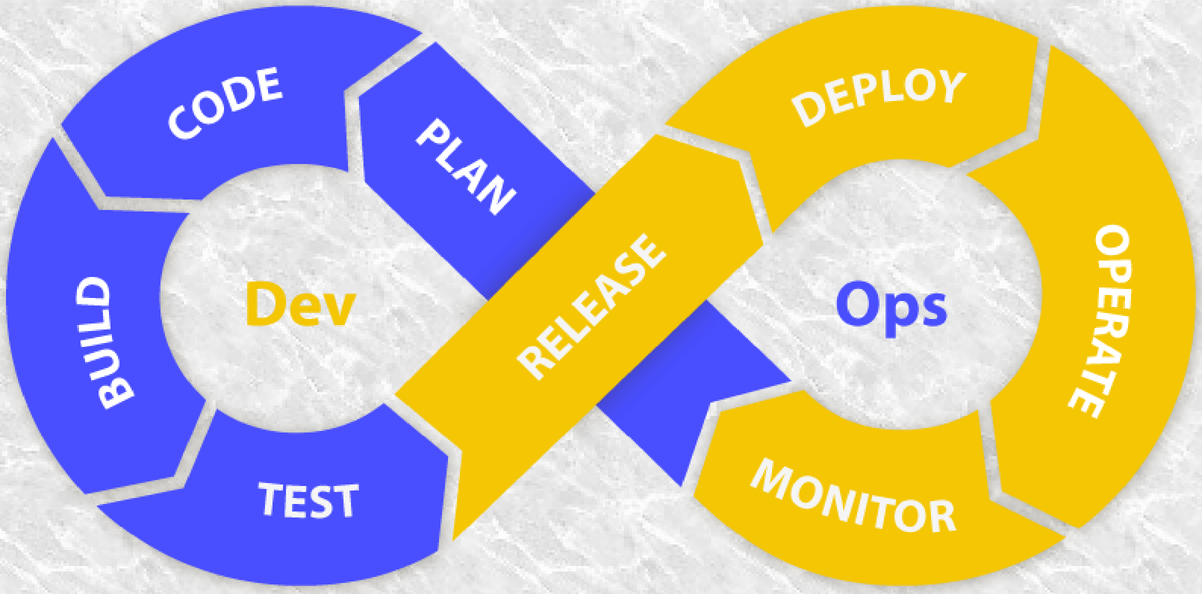
LPI DevOps Tools Engineers

LPIC Exam : 701-100



Brahim HAMDI

brahim.hamdi.consult@gmail.com

Linux Professional Institute DEVOPS TOOLS ENGINEER

aws certified DevOps Engineer Professional

aws certified Solutions Architect Associate

Linux Professional Institute LPIC-1 LINUX ADMINISTRATOR

Linux Professional Institute LPIC-2 LINUX ENGINEER

Linux Professional Institute LPIC-3 ENTERPRISE PROFESSIONAL MIXED ENVIRONMENT

CISCO CERTIFIED CCNA CYBER OPS

Septembre 2021

Plan

- **Module 1 : Modern Software Development**
- **Module 2 : Components, platforms and cloud**
- **Module 3 : Source code management**
- **Module 4 : System image creation and VM Deployment**
- **Module 5 : Container usage**
- **Module 6 : Container Infrastructure**
- **Module 7 : Container Deployment and Orchestration**
- **Module 8 : Ansible and configuration management tools**
- **Module 9 : CI / CD whith Jenkins**
- **Module 10 : IT monitoring**
- **Module 11 : Log management and analysis**

LPI DevOps Tools Engineers

Module 1

Modern Software Development

Plan

- Agile
- Service based applications
- RESTful APIs
- Application security risks

Agile

What is Agile ?

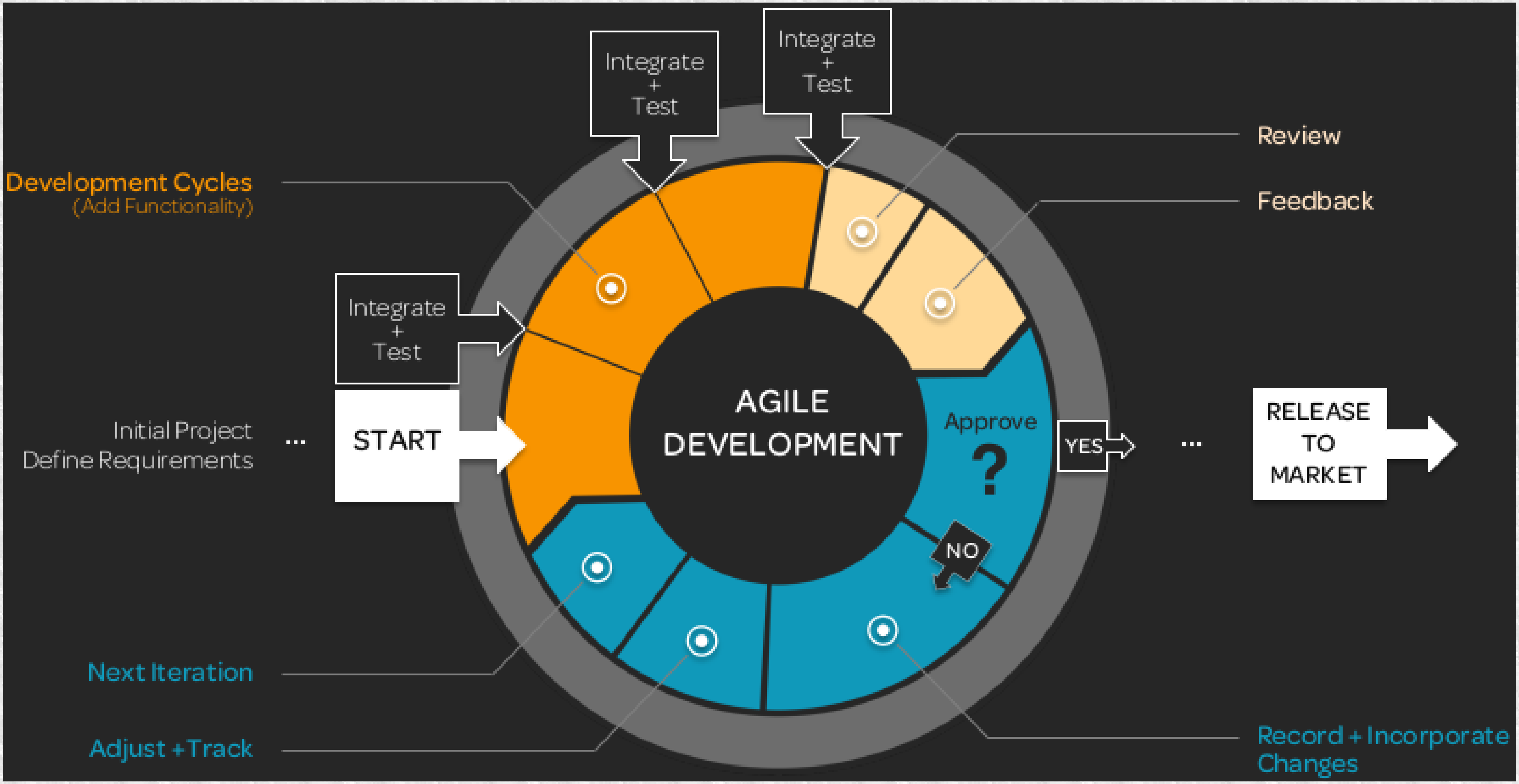
- Software development methodology.
- A set of values and principles
- Adaptive planning
- Evolutionary and iterative development
- Early delivery
- Continuous improvement
- Rapid and flexible response to change
- Scrum is the most widely used Agile method.
- Others agile methods and practices :
  -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

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 development



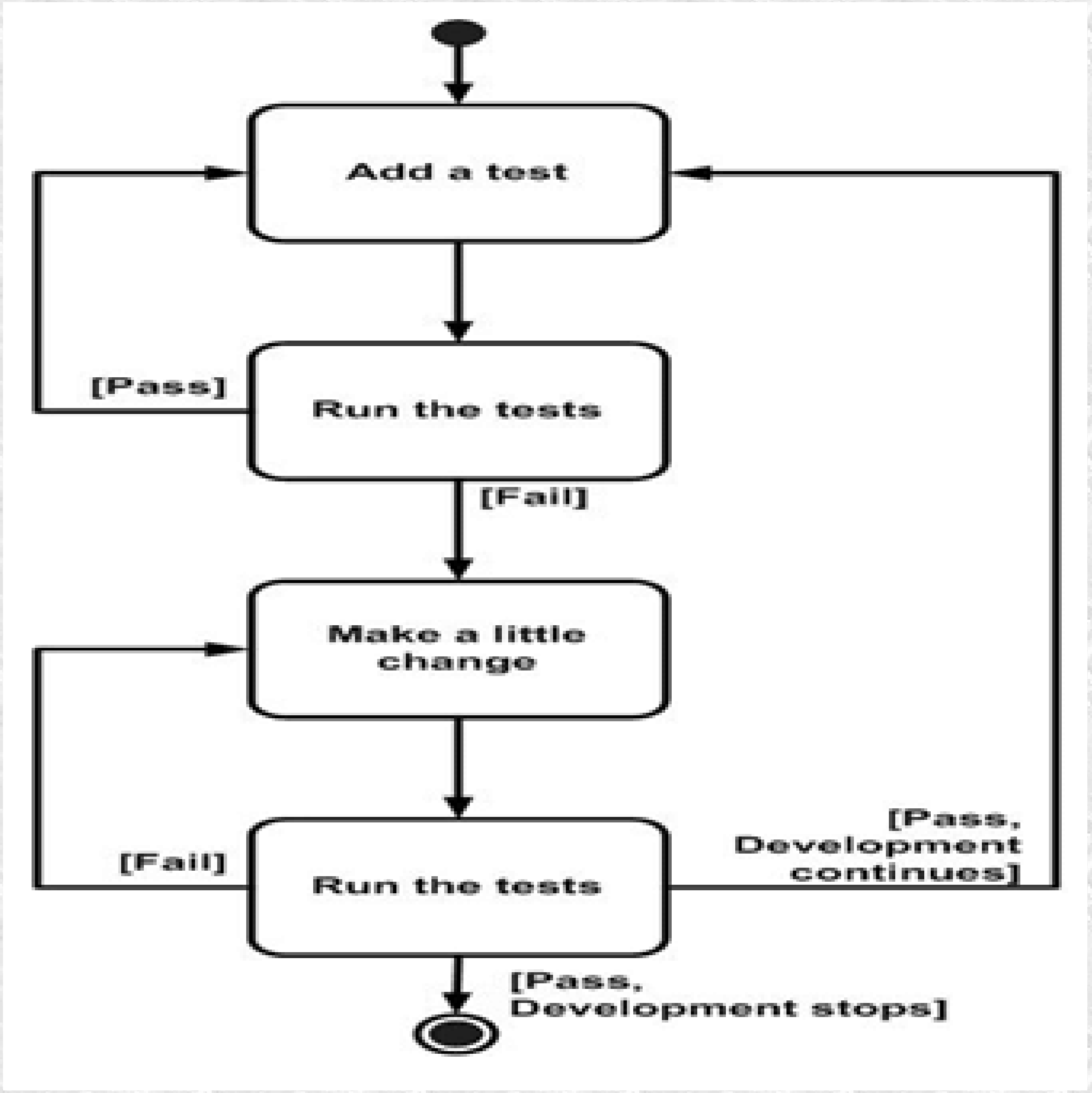
Agile vs DevOps

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

Agile
What is TDD ?

