in the swarm.
 - You can create overlay networks, in the same way as user-defined networks for standalone containers.
 - **Ingress network** : a special overlay network that facilitates load balancing among a service's nodes.
 - **Docker_gwbridge** : a bridge network that connects the overlay networks (including the ingress network) to an individual Docker daemon's physical network.
- The ingress network and docker_gwbridge network are created automatically when you initialize or join a swarm.

Docker swarm

Initialise a swarm

1. Make sure the Docker Engine daemon is started on all nodes.

2. On the manager node :

```
docker swarm init --advertise-addr <MANAGER-IP>
```

3. On each worker node :

```
docker swarm join --token \ <token_generated_by_manager> <MANAGER-IP>
```

4. On manager node, view information about nodes to verify :

```
docker node ls
```


Docker swarm

Deploy Swarm Services with Compose

- Docker Compose and Docker Swarm aim to have full integration ⇒ point a Compose app at a swarm cluster.
- 3 steps :

1. Initialise Swarm Mode

2. Create Docker Compose file

3. Deploy Services by using docker stack command :

```
docker stack deploy --compose-file docker-compose.yml myapp
```

- Details of the internal services can be discovered via :

```
docker stack services myapp
```

```
docker stack ps myapp
```

```
docker ps
```

```
docker service ls
```


Kubernetes

What is Kubernetes?

- A highly collaborative open source project originally conceived by Google
- Sometimes called:
 - Kube
 - K8s
- Start, stop, update, and manage a cluster of machines running containers
- Particularly suited for horizontally scaleable, stateless, or 'microservices' application architectures
 - K8s > (docker swarm + docker-compose)
- Kubernetes does NOT and will not expose all of the 'features' of the docker command line.
- Minikube : a tool that makes it easy to run Kubernetes locally.

Kubernetes

Kubernetes vs docker swarm : Terminology

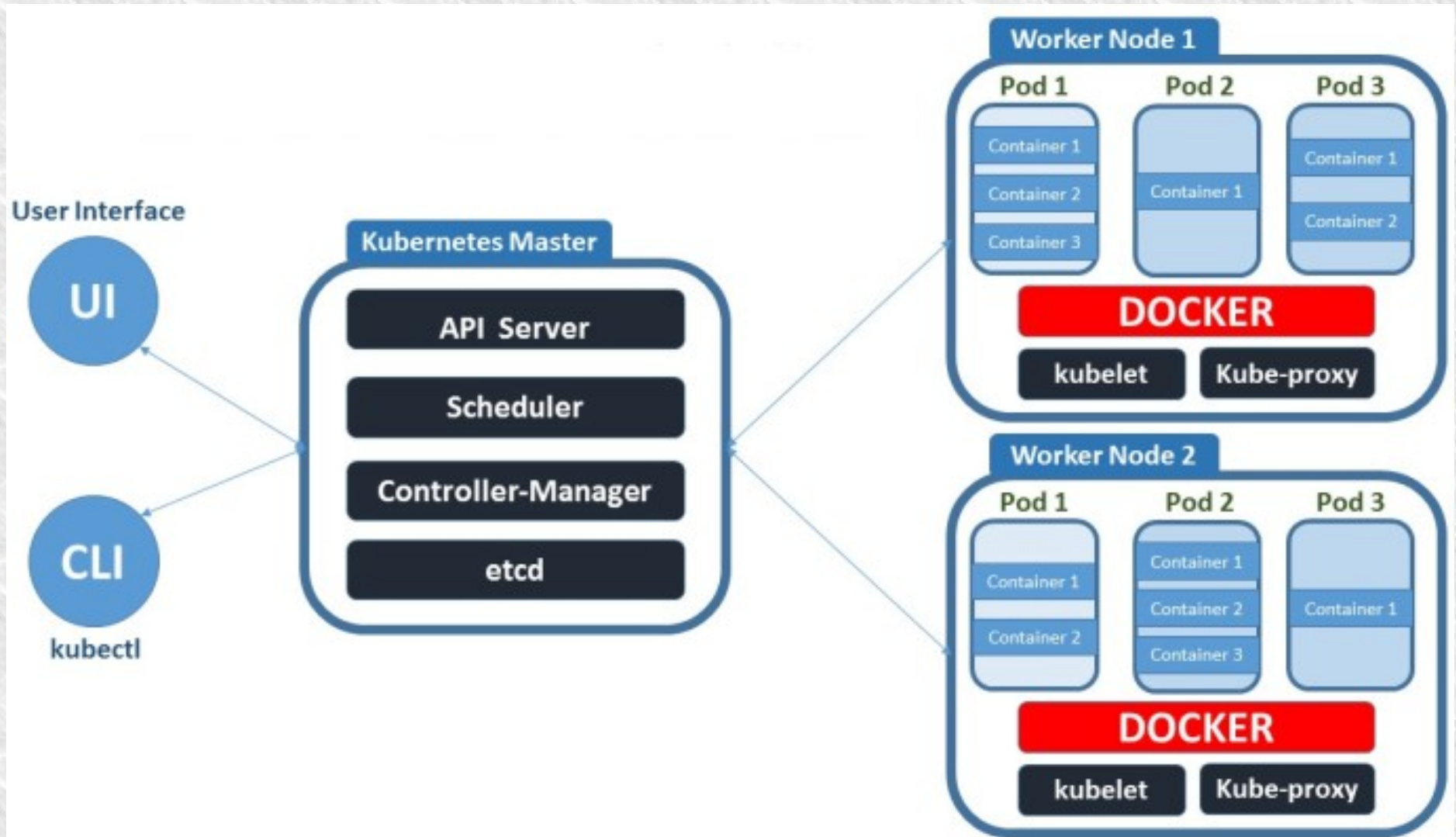
| | Docker swarm | Kubernetes |
|----------------------|---------------|-----------------------------|
| Controller | Manger | Master |
| Slave | Worker | Node worker |
| Workload Definition | Service | Deployment |
| Deployment Unit | Task | Pod |
| Scale-out Definition | Replicas | Replica Set |
| Service Discovery | DNS | DNS |
| Load Balancing | Ingress | Service |
| Port | PublishedPort | Endpoint |
| Storage | Volumes | Persistent Volumes / Claims |
| Network | Overlay | Flat Networking Space |

Kubernetes

Kubernetes vs docker swarm : Features

| Features | Docker Swarm | Kubernetes |
|---|---|--|
| Installation & Cluster configuration | Installation very simple, but cluster not very strong | Installation complicated ; but once setup, the cluster is very strong |
| GUI | No GUI | GUI is the Kubernetes Dashboard |
| Scalability | Highly scalable & scales faster than kubernetes | Highly scalable & scales faste |
| Auto-Scaling | Can not do auto-scaling | Can do auto-scaling |
| Load Balancing | Does auto load balancing of traffic between containers in the cluster | Manual intervention needed for load balancing traffic between different containers in different Pods |
| Rolling Updates & Rollbacks | Can deploy Rolling updates, but not automatic Rollbacks | Can deploy Rolling updates, & does automatic Rollbacks |
| Data Volumes | Can share storage volumes with any other container | Can share storage volumes only with other containers in same Pod |
| Logging & Monitoring | 3rd party tools like ELK should be used | In-built tools for logging & monitoring |

Kubernetes Architecture



Kubernetes Master

- Typically consists of:
 - **Kube-apiserver** : exposes the Kubernetes API
 - **Kube-scheduler** : select a node for newly created Pod
 - **Kube-controller-manager** : runs controller processes (nodes, jobs, sa, ...)
 - **etcd** : key value store for all cluster data.
- Might contain:
 - Kube-proxy
 - a network plugin
 - kubelet

Kubernetes Node

- Typically consists of:
 - **Kubelet** : An agent that makes sure that containers are running in a Pod.
 - **Kube-proxy** : A network proxy implementing part of the Kubernetes Service concept.
- Might contain:
 - **a network Plugin** : Responsible for allocating IP addresses to pods and enabling them to communicate with each other within the cluster

Kubernetes Pod

- Single schedulable unit of work
 - Can not move between machines.
 - Can not span machines.
- One or more containers
 - Shared network namespace
- Metadata about the container(s)
- Env vars – configuration for the container
- Every pod gets an unique IP
 - Assigned by the container engine, not kube

Kubernetes

Pod - example

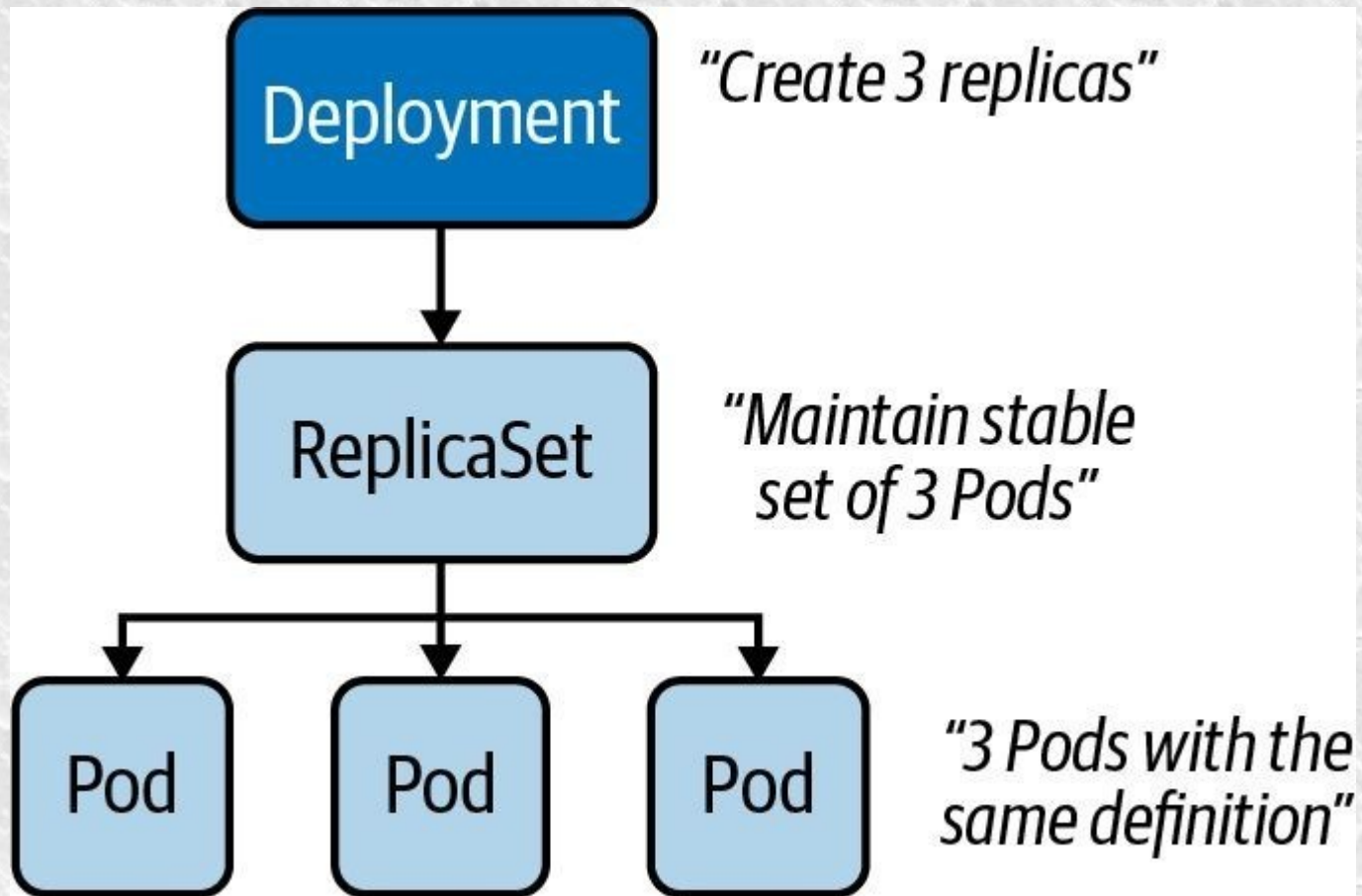


Kubernetes Deployment

- A Deployment controller provides declarative updates for Pods and ReplicaSets.
- You describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state.
- Deployment benefits :
 - Deploy a RS / pod
 - Horizontal / Vertical scaling
 - Autoscaling
 - Pause and resume the Deployment.
 - Deployment strategies (Rolling update, blue/green, canary, ...)
 - Rollback to older Deployment versions
 - ...

Kubernetes

Deployment - example

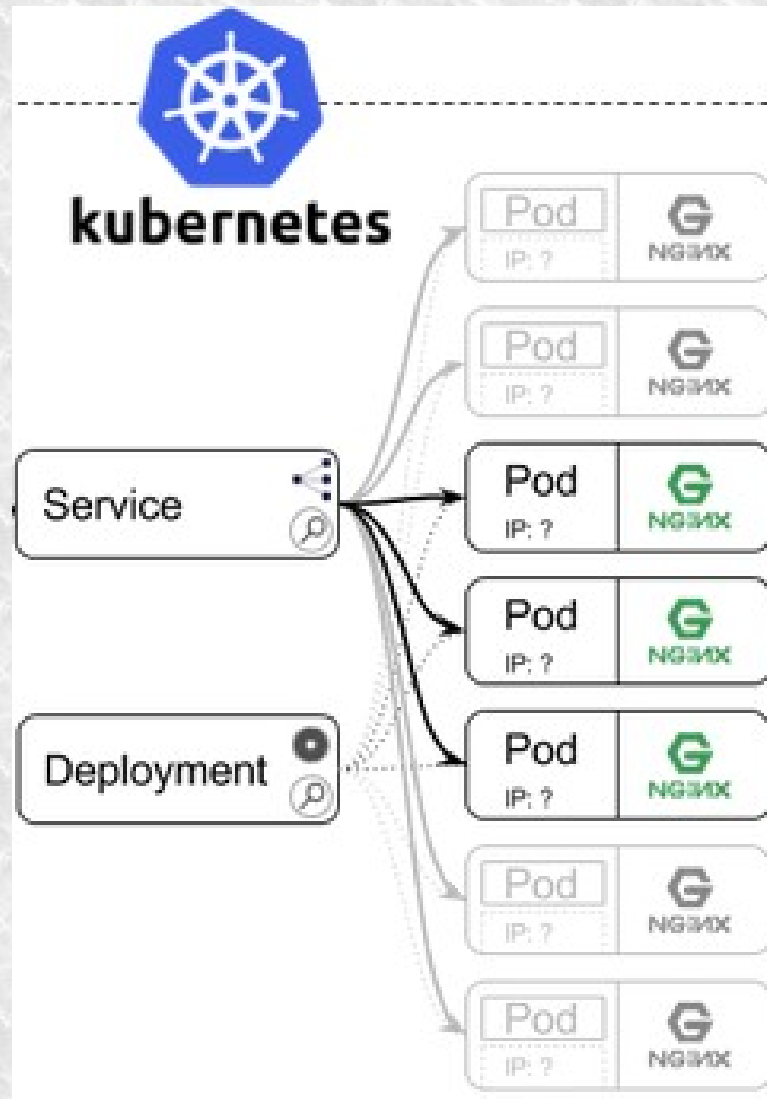


Kubernetes Services

- A grouping of pods that are running on the cluster.
- Sometimes called a micro-service.
- Usually determined by a Label - Selector.
- Provide important features that are standardized across the cluster:
 - Load-balancing
 - service discovery between applications
 - features to support zero-downtime application deployments.
- When creating a service, one or more ports can be configured.