- TDD – Test-Driven Development : A software development process.
- It refers to a style of programming in which three activities are nested:
  - Testing (in the form of writing unit tests).
  - Coding.
  - Refactoring.

Agile
TDD Cycle



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

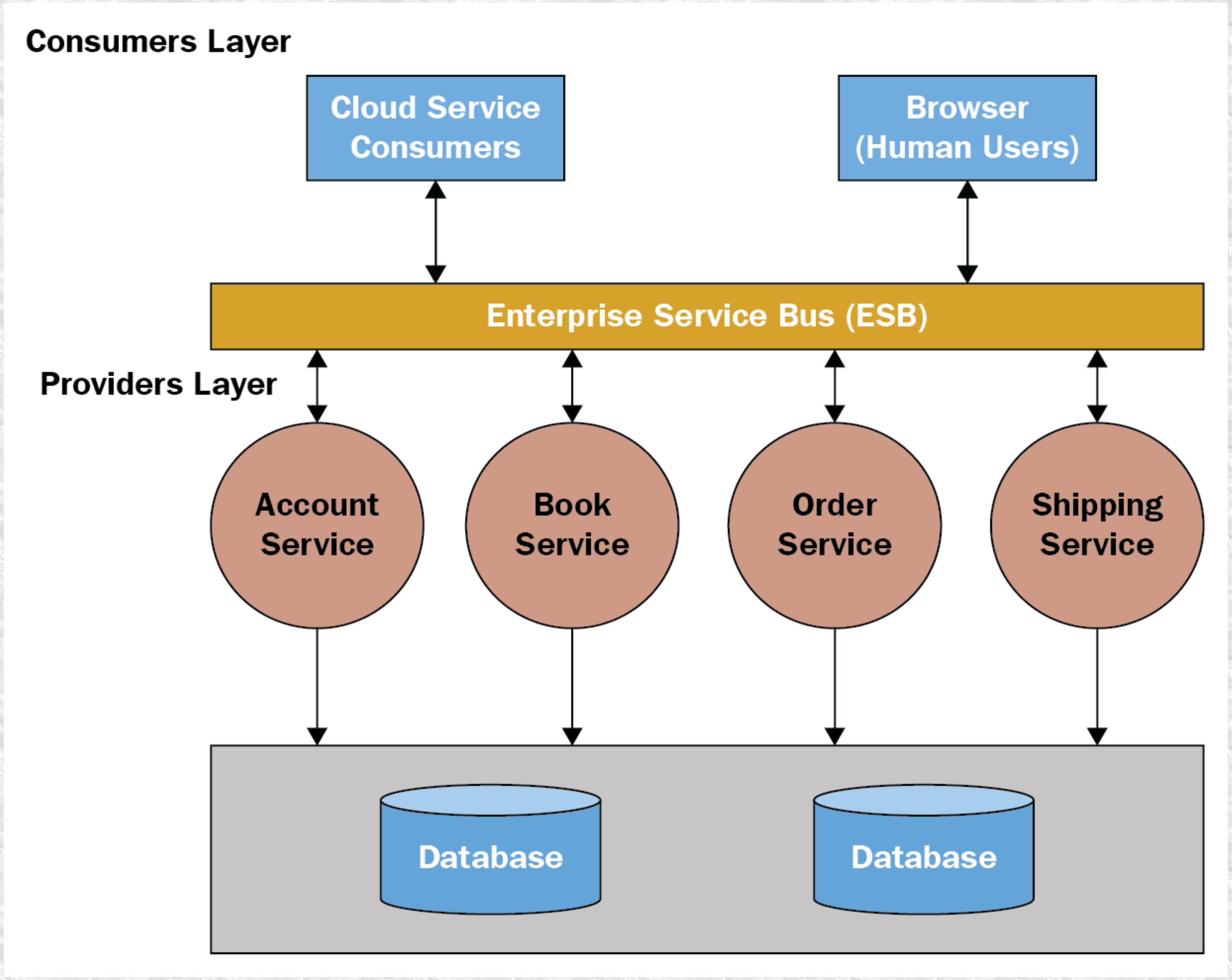
Monolithic vs. SOA vs. Microservices

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

Service based applications
What is SOA ?

- Service Oriented Architecture
- An approach to distributed systems architecture:
  - Loosely coupled services
  - Standard interface and protocol
- Communicates over an enterprise service bus (ESB)

Service based applications
SOA architecture



Service based applications
SOA properties

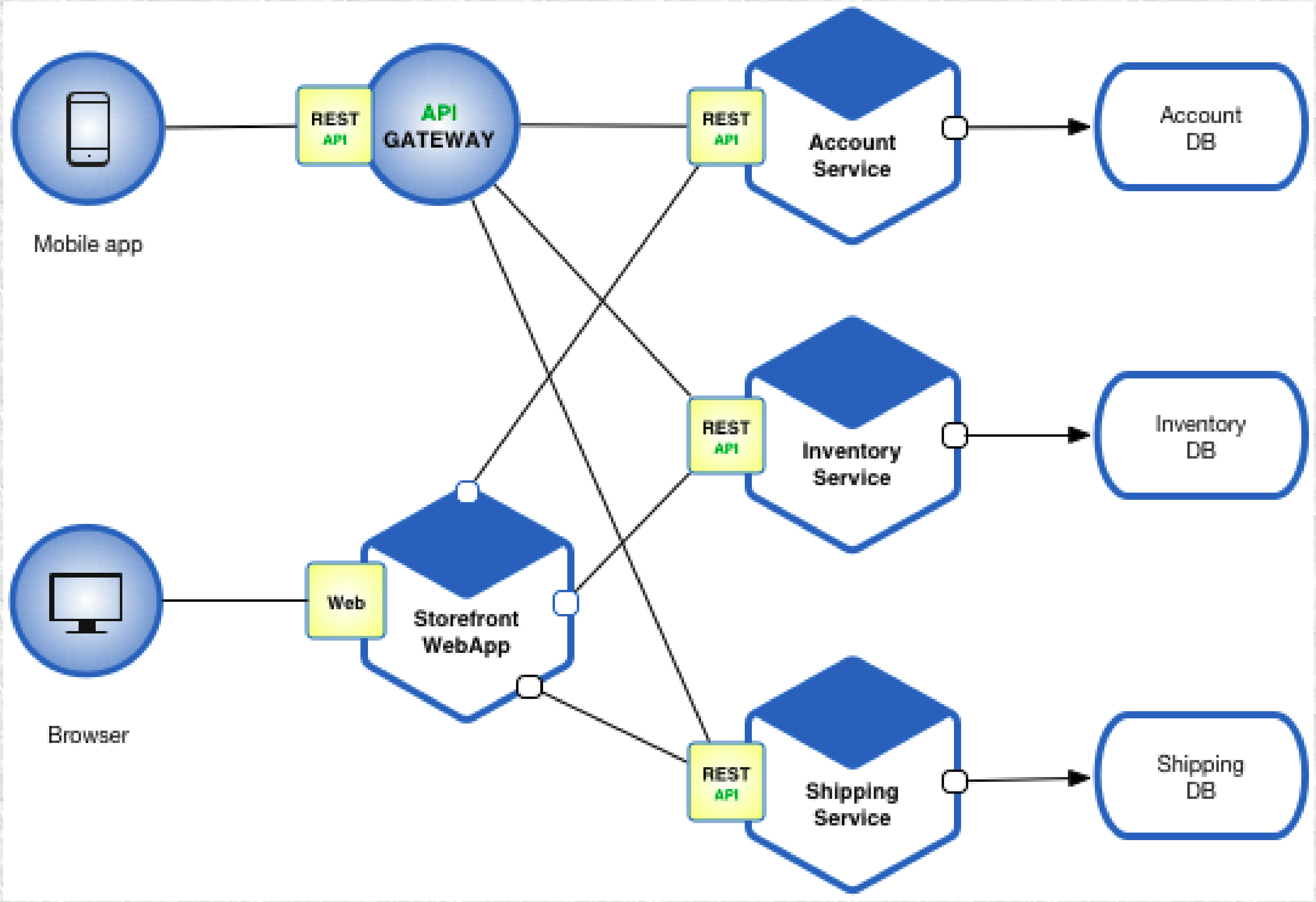
- A service has four properties:
  - It logically represents a business activity with a specified outcome
  - It is autonomous
  - It is a black box for its consumers
  - It may consist of other underlying services

Service based applications
What are Microservices?

- A microservice architecture breaks an application up into a collection of small, loosely-coupled services
- The opposite of monolithic architecture
- Microservices are small
- Microservices are loosely coupled
- Services should be fine-grained
- Protocols should be lightweight

Service based applications

Microservices architecture



Service based applications

What do microservices look like?

- There are many different ways to structure and organize a microservice architecture
- Services are independent:
  - Codebase
  - Running process
  - Built independently
  - Deployed independently
  - Scaled independently

Service based applications

Why use Microservices?

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

Service based applications

Choosing application architecture

- Each architecture has had its utility during its time and might still serve a need.
- 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 including desktop browsers, mobile browsers and native mobile applications.
  - Integrate with other applications via either web services or a message brokers.
  - 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
- Stateless
- 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

- There are 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

- The client sends the type of content that it is able to receive:
  - Accept
  - MIME:
    - application/json
    - application/xml