Kubernetes

Services - example



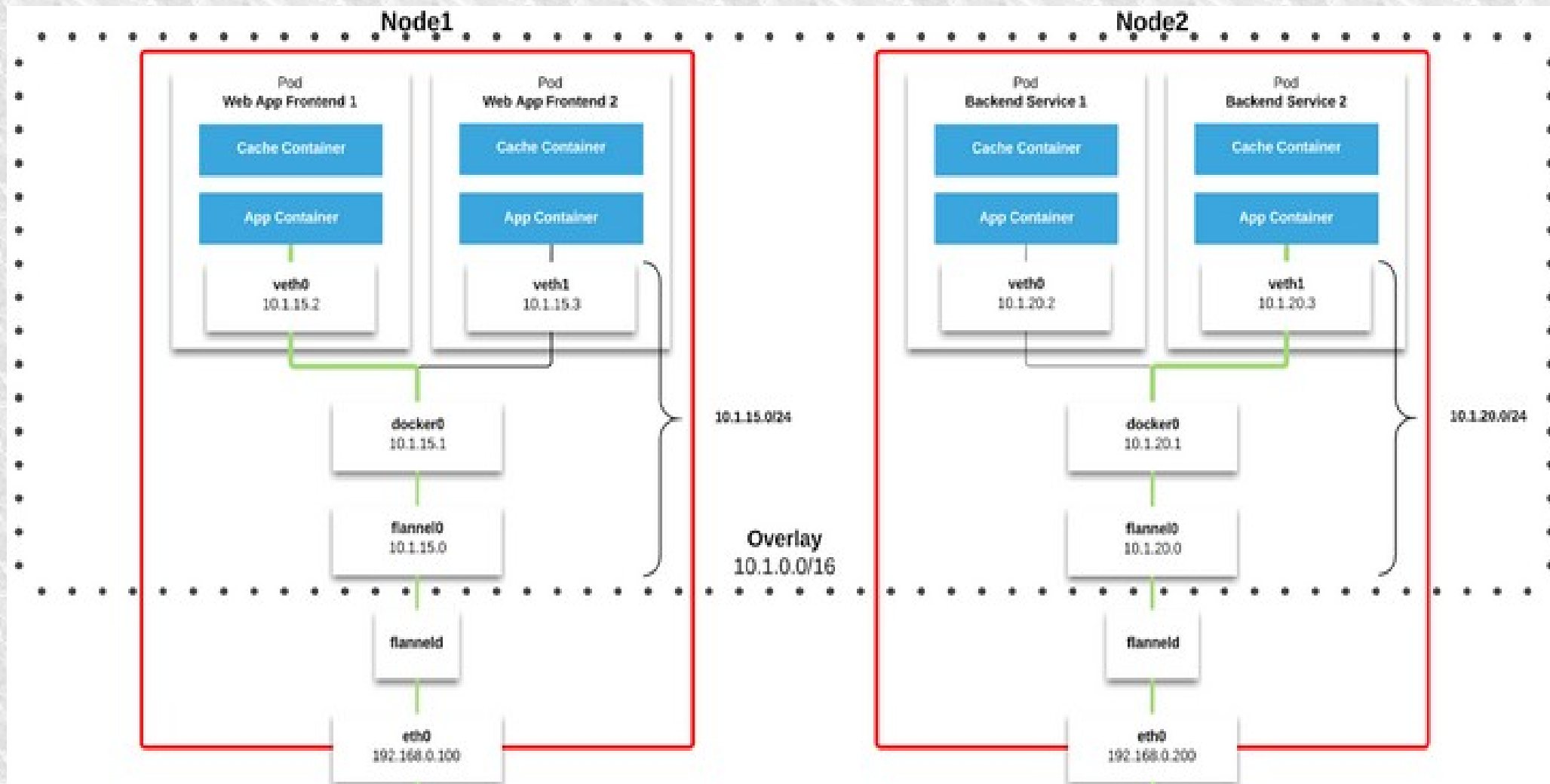
Kubernetes

Network Model

- Every Pod get its own IP address
 - Not need to explicitly create links between Pods
 - Almost, never need to deal with mapping container ports to host ports
- pods on a node can communicate with all pods on all nodes without NAT
- agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node
- There are a number of ways that this network model can be implemented.
 - Flannel is a very simple overlay network that satisfies the Kubernetes requirements.

Kubernetes

Networking with flannel



Kubernetes

kubeadm command

- kubeadm performs the actions necessary to get a minimum viable cluster up and running.
- Initializes a Kubernetes control-plane node (master) :
 - *Kubeadm init*
- initializes a Kubernetes worker node and joins it to the cluster :
 - *kubeadm join*
- Reverts any changes made by kubeadm init or kubeadm join :
 - *kubeadm reset*

Kubernetes

kubectl command

- Running commands against Kubernetes clusters.
- Syntax :

kubectl [command] [TYPE] [NAME] [flags]

- Command: create, get, describe, delete
- Type : ressource type (pod, service, depoyment, node, ...)
- NAME: ressource name (pod1, node1, etc ...)
- flags: Specifies optional flags.

LPI DevOps Tools Engineers

Module 6

Ansible and configuration management tools

Plan

- Configuration management tools
- Ansible
- Inventory
- Playbook
- Variables
- Roles
- ansible-vault

Configuration management tools

Problem – repetitive jobs

- Building VM templates
 - ISO install and configuration
 - Network setup
 - Set up users/group, security, authentication/authorization
 - Software install and configuration
- Building out clusters
 - Cloning N number of VMs from X number of templates
 - Hostname/network configuration
- Server maintenance
- Etc ...

Configuration management tools

Solution : Ansible, Chef, Puppet, ...

| | Language | Agent | Configuration | Communication |
|----------------|----------|-------|---------------|---------------|
| Ansible | Python | No | YAML | OpenSSH |
| Chef | Ruby | Yes | Ruby | SSL |
| Puppet | Ruby | Yes | Puppet DSL | SSL |

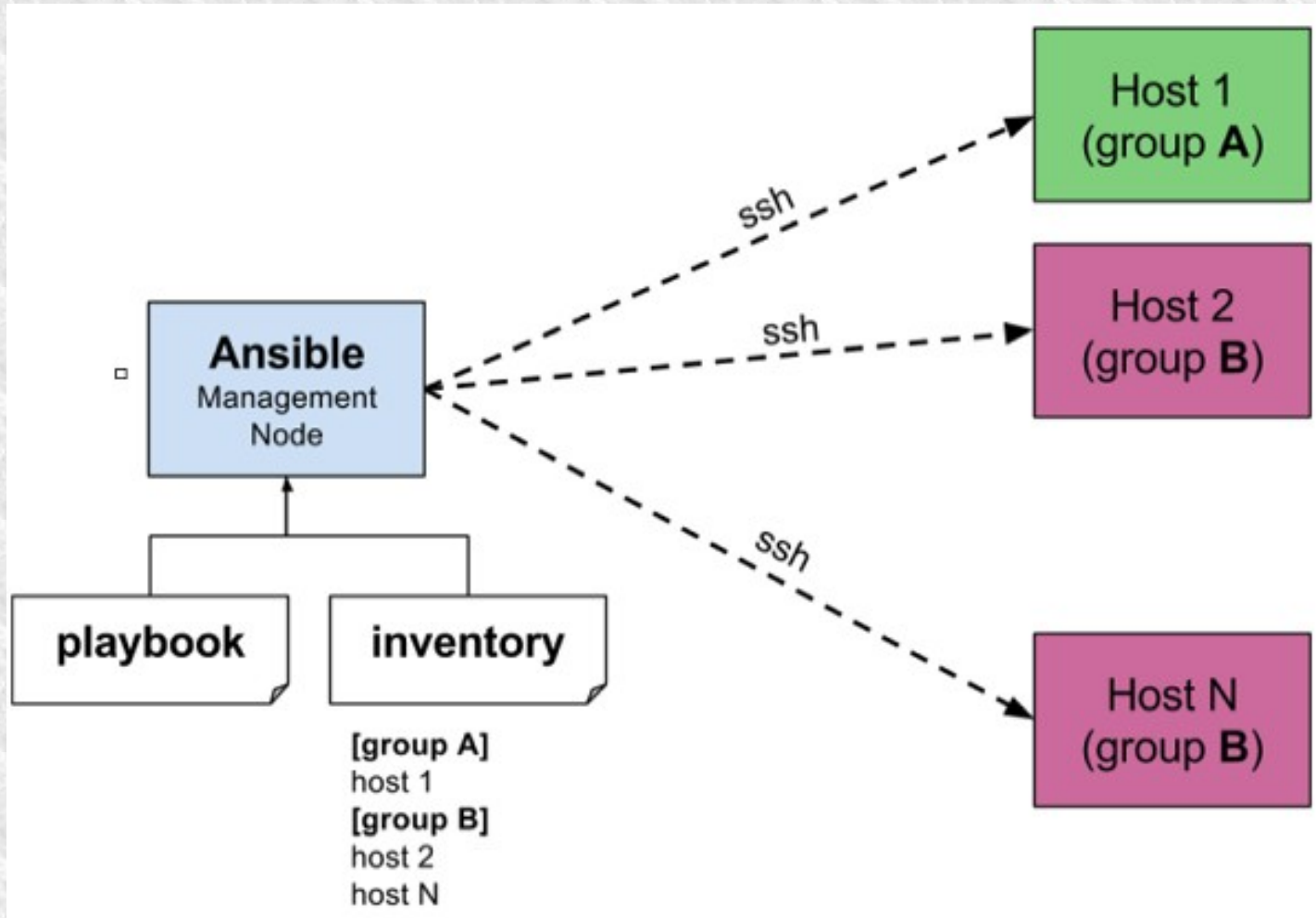
Configuration management tools

Why ansible

- Agentless!
- Uses SSH
- Easy-to-read Syntax as YAML file
- Push-Based
- Built-in-Modules
- Full power at the CLI (ansible-doc -l)

Ansible

How ansible work



Inventory

What is it ?

- A list of hosts and groups in */etc/ansible/hosts* by default.
- Can be dynamic or static
- Groups defined by brackets [] and by name
 - Describe systems
 - Decide what systems you are controlling at what times and for what purpose (roles)
 - Groups can be nested with :children
- Hosts can be in more than one group
 - server could be both a webserver and a dbserver.
 - variables will come from all of the groups they are a member of

Inventory Example

- **INI-like version :**

```
mail.example.com
[webservers]
foo.example.com
bar.example.com
[dbservers]
one.example.com
two.example.com
three.example.com
```

- **YAML version :**

```
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:
```


Inventory

Host selection

- Host selection can be done by including or excluding groups and single hosts
- Selection can be done by passing :
 - all / *
 - Groups names
 - Exclusion (all:!CentOS)
 - Intersection (webserver:&staging)
 - Regex

Playbook

ad hoc commands

- Ad-Hoc: commands which execute single tasks
- Tasks: leverage an Ansible module, which is executed on the target host
- Modules:
 - Written in Python (mostly)
 - Shipped via SSH to the target host
 - Return JSON, interpreted by Ansible for outcome
 - Removed once executed
- Examples :
 - Deleting whole directory and files on server1 :
\$ ansible abc -m file -a "dest = /path/user1/new state = absent"
 - Gathering Facts on all servers/machines

\$ ansible all -m setup

Brahim HAMDI

110

Playbook

Orchestration with playbooks

- The true power of ansible comes from abstraction and orchestration, using playbooks
- Playbooks are the files where Ansible code is written (in YAML format).
- It is a set of ordered tasks, combined with selected targets.
- Playbooks provide ready-made strategies for bringing (groups of) hosts to a desired state.
- Groups/hosts are defined in inventory file.
- Run an ansible playbook :

\$ ansible-playbook file.yml

Playbook Loops

- Many types of general and special purpose loops :
 - with_nested
 - with_dict
 - with_fileglob
 - with_together
 - with_sequence
 - until
 - with_random_choice
 - with_first_found
 - with_indexed_items
 - with_lines

Playbook

Conditional tasks

- When : only run this on Red Hat OS :
- Example :

- name: This is a Play

hosts: web-servers

remote_user: mberube

become: sudo

tasks:

- name: install Apache

yum: name=httpd state=installed

when: ansible_os_family == "RedHat"

Playbook

Handlers

- Only run if task has a “changed” status

- Example :

```
- name: This is a Play
hosts: web-servers
```

tasks:

```
- yum: name={{ item }} state=installed
```

```
with_items:
```

```
- httpd
```

```
- memcached
```

```
notify: Restart Apache
```

```
- template: src=templates/web.conf.j2 dest=/etc/httpd/conf.d/web.conf
```

```
notify: Restart Apache
```

handlers:

```
- name: Restart Apache
```

```
service: name=httpd state=restarted
```

Playbook Tags

- Example of tag usage (example.yml) :

tasks:

- yum: name={{ item }} state=installed

with_items:

- httpd

- memcached

tags:

- packages

- template: src=templates/src.j2 dest=/etc/foo.conf

tags:

- configuration

- Running with tags :

\$ ansible-playbook example.yml --tags "configuration"

\$ ansible-playbook example.yml --skip-tags "notification"

Variables

Setting Variables

- Variables in Ansible help you to contextualise and abstract roles.
- Variables can be defined in several areas
 - Inventory
 - Playbook
 - Files and Roles
 - Command Line
 - Facts

Variables

Host Variables

- Host variables are assigned in the inventory.
- Arbitrary variables can be assigned to individual hosts.
- There are also variables which change the way

Ansible behaves when managing hosts e.g :

```
90.147.156.175 \
```

```
ansible_ssh_private_key_file=~/.ssh/ansible-default.key \ ansible_ssh_user=centos
```

Variables

Group Variables

- Hosts are grouped according to aspects, or any desired grouping.
- Ansible allows you to define group variables which are available for any host in a group
- Group variables can be defined in the inventory:

```
[webservers:vars]
```

```
http_port=80
```

- Or in separate files under group_vars

```
group_vars/webservers → ---
```

```
http_port=80
```

Variables

Registering and using variables

- Ansible registers are used to capture the output (result) of a task to a variable.
 - can then use the value of these registers for different scenarios like a conditional statement, logging etc.
- The variables will contain the value returned by the task.
- Each registered variables will be valid on the remote host where the task was run for the rest of the playbook execution.
- Example

- *hosts: all*

tasks:

- *name: Ansible register variable basic example*

*shell: "find *.txt"*

args:

chdir: "/Users/mdtutorials2/Documents/Ansible"

register: find_output

- *debug:*

var: find_output

Variables

Reference a field

- supports dictionaries which map keys to values.
- Example :

foo:

field1: one

field2: two

- can then reference a specific field in the dictionary using :
 - bracket notation : *foo['field1']*
 - or
 - dot notation: *foo.field1*

Variables

Magic Variables

- Some variables are automatically created and filled by Ansible :
 - inventory_dir
 - inventory_hostname
 - inventory_hostname_s
 - host
 - inventory_file
 - playbook_dir
 - play_hosts
 - hostvars
 - groups
 - group_names
 - ansible_ssh_user

Roles

What is role

- A redistributable and reusable collection of:
 - tasks
 - files
 - scripts
 - templates
 - variables
- Often used to setup and configure services
 - install packages
 - copying files
 - starting deamons

Roles

Directory Structure

- Roles are usually placed in a "library" in a sub-directory.
- Each role has a standard structure :

roles

└─ myapp

└─ defaults

└─ files

└─ handlers

└─ meta

└─ tasks

└─ templates

└─ vars

Roles

ansible-galaxy

- A new role can be created using :

ansible-galaxy init <rolename>

- Ensure that you create the role in the "roles" directory, or you won't be able to simply call them by name in the playbooks.
- Ansible Galaxy creates all the files you need to get started, including a README and a meta file.
- Roles can be shared and discovered via :

<http://galaxy.ansible.com>

Roles

Playbook examples

- ---
- *hosts: webservers*
 roles:
 - *common*
 - *webservers*
- ---
- *hosts: webservers*
 roles:
 - *common*
 - *{ role: myapp, dir: '/opt/a',port: 5000 }*
 - *{ role: myapp, dir: '/opt/b',Port: 5001 }*
- ---
- *hosts: webservers*
 roles:
 - *{ role: foo, when: "ansible_os_family == 'RedHat'" }*

ansible-vault

What is it

- A feature of ansible that keep sensitive data such as passwords or keys in encrypted files (rather than as plaintext in playbooks or roles)
- These vault files can then be distributed or placed in source control.
- To enable this feature, a command line tool - ansible-vault - is used to edit files.
- It can encrypt any structured data file used by Ansible.
 - “group_vars/” or “host_vars/” inventory variables,
 - variables loaded by “include_vars” or “vars_files”,
 - Role variables and defaults,
 - Ansible tasks, handlers, etc ...

ansible-vault

How to use - examples

- Creating encrypted files (new files) :
ansible-vault create foo.yml
- Encrypting Unencrypted Files (existing files) :
ansible-vault encrypt foo.yml bar.yml baz.yml
- Decrypting Encrypted Files
ansible-vault decrypt foo.yml bar.yml baz.yml
- Editing Encrypted Files
ansible-vault edit foo.yml
- Rekeying Encrypted Files
ansible-vault rekey foo.yml bar.yml baz.yml
- Create encrypted variables to embed in yaml
encrypt_string
- Viewing Encrypted Files
ansible-vault view foo.yml bar.yml baz.yml