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"
                    }
}
```

Application security risks
Most security risks

- SQL injection / LDAP injection
- Broken authentication
- Broken access control
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)
- Unvalidated redirects and forwards
- Etc ...

Module 2

Components, platforms and cloud

Plan

- Data platforms and concepts
- PaaS platforms
- Deployment strategies
- OpenStack
- Cloud-init
- Content Delivery Networks

Data platforms and concepts
Relational database

- Based on the relational model of data.
- Relational database systems use SQL.
- Relational model organizes data into one or more tables.
- Each row in a table has its own unique key (primary key).
- MySQL (MariaDB), Oracle, Postgres, etc ...

Data platforms and concepts

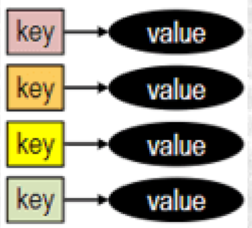
NoSQL database

- Mechanism for storage and retrieval of data other than the tabular relations used in relational databases.
- Increasingly used in big data and real-time web applications
- Properties :
  - Simplicity of design
  - Simpler scaling to clusters of machines (problem for relational databases)
  - Finer control over availability.
  - Some operations faster (than relational DB)

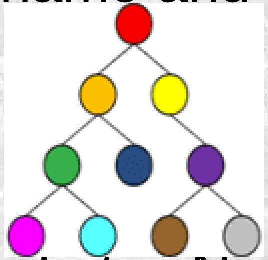
Data platforms and concepts

Types of NoSQL database

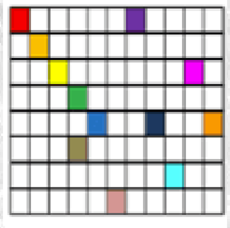
- Key-value :



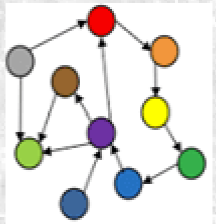
  - Examples : Redis, Dynamo and Memcached
- Document-oriented :



  - Examples : MongoDB, Lotus Notes, Amazon SimpleDB
- Column-based :



  - Examples : Hbase, Cassandra, Hypertable
- Graph-based :



  - Examples : Neo4J, Infinite Graph

Data platforms and concepts
Object storage

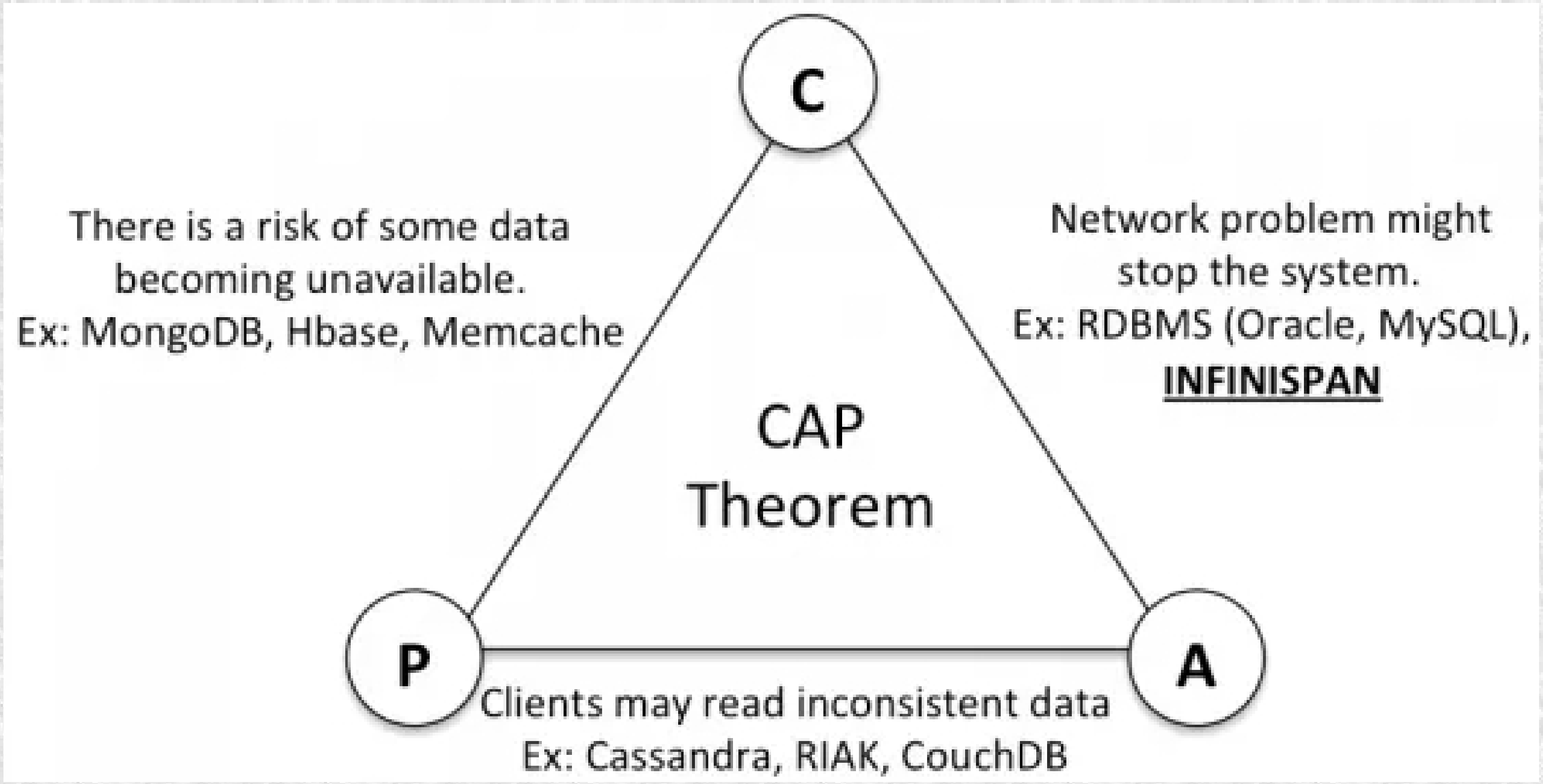
- Manages data as objects
- Opposed to other storage architectures :
  - File systems : manages data as a file hierarchy
  - Block storage : manages data as blocks
- Each object typically includes :
  - The data itself,
  - Metadata (additional informations)
  - A globally unique identifier.
- can be implemented at multiple levels :
  - device level (SCSI device, etc ...)
  - system level (used by some distributed file systems)
  - cloud level (Openstack swift, AWS S3, Google Cloud Storage)

Data platforms and concepts
CAP theorem

- CAP : Consistency, Availability and Partition-tolerance.
- It is impossible for a distributed data store to simultaneously provide more than two out of the three guarantees :
  - Consistency : Receive the same information, regardless the node that process the order.
  - Availability : the system provides answers for all requests it receives, even if one or more nodes are down.
  - Partition-tolerance : the system still Works even though it has been divided by a network failure.

Data platforms and concepts

CAP theorem

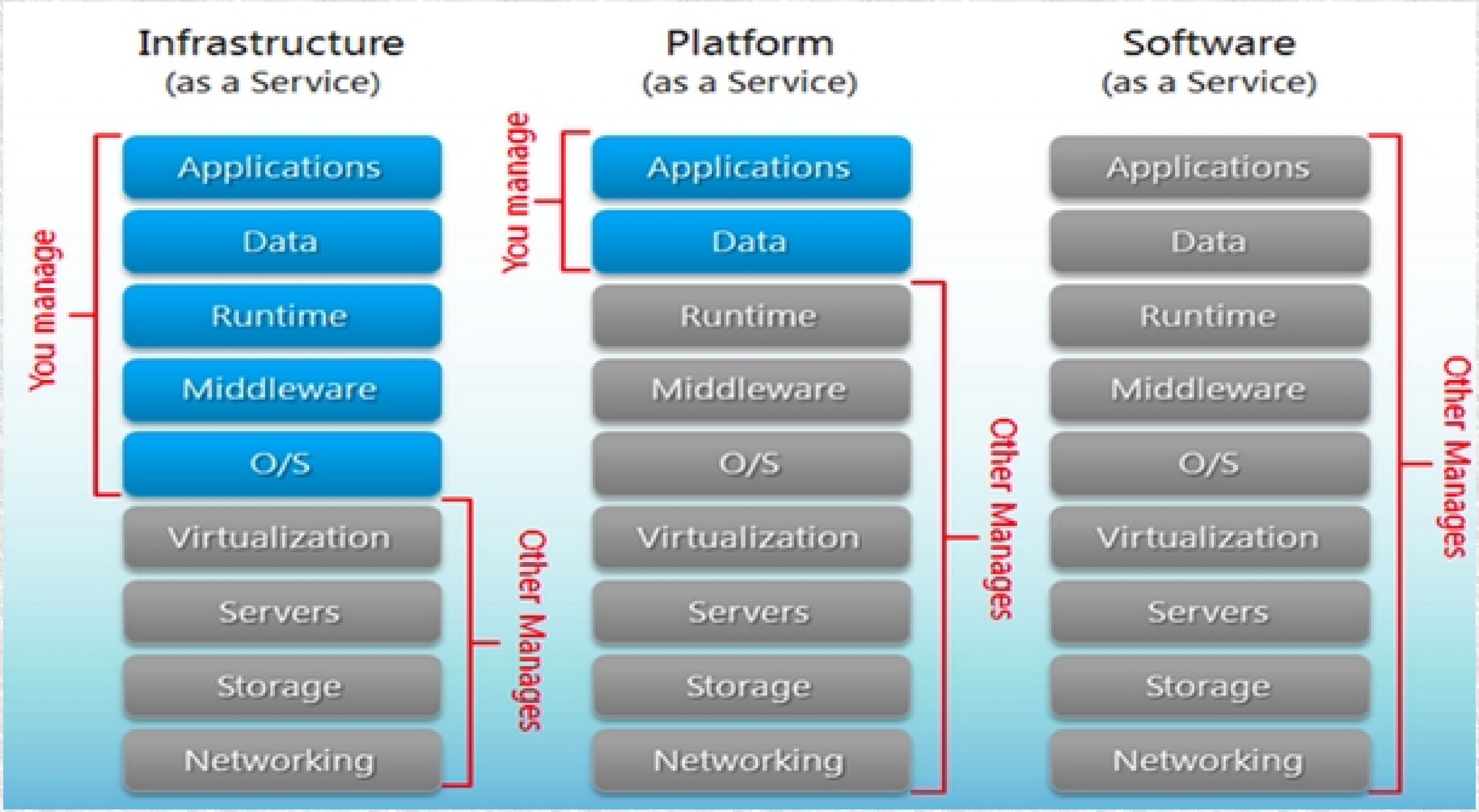

C
There is a risk of some data becoming unavailable.
Ex: MongoDB, Hbase, Memcache
Network problem might stop the system.
Ex: RDBMS (Oracle, MySQL), INFINISPAN
CAP Theorem
P
A
Clients may read inconsistent data
Ex: Cassandra, RIAK, CouchDB

Data platforms and concepts
ACID properties

- ACID : Atomicity, Consistency, Isolation and Durability.
- Set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc ...
  - Atomicity : each transaction is treated as a single "unit", which either succeeds completely, or fails completely.
  - Consistency (integrity): Ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants (only starts what can be finished).
  - Isolation: two or more transactions made at the same time must be independent and do not affect each other.
  - Durability: If a transaction is successful, it will persist in the system (recorded in non-volatile memory)

Paas platforms

Cloud services



PaaS platforms
cloud PaaS software

- AWS Lambda