LPI DevOps Tools Engineers

Module 7

CI/CD with Jenkins

Plan

- CI/CD
- Jenkins
- Building a CI/CD Pipeline Using Jenkins

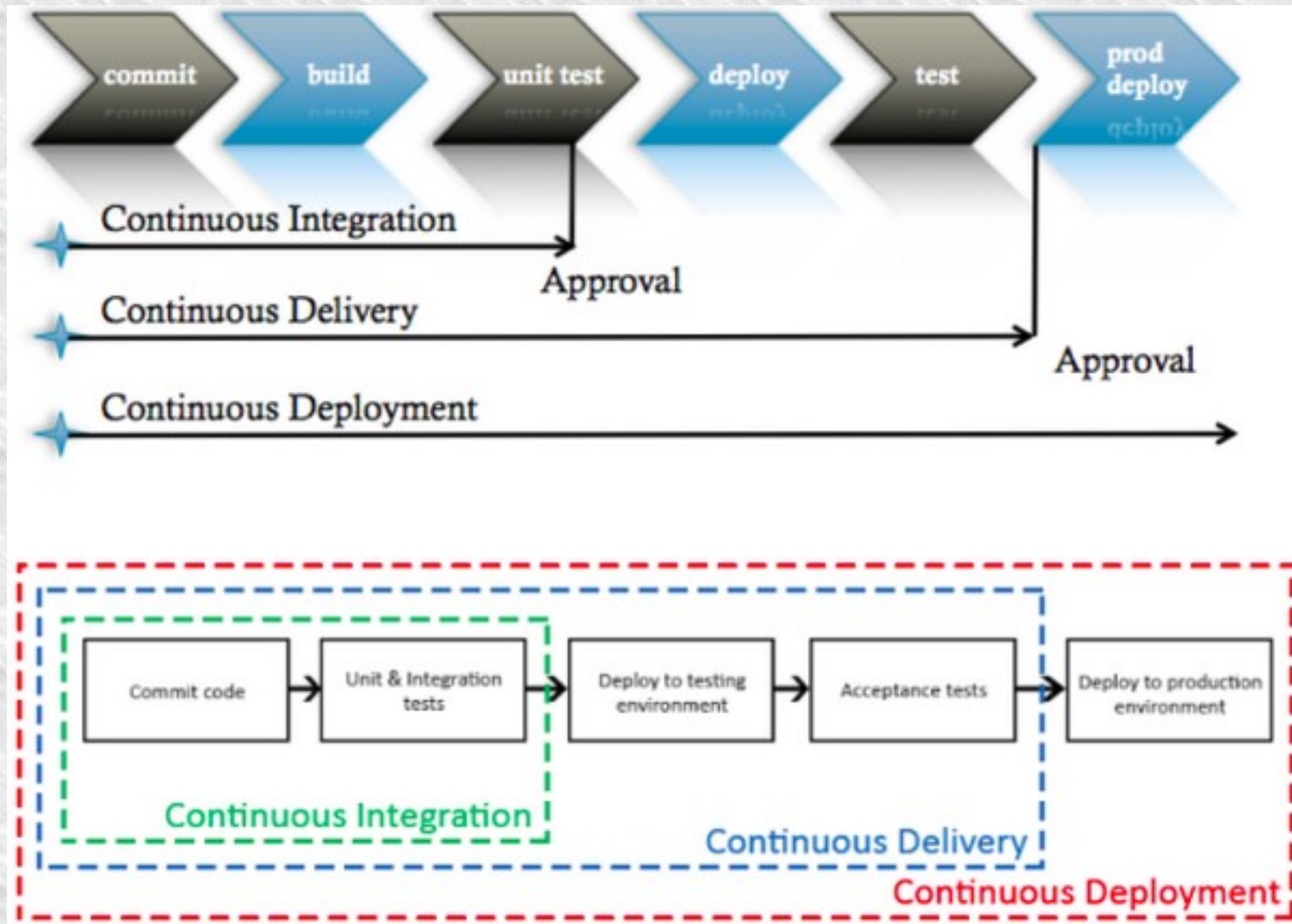
CI/CD

DevOps lifecycle



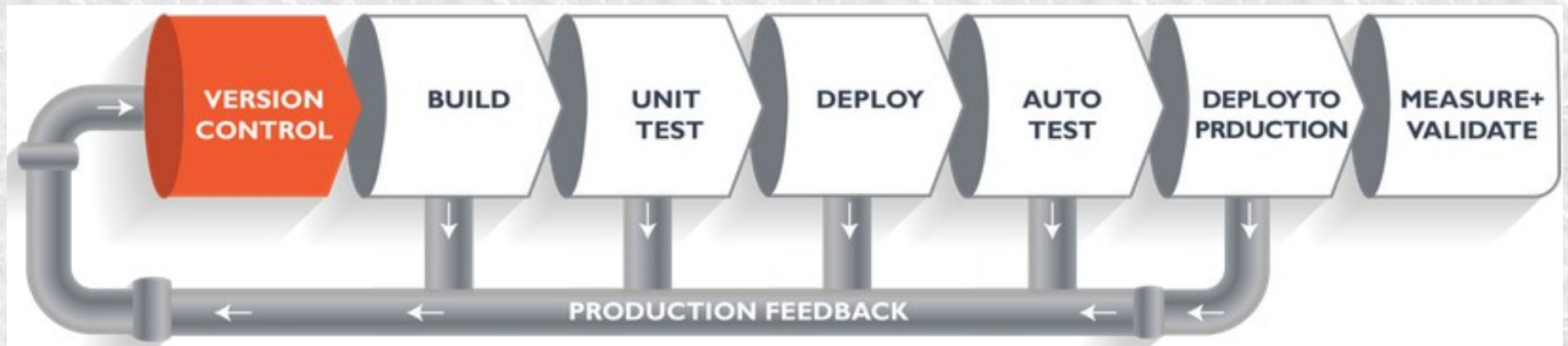
CI/CD

What is CI/CD ?



CI/CD

CI/CD pipeline



Jenkins

Tools in the CI/CD Pipeline

- Entire software development lifecycle in DevOps/automated mode :
 - automate the entire process, from the time the development team gives the code and commits it to the time get it into production.
- ⇒ need automation tools
- Git used by development team to commit the code.
 - From Git, Jenkins pulls the code and then moves it into the commit phase
 - Tools like maven (for java code) with in Jenkins can then compile that code
 - Then jenkins deployed the exec to run a series of tests.
 - Then, it moves by jenkins on to the staging server to deploy it using Docker.
 - After a series of unit tests or sanity tests, it moves on to production.

Jenkins

Plugins and mailer

- Jenkins uses plugins to :
 - Integrate most version control systems.
 - Support many build tools.
 - Generate unit test reports in various formats (JUnit, NUnit, etc ...).
 - Supports automated tests.
 - Etc ...
- It allows configuring email notifications for build results.
 - Failed build.
 - Unstable build.
 - Successful build after a failed build, indicating that a crisis is over
 - Unstable build after a successful one, indicating that there's a regression

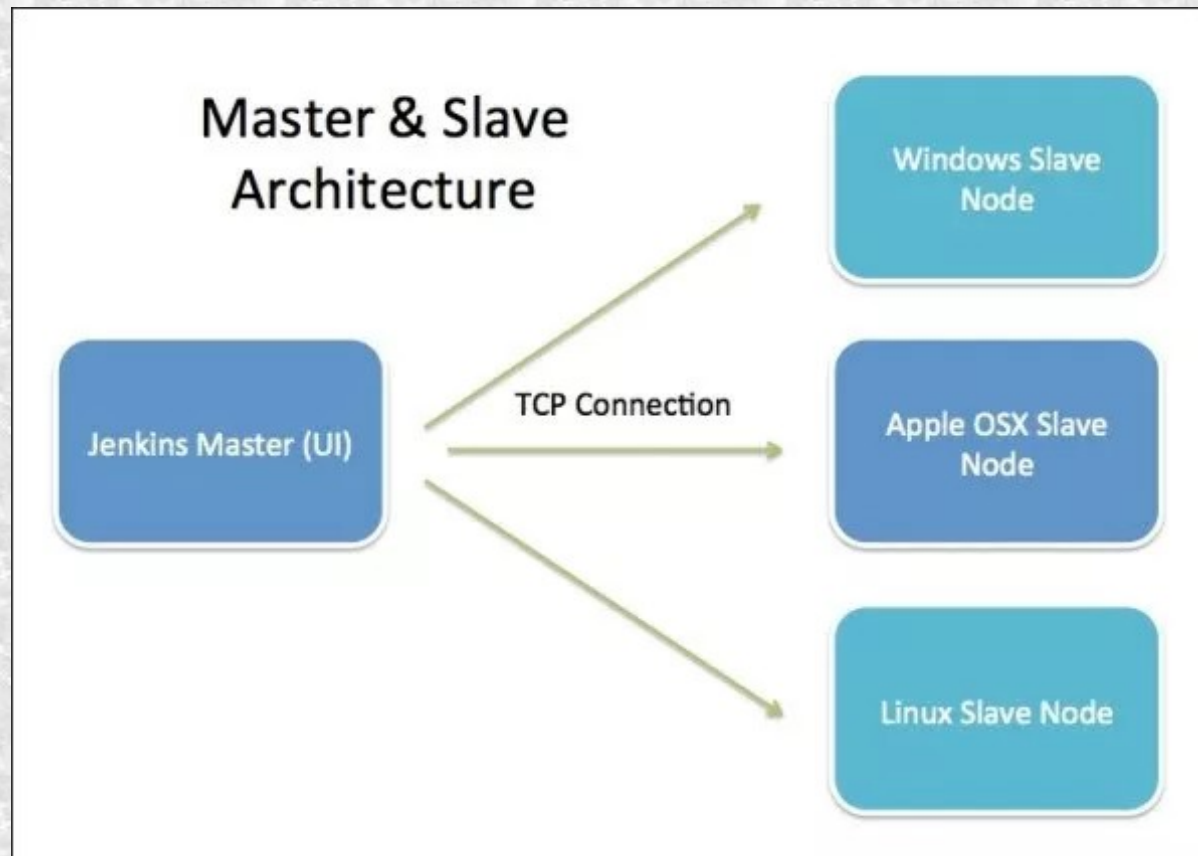
Jenkins

Master/slave architecture

- Jenkins supports the master-slave architecture.
 - known as Jenkins Distributed Builds.
- Jenkins can run the same test case on different environments in parallel using Jenkins Distributed Builds.
 - which in turn helps to achieve the desired results quickly.
- All of the job results are collected and combined on the master node for monitoring.