- Plesk
- Google Cloud Functions
- Azure Web Apps
- Oracle Cloud PaaS
- OpenShift
- Cloud Foundry
- Etc ...

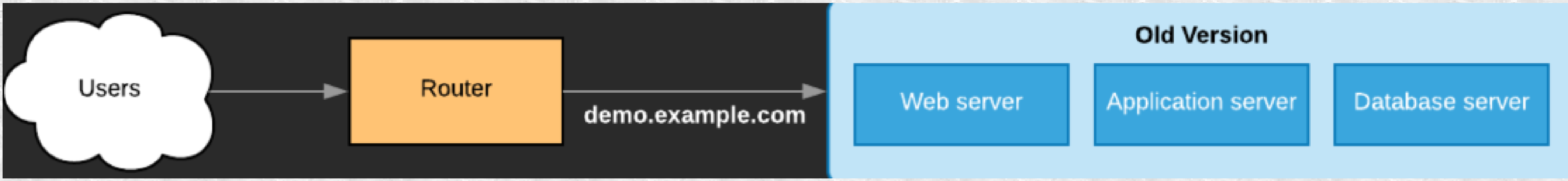
Deployment strategies

Popular Deployment Strategies

- Application and infrastructure teams should devise and adopt a deployment strategy suitable for their use case.
- The most poplular deployment strategies :
  - "Big Bang" Deployment : the full solution is developed and tested and then replaces the current system at once.
  - Rolling Deployment : An application’s new version gradually replaces the old one.
  - Blue-Green Deployment : Two identical production environments work in parallel.
  - A/B Testing : comparing two version and measuring the resultant responses.
  - Canary Deployment : deploying an application in small, incremental steps, and only to a small group of people.

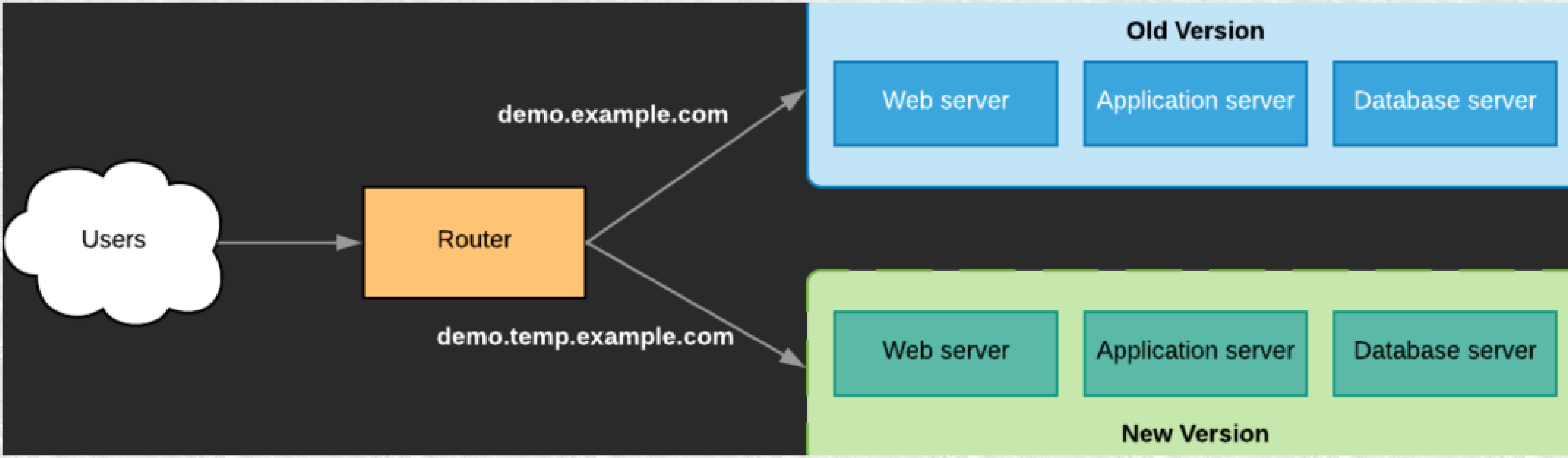
Deployment strategies

Blue-Green deployment



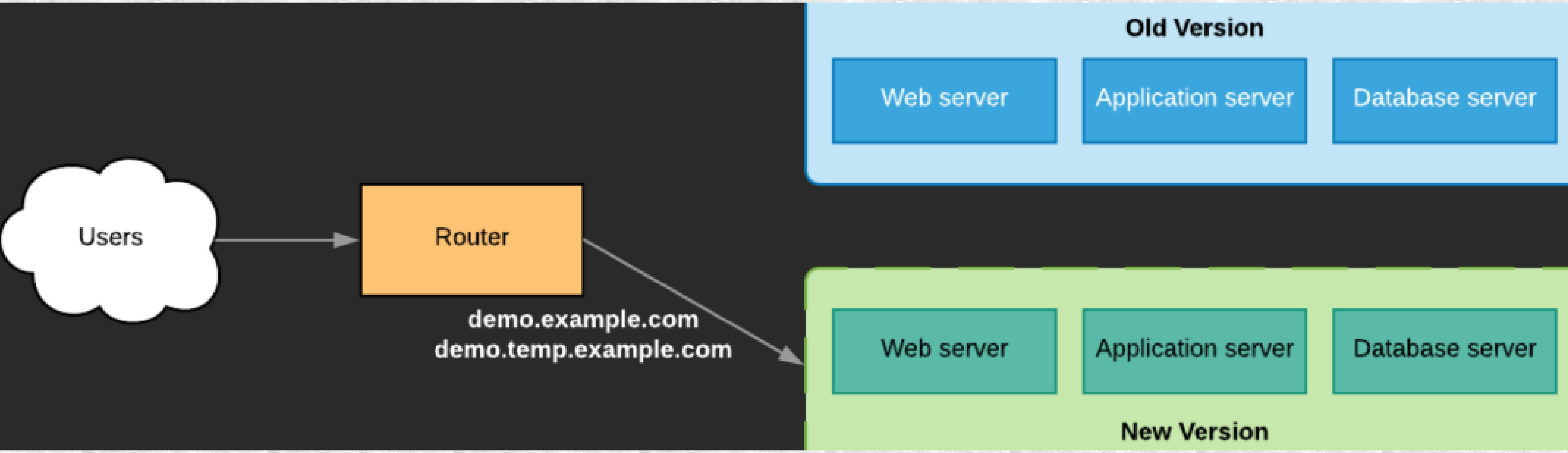
Deployment strategies

Blue-Green deployment



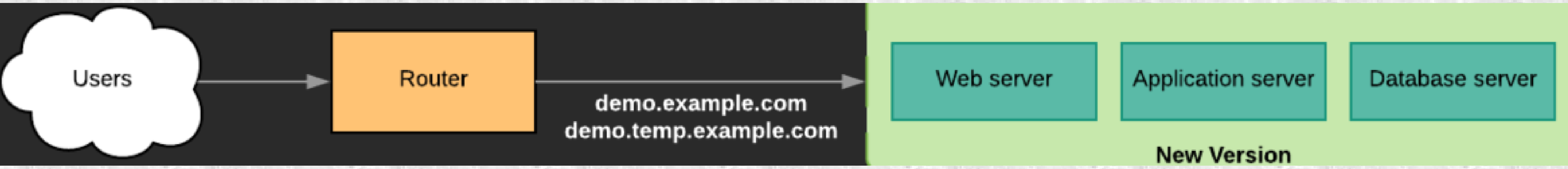
Deployment strategies

Blue-Green deployment



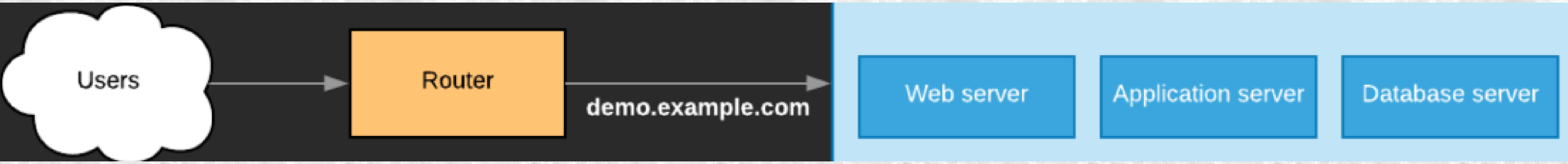
Deployment strategies

Blue-Green deployment



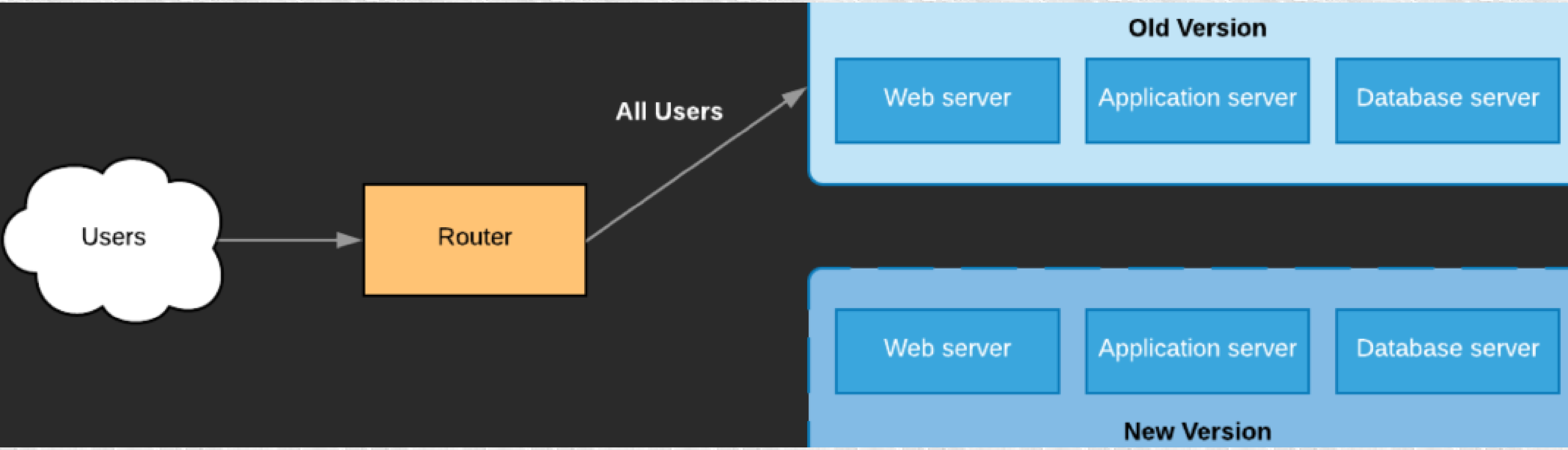
Deployment strategies

Blue-Green deployment

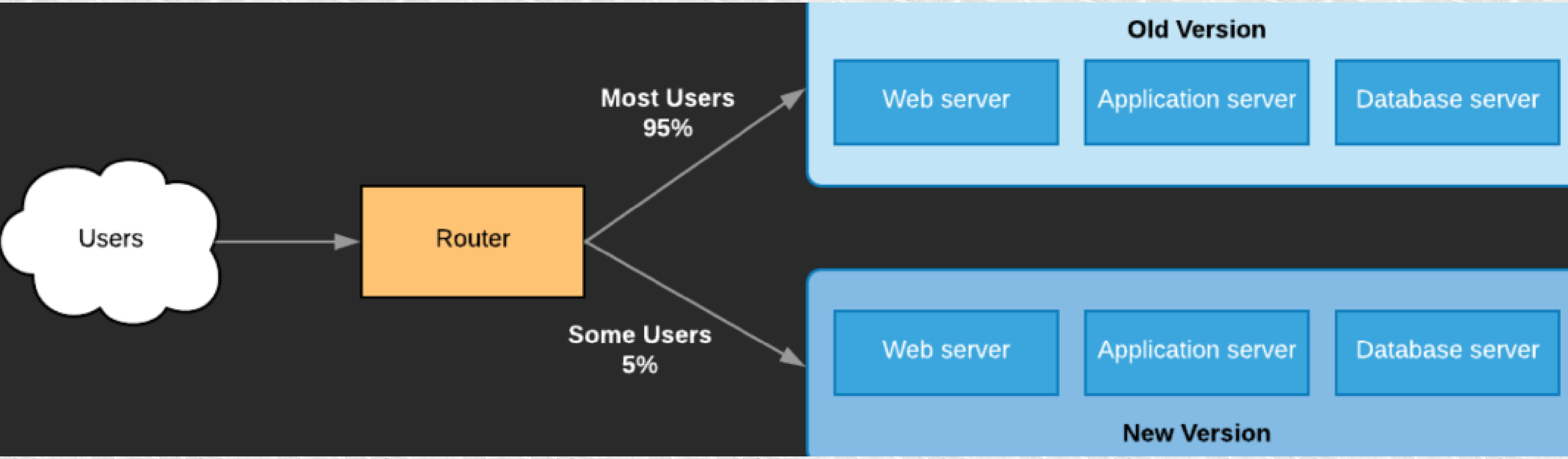


Deployment strategies

Canary deployment

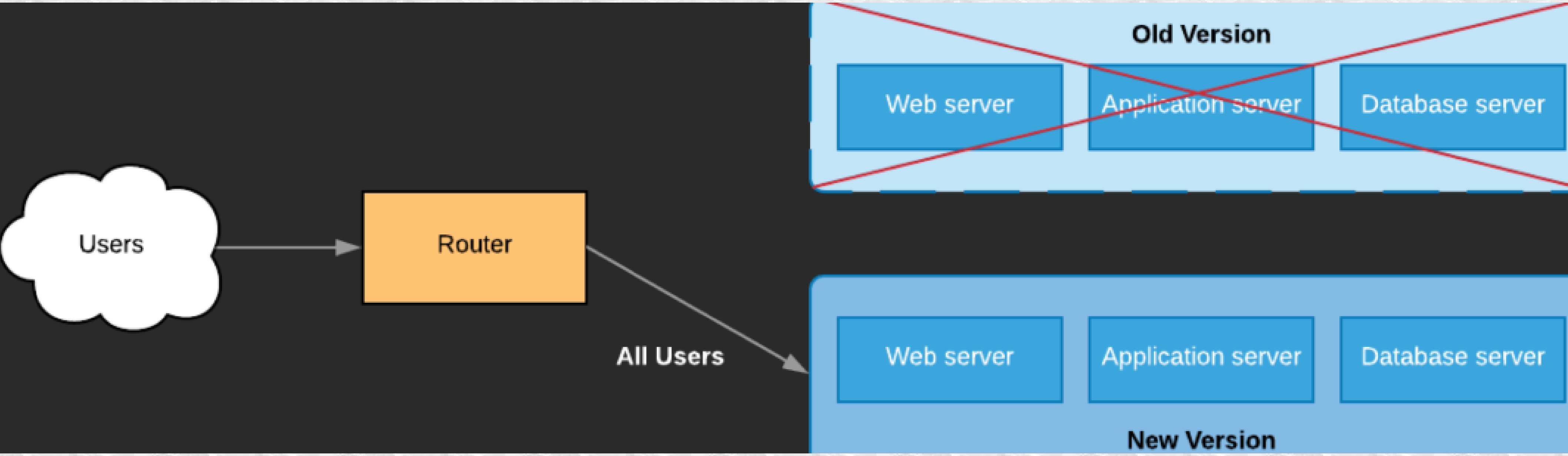


Deployment strategies
Canary deployment



Deployment strategies

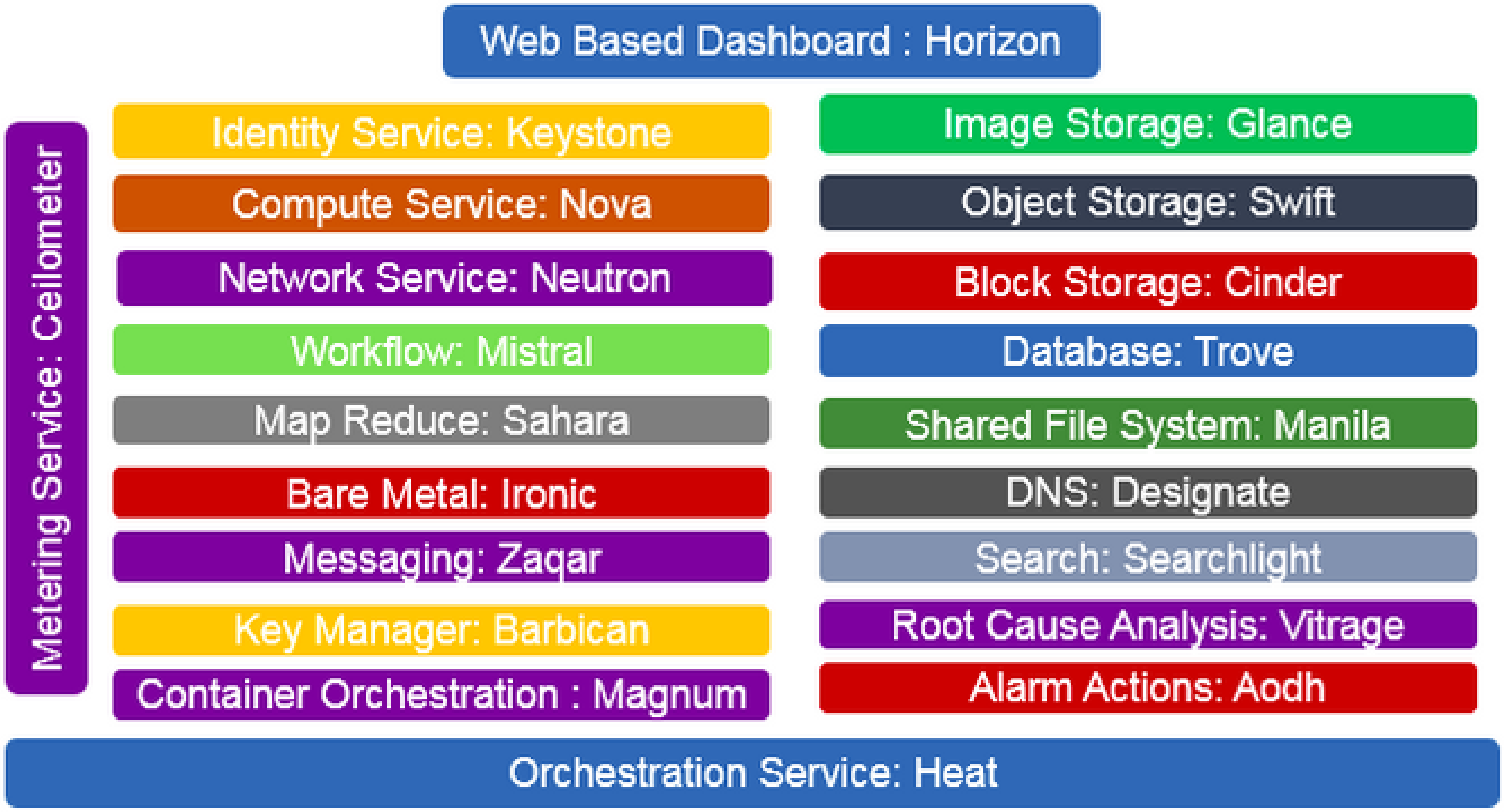
Canary deployment



Openstack
Presentation

- Open-source software platform for cloud computing, mostly deployed as IaaS.
- 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.
- Managed through a web-based dashboard, command-line tools, or RESTful API.

Openstack
Components



LPI DevOps Tools Engineers

Module 3

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
- SCM involves tracking the modifications to code.
- Tracking modifications assists development and colloaboration by :
  - Providing a running history of development
  - helping to resolve conflicts when merging contributions from multiple sources.
- 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"

SCM solutions
SCM types

- Two types of version control: centralized and distributed.
- 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 developer'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

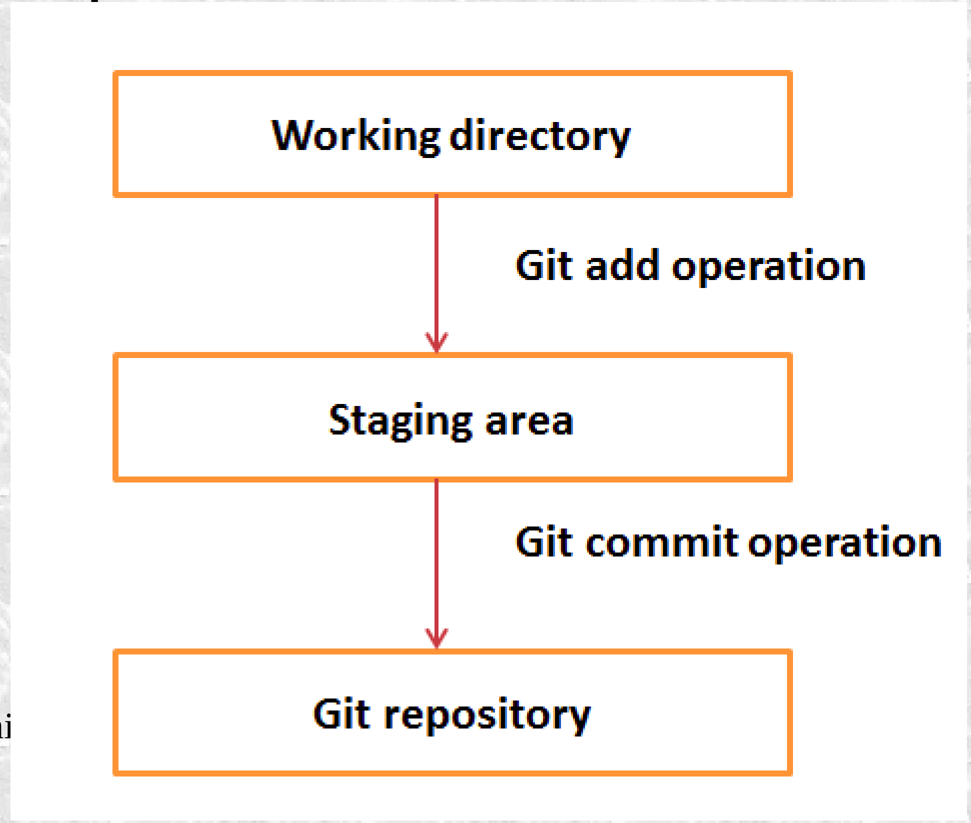
Git repository

- Local Repository : Typically is on developer’s computer.
  - Developer 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

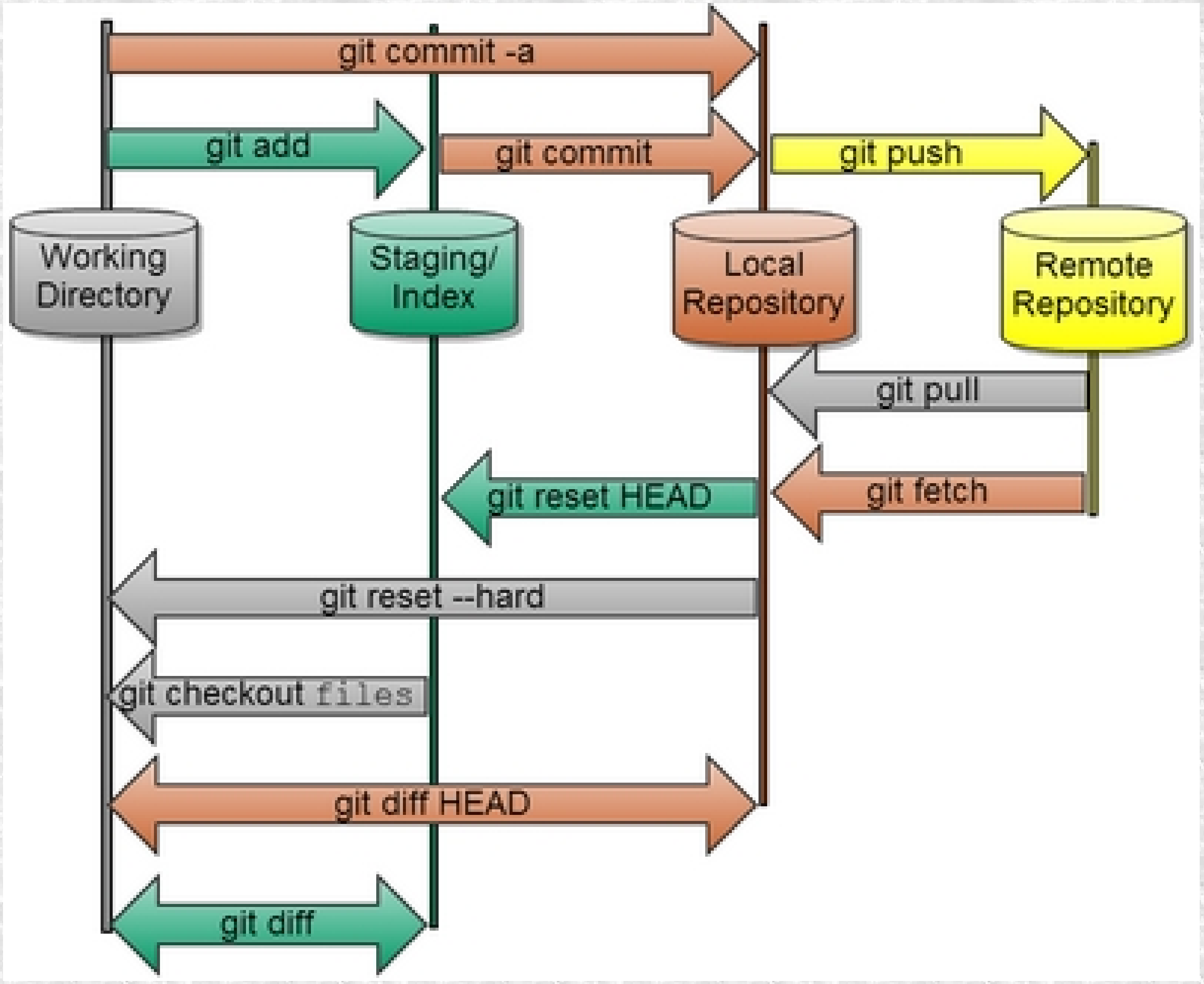
Working Directory and Staging Area or Index

- 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 commands

main commands



Git merge conflict

Understanding merge conflicts

- Conflicts generally arise when :
  - Two people have changed the same lines in a file
  - If one developer deleted a file while another developer was modifying it.
- Git cannot automatically determine what is correct.
- Conflicts only affect the developer 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 developers' responsibility to resolve the conflict.

Git merge conflict

Types of merge conflicts

- A merge can enter a conflicted state at two separate points.
  - Git fails to start the merge :
    - A merge will fail to start when Git sees there are changes in either the working directory or staging area of the current project.
  - Git fails during the merge :
    - A failure DURING a merge indicates a conflict between the current local branch and the branch being merged
    - This indicates a conflict with another developers code.

LPI DevOps Tools Engineers

Module 4

System image creation and VM Deployment

Plan

- Vagrant
- Vagrantfile
- Vagrantbox
- Packer

Vagrant

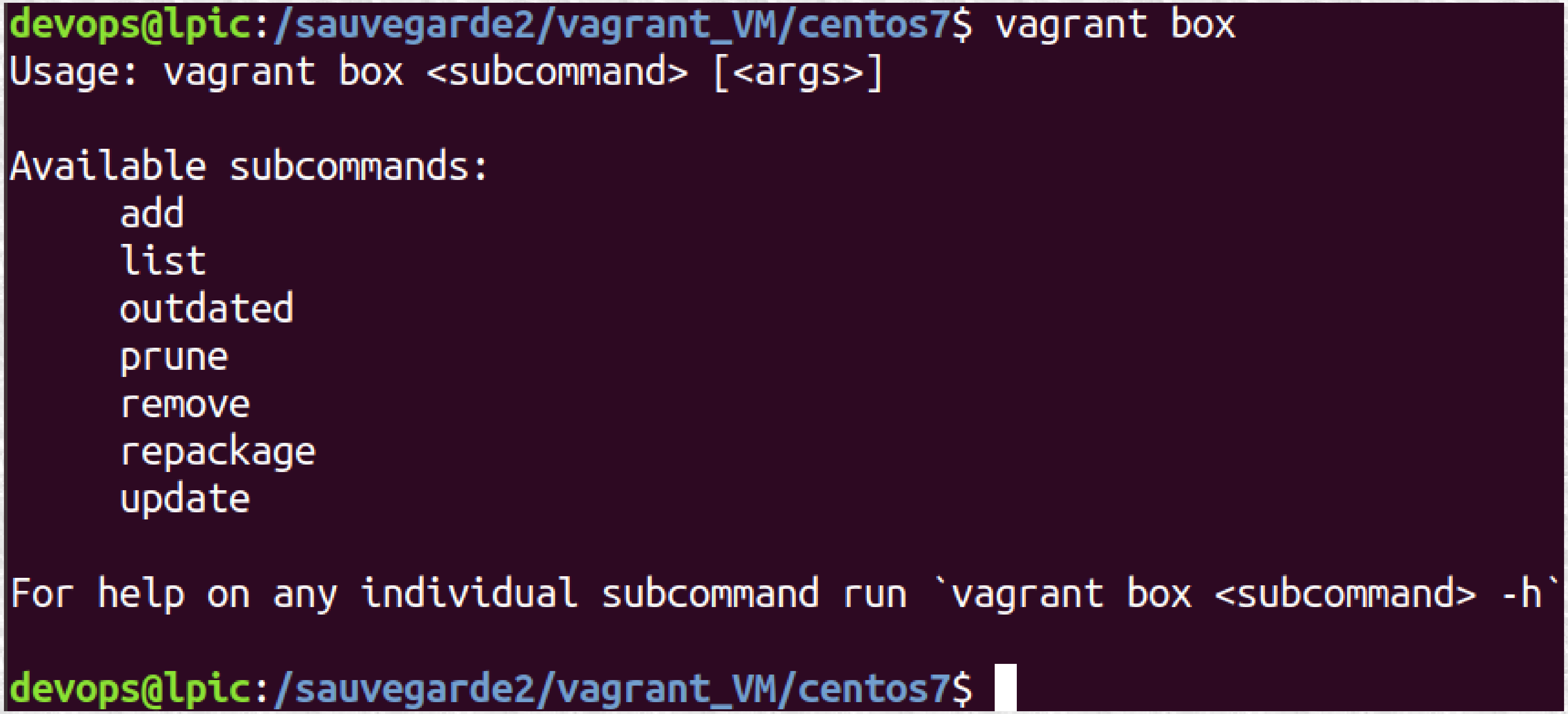