Jenkins

Master/slave architecture - example



Jenkins SDL

What is it ?

- A Jenkinsfile can be written using two types of syntax
 - Declarative syntax
 - Scripted syntax
- Declarative and Scripted Pipelines are constructed fundamentally differently.
- Declarative Pipeline is a more recent feature of Jenkins Pipeline which:
 - provides richer syntactical features over Scripted Pipeline syntax,
 - is designed to make writing and reading Pipeline code easier.

Jenkinsfile

Declarative pipeline

- Pipelines can be defined with a simpler syntax.
- Declarative “section” blocks for common configuration areas, like
 - Stages
 - Tools
 - post-build actions
 - Notifications
 - Environment
 - build agent or Docker image
- All wrapped up in a *pipeline* { ... } step, with syntactic and semantic validation available.
- It's configured and run from a Jenkinsfile.

LPI DevOps Tools Engineers

Module 8

IT monitoring

Plan

- Monitoring
- Prometheus
- Prometheus' configuration file
- Exposing metrics for Prometheus
- Prometheus alertmanager
- Dashboarding with grafana

Monitoring

Why monitor ?

- Observe behavior of business functions/applications in real-time
 - Availability and health
 - Performance
 - Etc ...
- Gather operational metrics
 - And prepare | wrangle metrics (tag, filter, enrich, aggregate, ...)
- Raise alert
 - To human (via ticket/SMS/...)
 - To automated handler/agent
- Support issue resolution (data for root cause analysis)
- Analyze trends + effects/impact of change.

Monitoring

Metrics are collected across the stack

- Business Applications
 - SaaS, Standard Applications
 - Custom | Tailor made applications
- Platform
 - Web Server, Application Server
 - Database
 - LDAP
- Infrastructure
 - Container, Container Platform (Docker, Kubernetes, ...)
 - Operating System
 - Cache
 - Proxy, Load Balancer
 - Network
 - Storage, File System

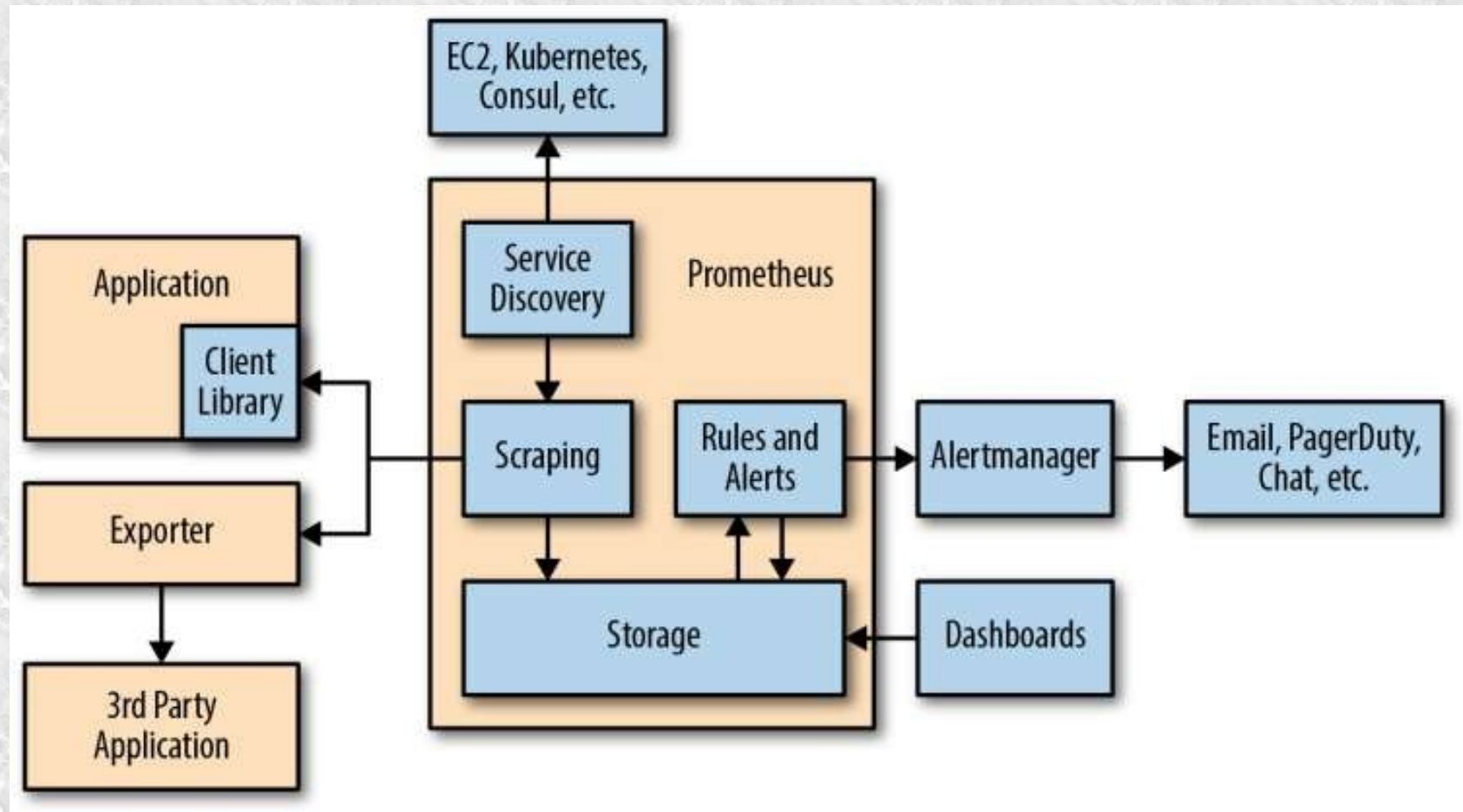
Prometheus

What is it ?



- Part of CNCF
- Since 2012
- Written in Go Lang
- Open source
- Gathering metrics into database
 - Scheduled pull |harvest| scrape actions – HTTP/TCP requests
 - Accessing Exporters and built in (scrape) endpoints.
- Treating time-series data as a data source for generating alerts.

Prometheus Architecture



Prometheus

Prometheus' configuration file

- Simply YAML file (.yml extension).
- Divided into three parts: `global`, `rule_files`, and `scrape_configs`.
 - `global` : the general configuration of Prometheus.
 - *scrape_interval* : How often Prometheus scrapes targets
 - *evaluation_interval* controls how often the software will evaluate rules.
 - `rule_files` : information of the location of any rules we want the Prometheus server to load.
 - `scrape_configs` : which resources Prometheus monitors.

Prometheus

Prometheus' configuration file Example

global:

scrape_interval: 15s

evaluation_interval: 15s

rule_files:

- "first.rules"

- "second.rules"

scrape_configs:

- job_name: 'prometheus'

scrape_interval: 5s

static_configs:

- targets: ['localhost:9090']

Exposing metrics for Prometheus

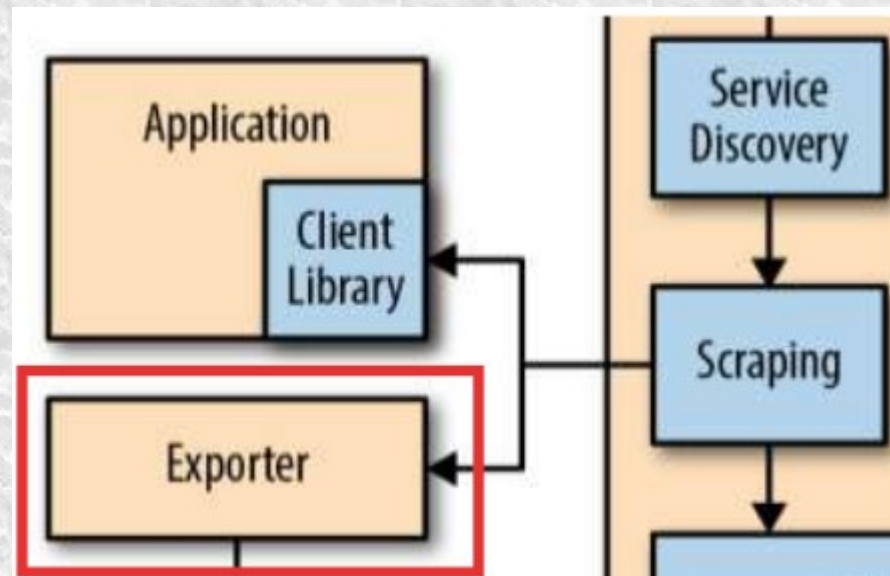
Scrape Metrics for Prometheus

- Configure the endpoint on the Prometheus server in the prometheus' configuration file.
- Have the application or environment listen for HTTP requests at a specific endpoint (for example: host:port/metrics).
- Return Metrics in the proper format to GET requests to this endpoint.
- Use a Client Library to easily compose the proper metrics response messages.

Exposing metrics for Prometheus

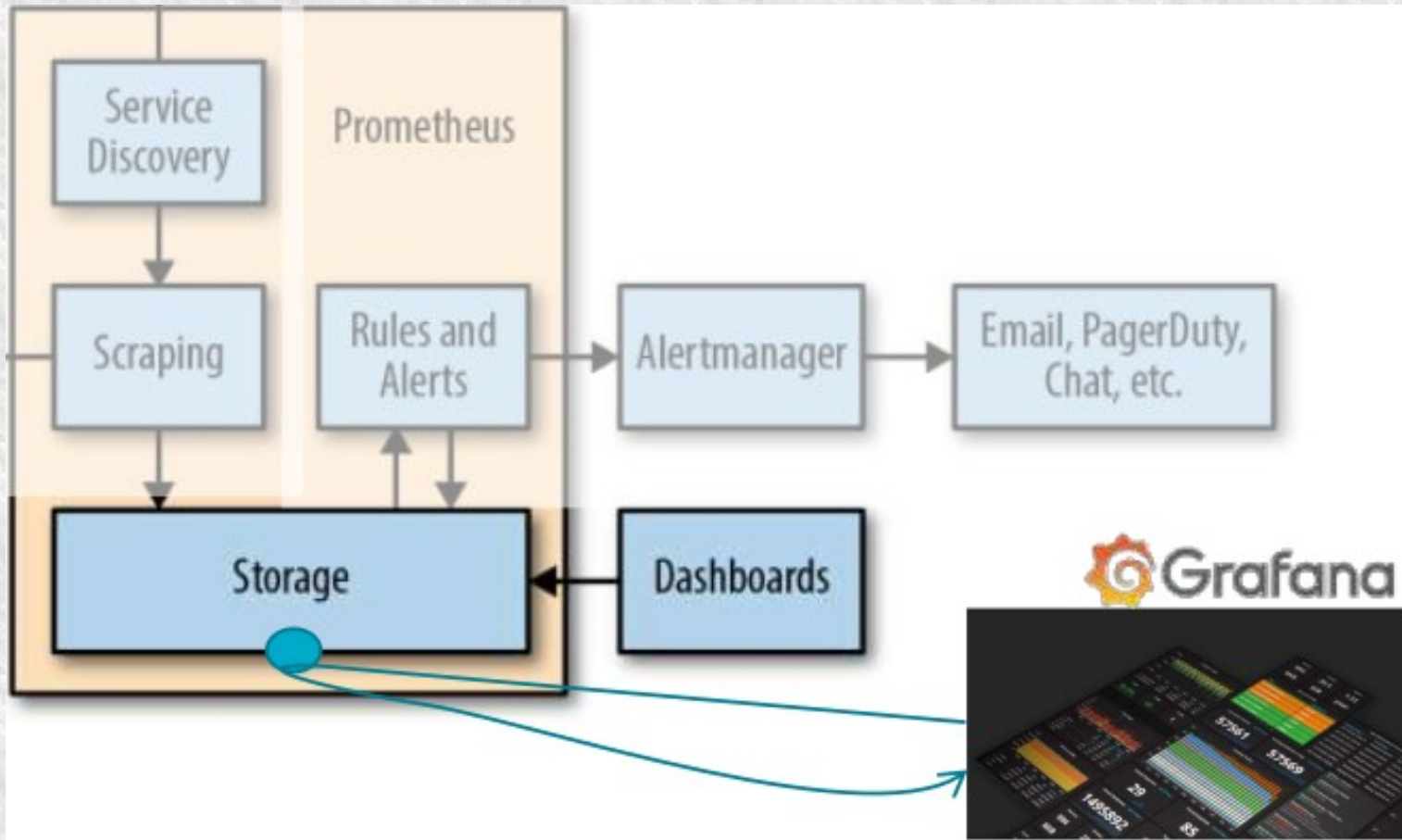
Prometheus Exporters

- Specialized adapters to expose metrics for specific technology components.
 - Installed and configured for a specific component
 - Scraped by Prometheus based on config file prometheus.yml that references the endpoint exposed by the exporter.
- Exporters (<https://prometheus.io/docs/instrumenting/exporters/>)
 - Linux (node exporter)
 - Windows (WMI exporter)
 - Databases (mysql_exporter, ...)
 - Messaging Systems
 - Storage
 - Graphique (Graphite exporter)
 - APIs
 - Logging
 - Monitoring Systems
 - Application Servers & Container Platforms
 - Blackbox Exporter (blackbox_exporter) : TCP/IP, ...



Grafana

Dashboarding



LPI DevOps Tools Engineers

Module 9

Log management and analysis

Plan

- ELK stack
- Elasticsearch
- Logstash
- Kibana
- Filebeat

ELK stack

Why log analysis?

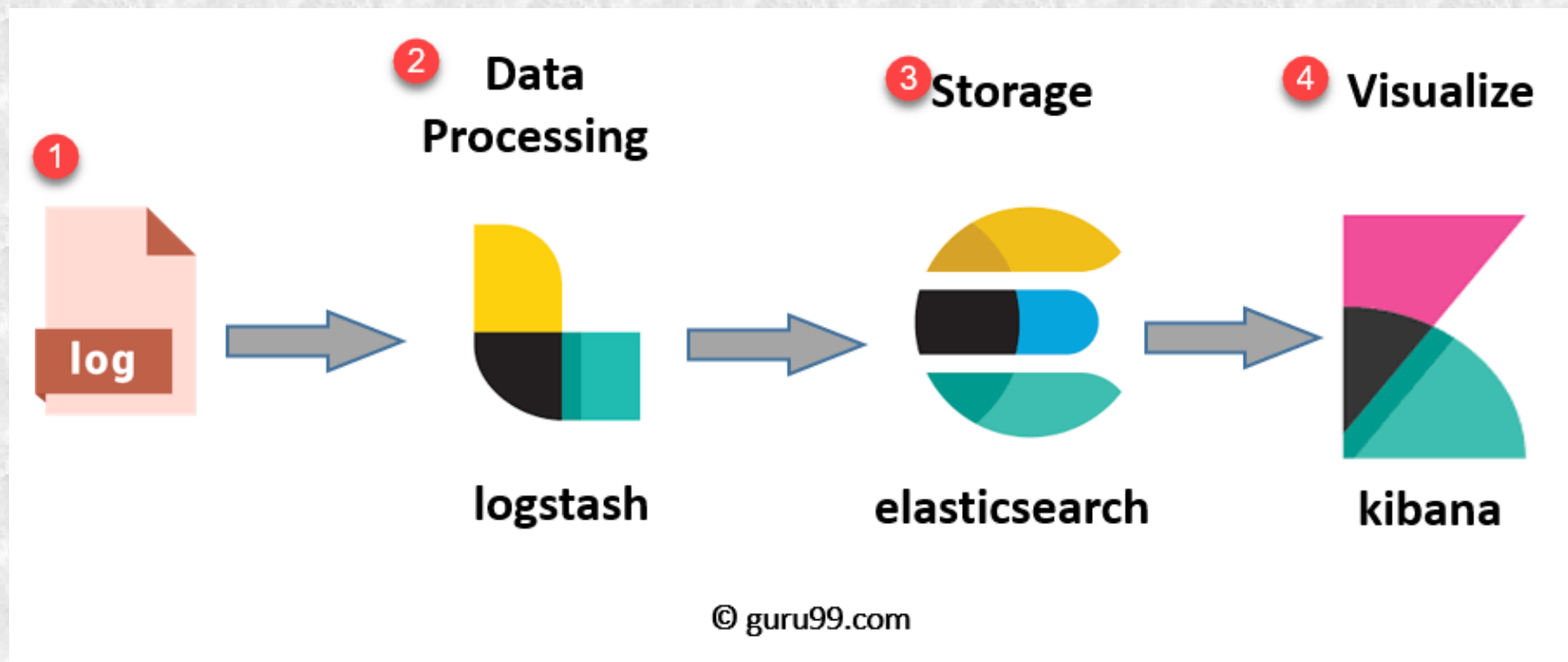
- Log management platform can monitor all above-given issues as well as process operating system logs, NGINX, IIS server log for web traffic analysis, application logs, and logs on cloud.
- Log management helps DevOps engineers, system admin to make better business decisions.
- The performance of virtual machines in the cloud may vary based on the specific loads, environments, and number of active users in the system.
 - Therefore, reliability and node failure can become a significant issue.