What’s vagrant

- 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 Vagrantbox is a tarred, gzip 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 (For Virtualbox)
  - the virtual machine image.
- box.ovf
  - defines the virtual hardware for the box.
- metadata.json
  - tells vagrant what provider the box works with.

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

What is Packer

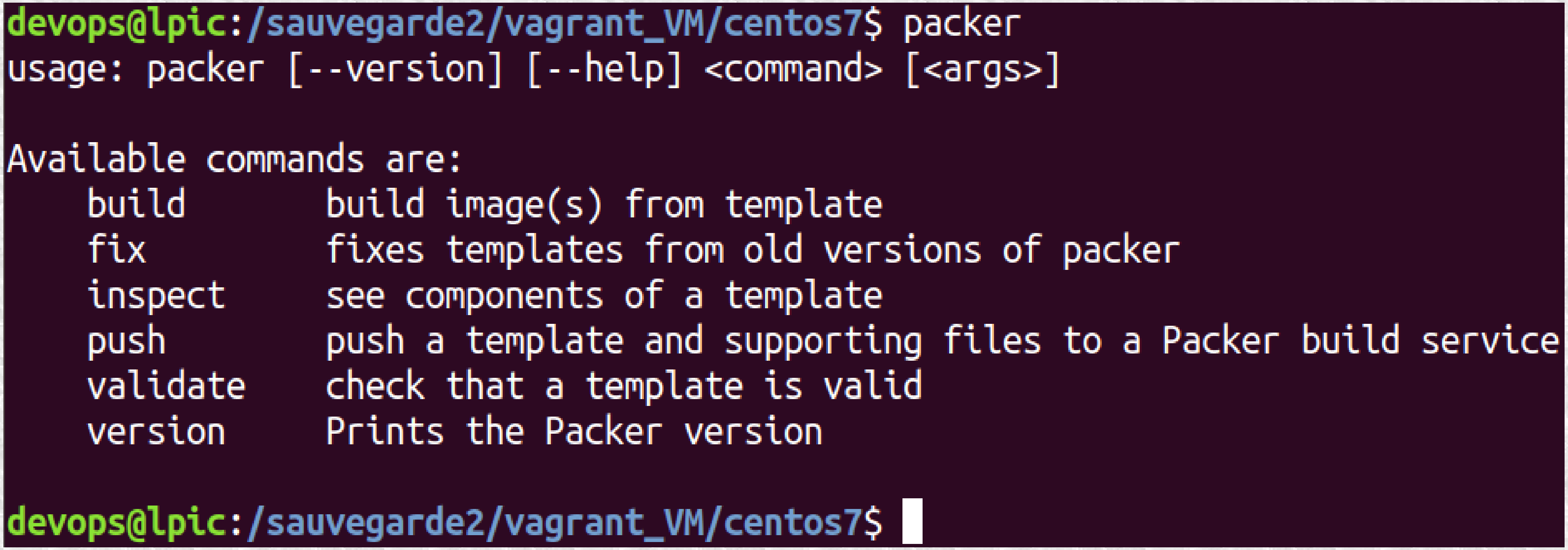
- Open source tool for creating identical machine images :
  - for multiple platforms
  - from a single source configuration.
- Advantages of using Packer :
  - Fast infrastructure deployment
  - Multi-provider portability
  - Stability
  - Identicality

Packer
Use cases

- Continuous Delivery
  - Generate new machine images for multiple platforms on every change to Ansible, Puppet or Chef repositories
- Environment Parity
  - Keep all dev/test/prod environments as similar as possible.
- Auto-Scaling acceleration
  - Launch completely provisioned and configured instances in seconds, rather than minutes or even hours.

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

- Builders are 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

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

Module 5
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

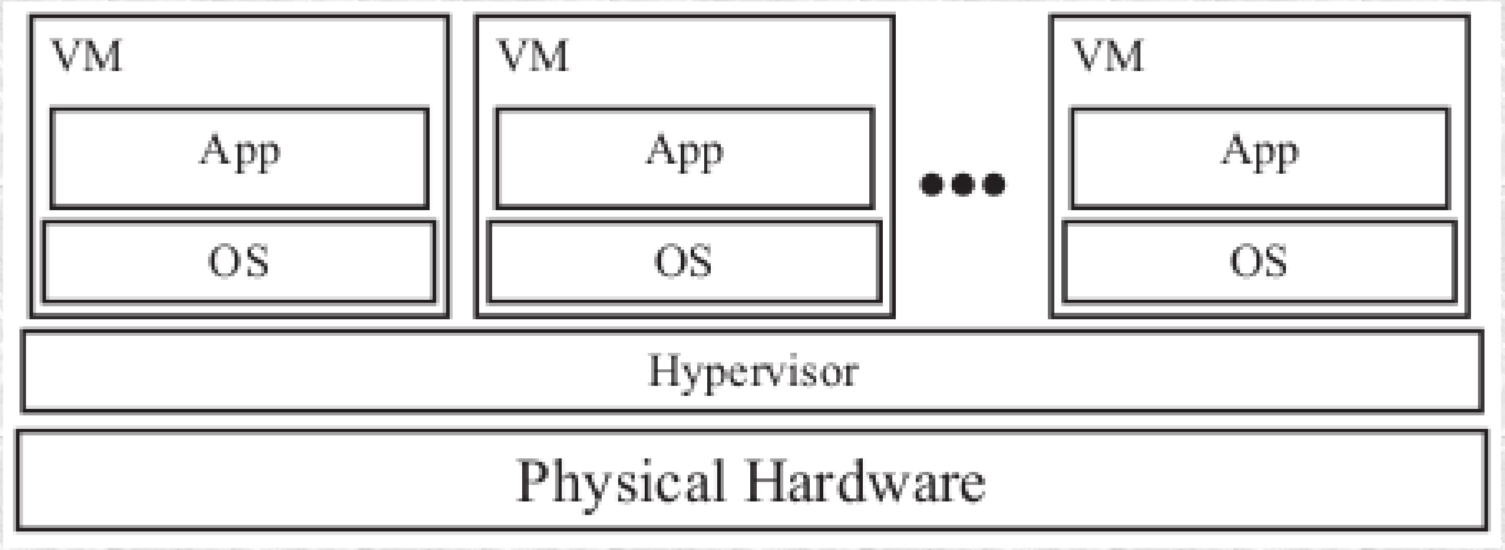
What is a Container and Why?

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?

Problems of Virtualization



- 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?

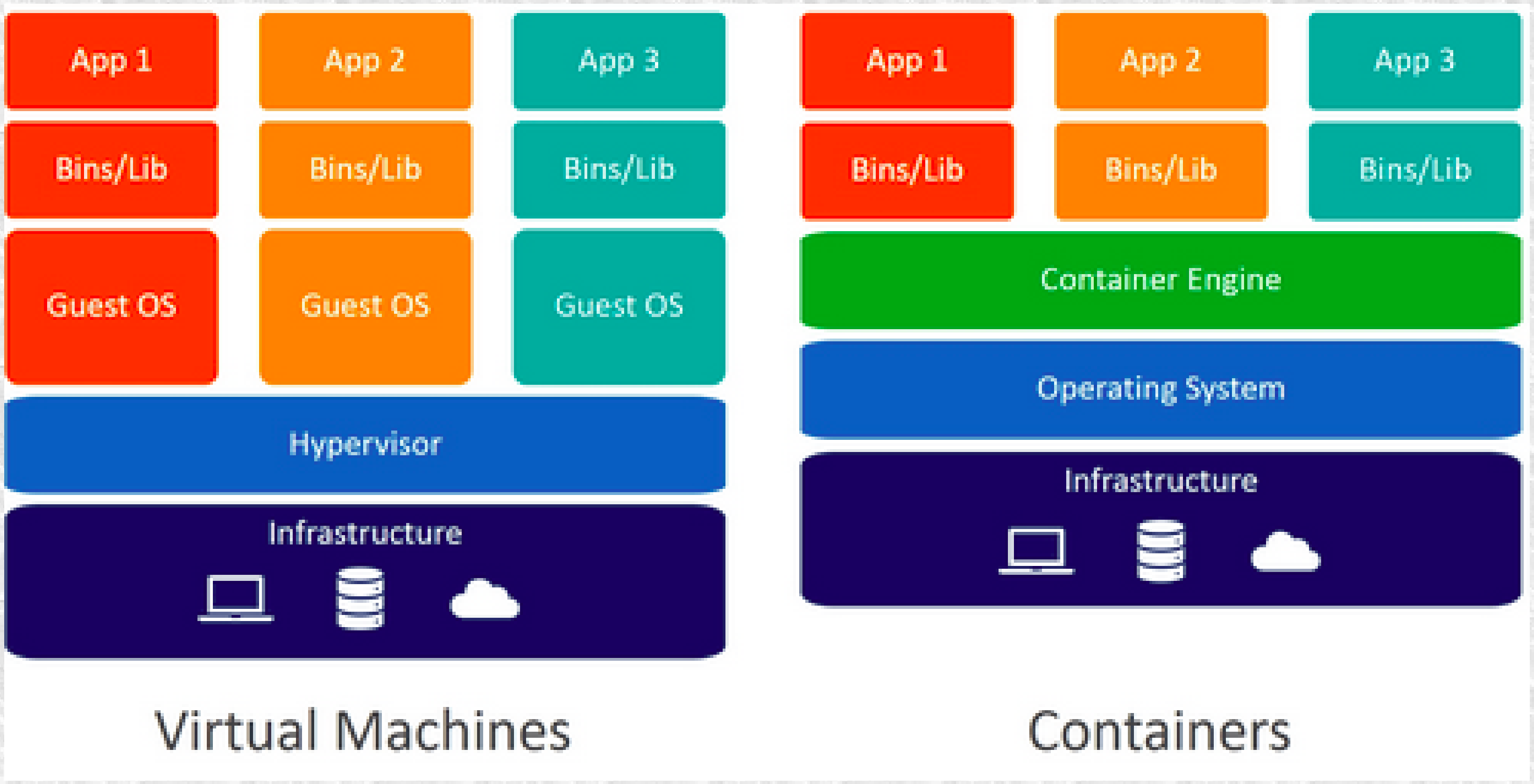
Solution: Containers

- 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



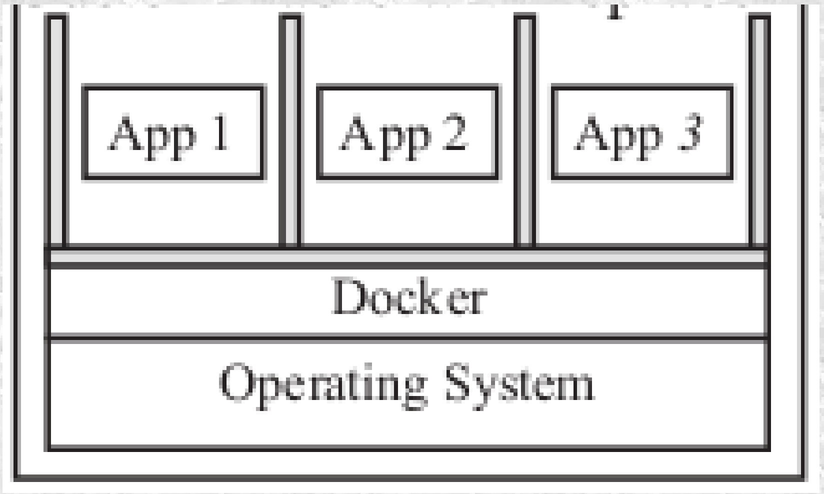
What is a Container and Why?
Containers

- Containers have all the good properties of VMs
  - Come complete with all files and data that you need to run
  - Multiple copies can be run on the same machine or different machines ⇒ Scalable.