ELK stack

What is the ELK Stack?

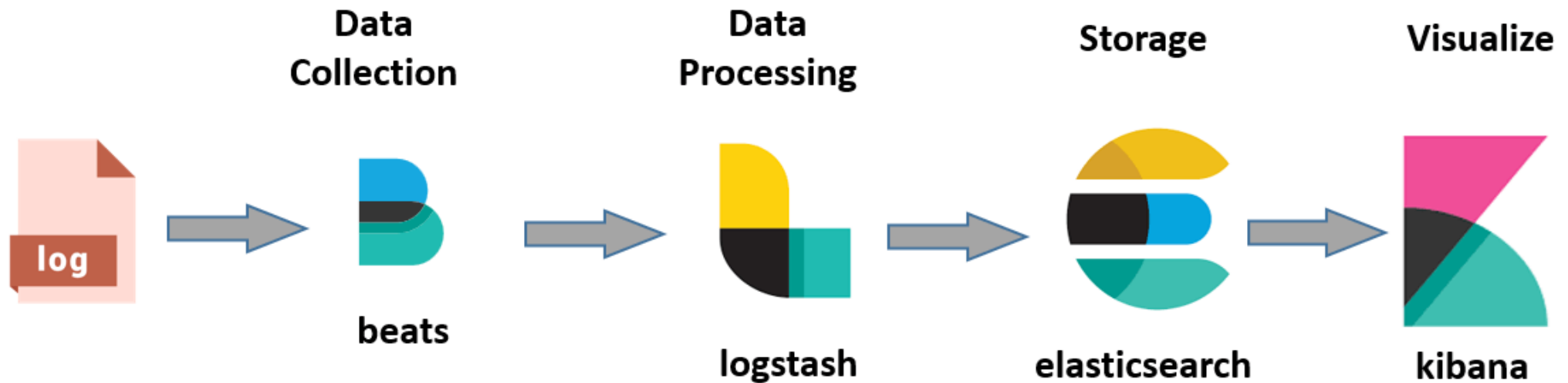
- A collection of three open-source products :
 - E stands for Elasticsearch: used for storing logs
 - L stands for LogStash : used for both shipping as well as processing.
 - K stands for Kibana: is a visualization tool (a web interface) which is hosted through Nginx or Apache
- Designed to take data from any source, in any format, and to search, analyze, and visualize that data in real time.
- Provides centralized logging that be useful when attempting to identify problems with servers or applications.
- It allows user to search all your logs in a single place.

ELK stack Architecture



ELK stack Architecture

- Beats : One more component is needed or Data collection called.



© guru99.com

Elasticsearch

What is the Elasticsearch?

- NoSQL database built with RESTful APIS.
- It offers advanced queries to perform detail analysis and stores all the data centrally.
- Also allows you to store, search and analyze big volume of data.
- Executing a quick search of the documents.
 - also offers complex analytics and many advanced features.
- Offers many features and advantages.

Logstash

What is Logstash?

- It is the data collection pipeline tool.
- It collects data inputs and feeds into the Elasticsearch.
- It gathers all types of data from the different source and makes it available for further use.
- Logstash can unify data from disparate sources and normalize the data into your desired destinations.
- It 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.

Kibana

What is Kibana?

- A data visualization which completes the ELK stack.
- 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.
- In Kibana there are different methods for performing searches on data.

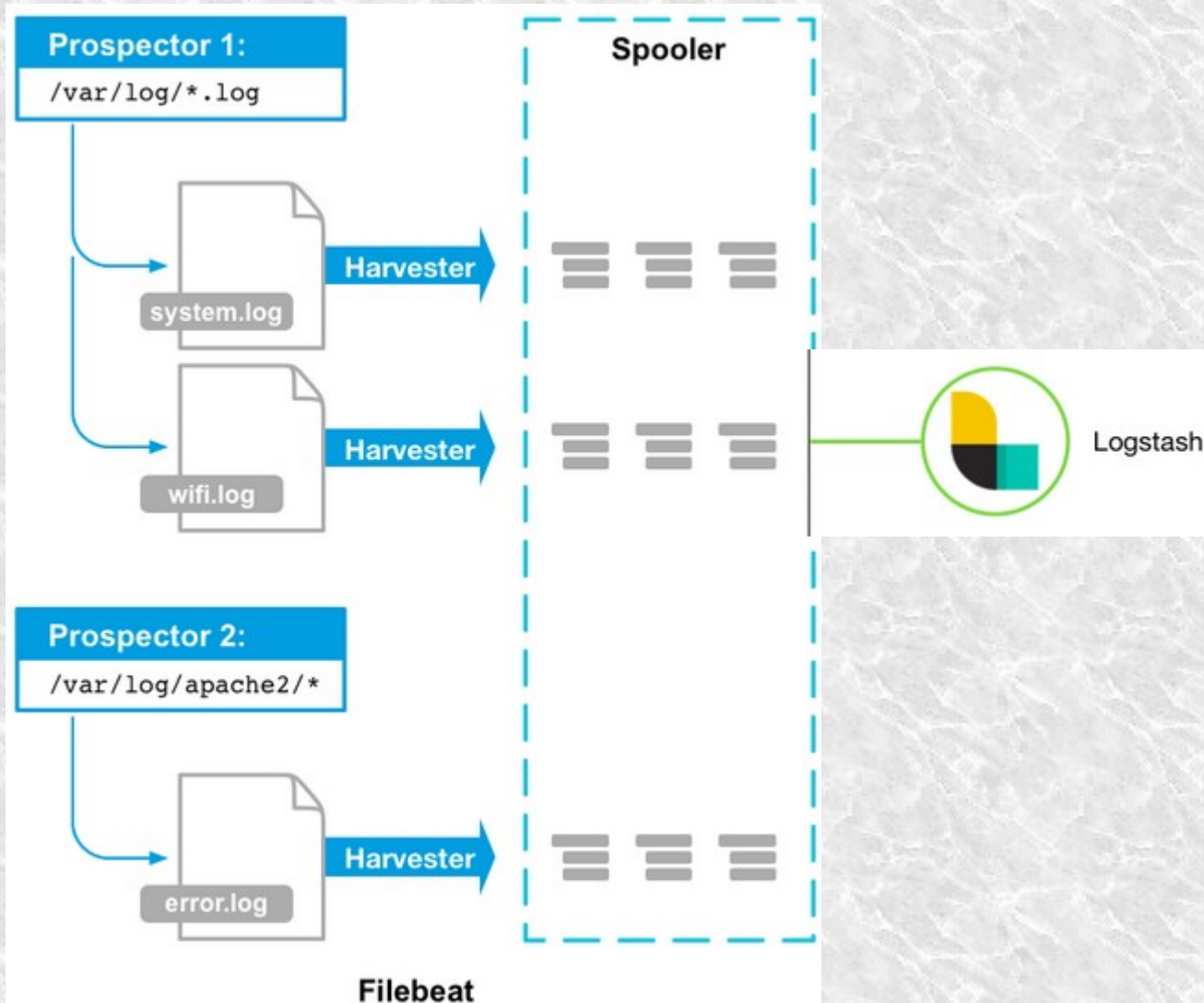
Filebeat

What is Filebeat

- Beats : a group of lightweight shippers installed on hosts for shipping different kinds of data into the ELK Stack for analysis.
- Each beat is dedicated to shipping different types of information :
 - Winlogbeat : ships Windows event logs.
 - Metricbeat : ships host metrics
 - Filebeat ...
- Filebeat is a log shipper belonging to the Beats family.
- Filebeat, as the name implies, ships log files.

Filebeat

Integration with logstash



Filebeat

Integration with logstash

- Common Config : Filebeat

- filebeat.prospectors:
 - *type: log*
 - enabled: true*
 - paths:*
 - */data/logs/reallog/2018-12-27.log*
- output.logstash:
 - hosts: ["target.aggserver.com:5044"]*

- Common Config : Logstash

```
input {  
  beats {  
    port => 5044  
  }  
}  
output {  
  file {  
    path => "/data/logstash/2018-12-27.log"  
    codec => line { format => "%{message}" }  
  }  
}
```