  - Same image can run on a personal machine, in a data center or in a cloud.
  - Isolation: For example, “Show Process” (ps on Linux) command in a container will show only the processes in the container.
  - Can be stopped. Saved and moved to another machine or for later run.
  - Can be saved as immuable image

Docker and containers

Docker

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



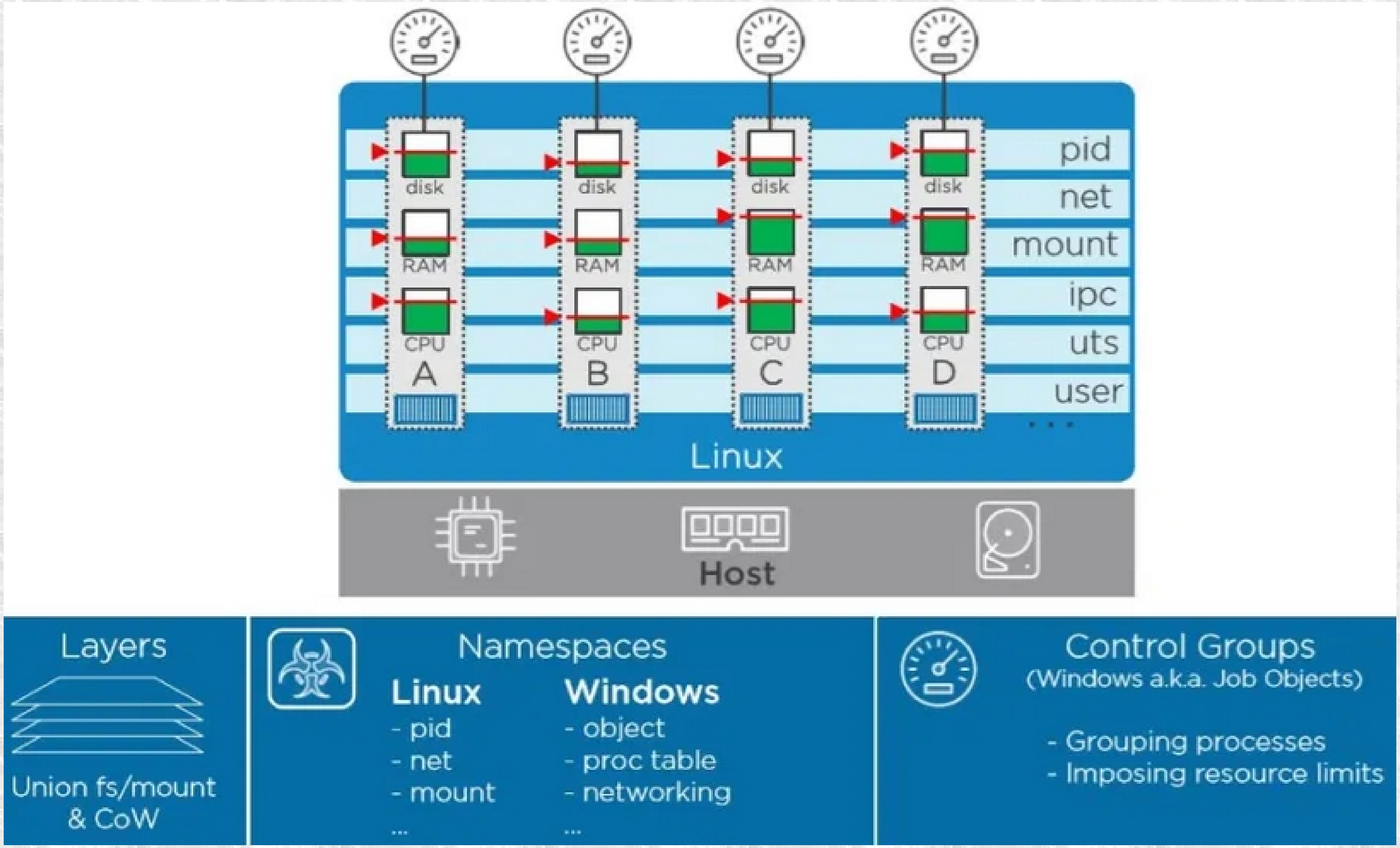
Docker and containers

Docker

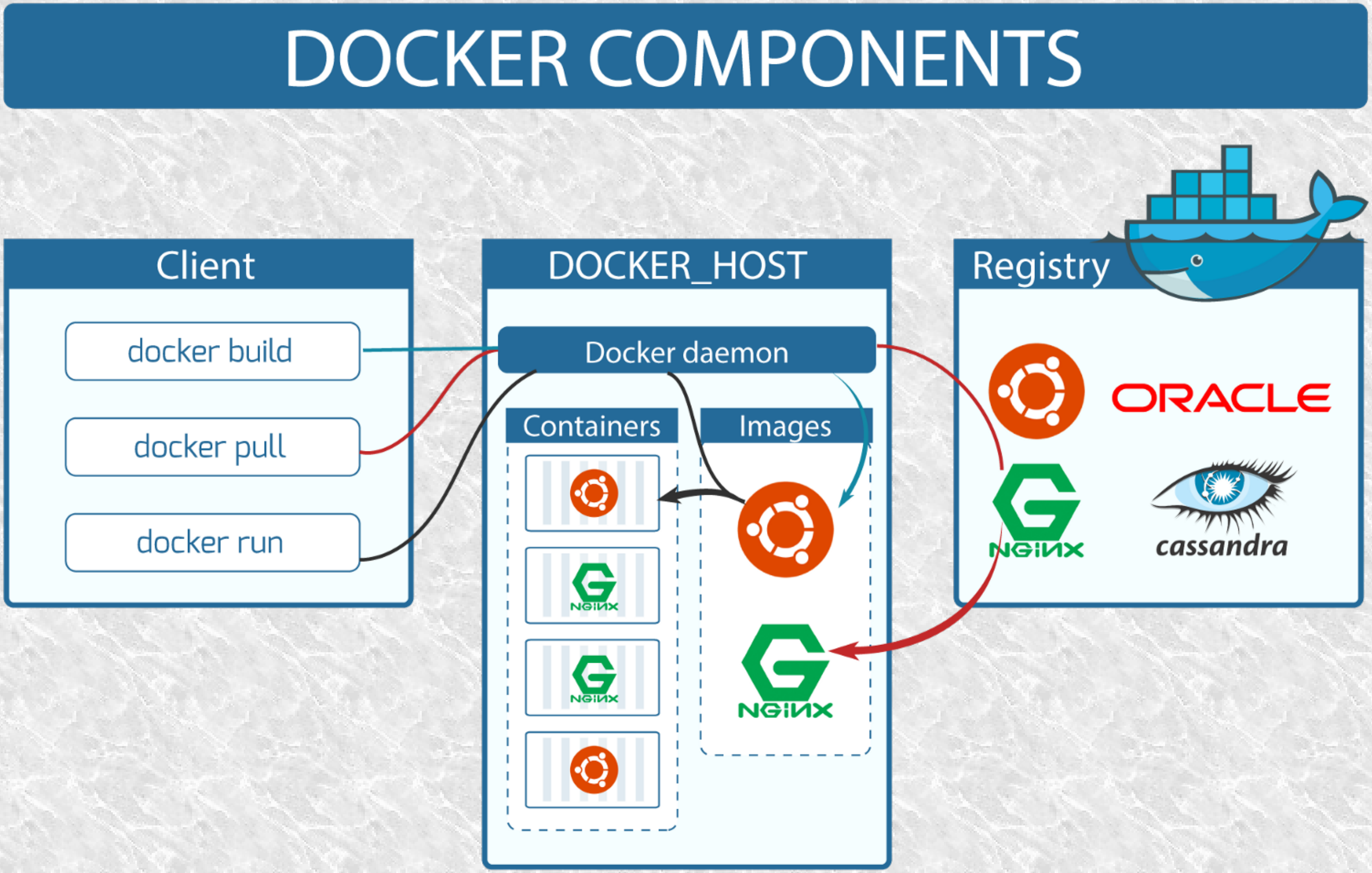
- Docker Engine: Runtime.
- Two Editions:
  - Community Edition (CE): Free for experimentation.
  - Enterprise Edition (EE): For deployment with paid support.
- Written in “Go” programming language from Google.

Docker and containers

Docker container concepts



Docker and containers



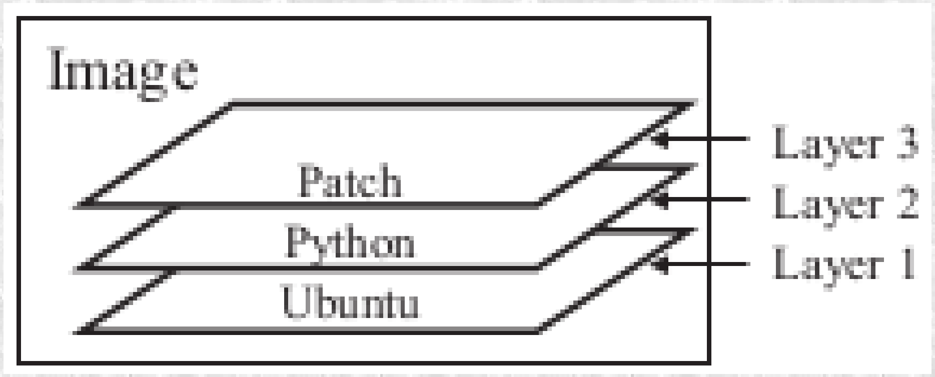
Docker and containers
Image Registries

- Containers are built from images and can be saves 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 type:
  - 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

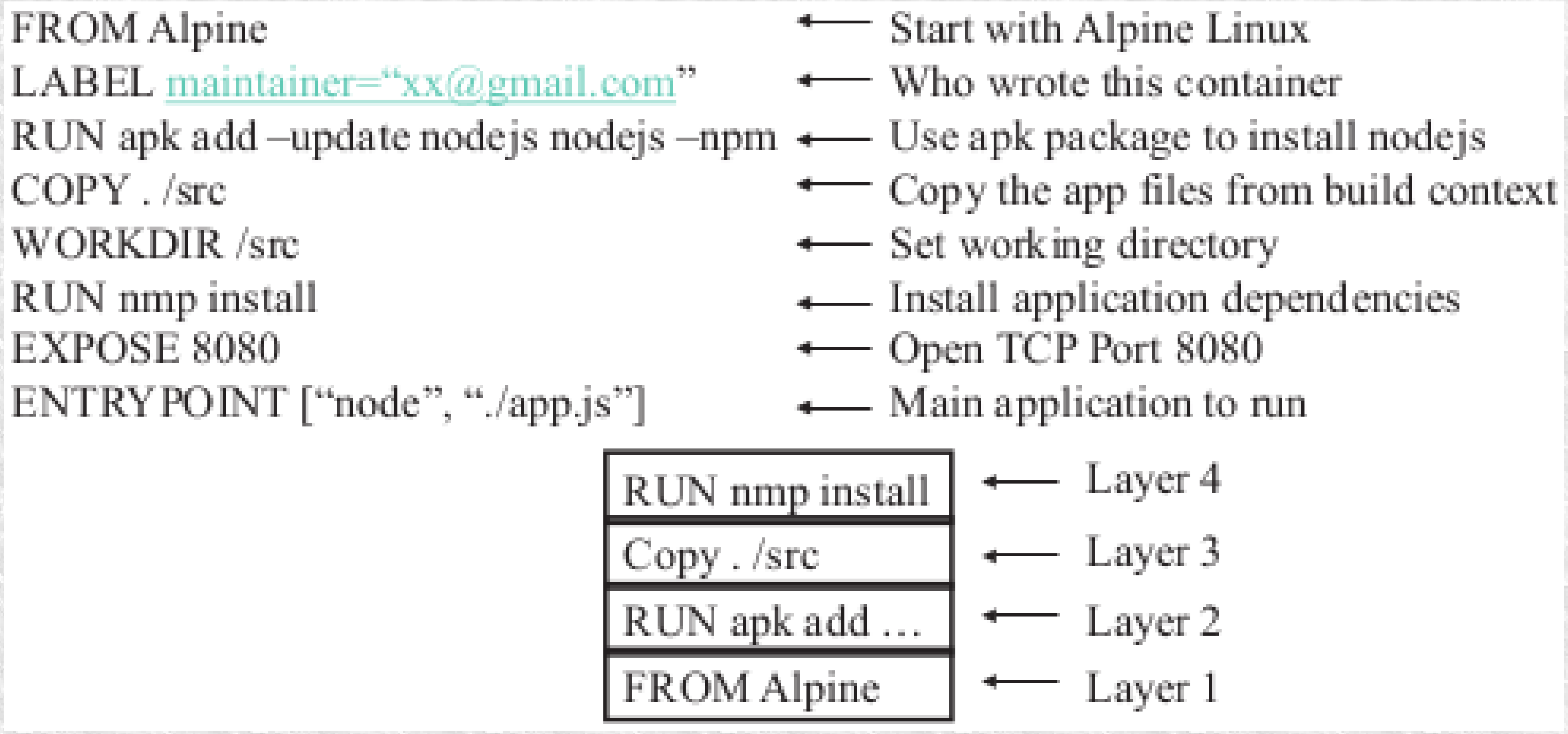


- Layers can be shared among many containers

Docker and containers

Building Container Images

- Create a Dockerfile that describes the application, its dependencies, and how to run it.



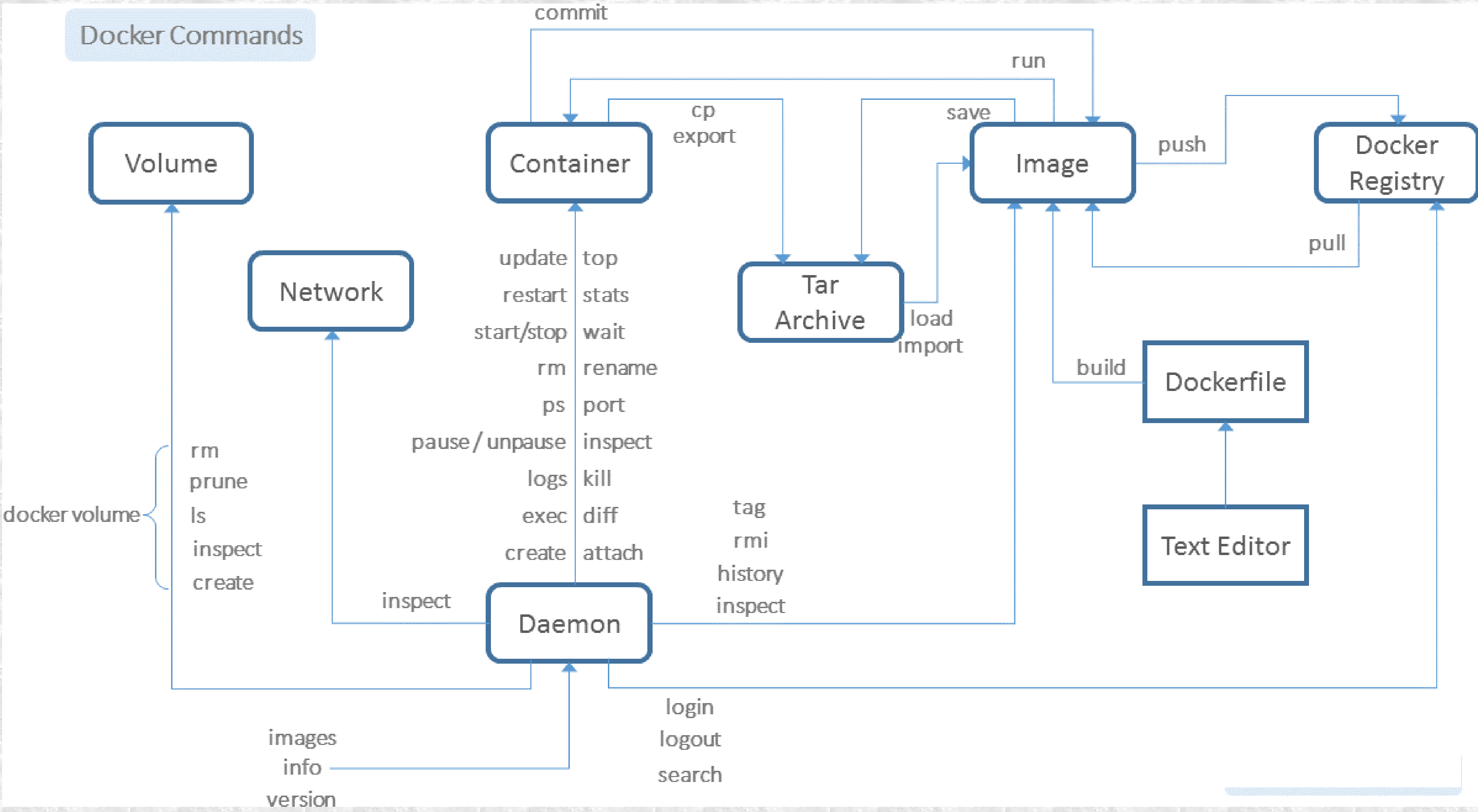
FROM Alpine	←	Start with Alpine Linux
LABEL maintainer="xx@gmail.com"	←	Who wrote this container
RUN apk add –update nodejs nodejs –npm	←	Use apk package to install nodejs
COPY . /src	←	Copy the app files from build context
WORKDIR /src	←	Set working directory
RUN nmp install	←	Install application dependencies
EXPOSE 8080	←	Open TCP Port 8080
ENTRYPOINT ["node", "./app.js"]	←	Main application to run

RUN nmp install	←	Layer 4
Copy . /src	←	Layer 3
RUN apk add …	←	Layer 2
FROM Alpine	←	Layer 1

- WORKDIR, EXPOSE, ENTRYPOINT result in tags. Others in Layers.

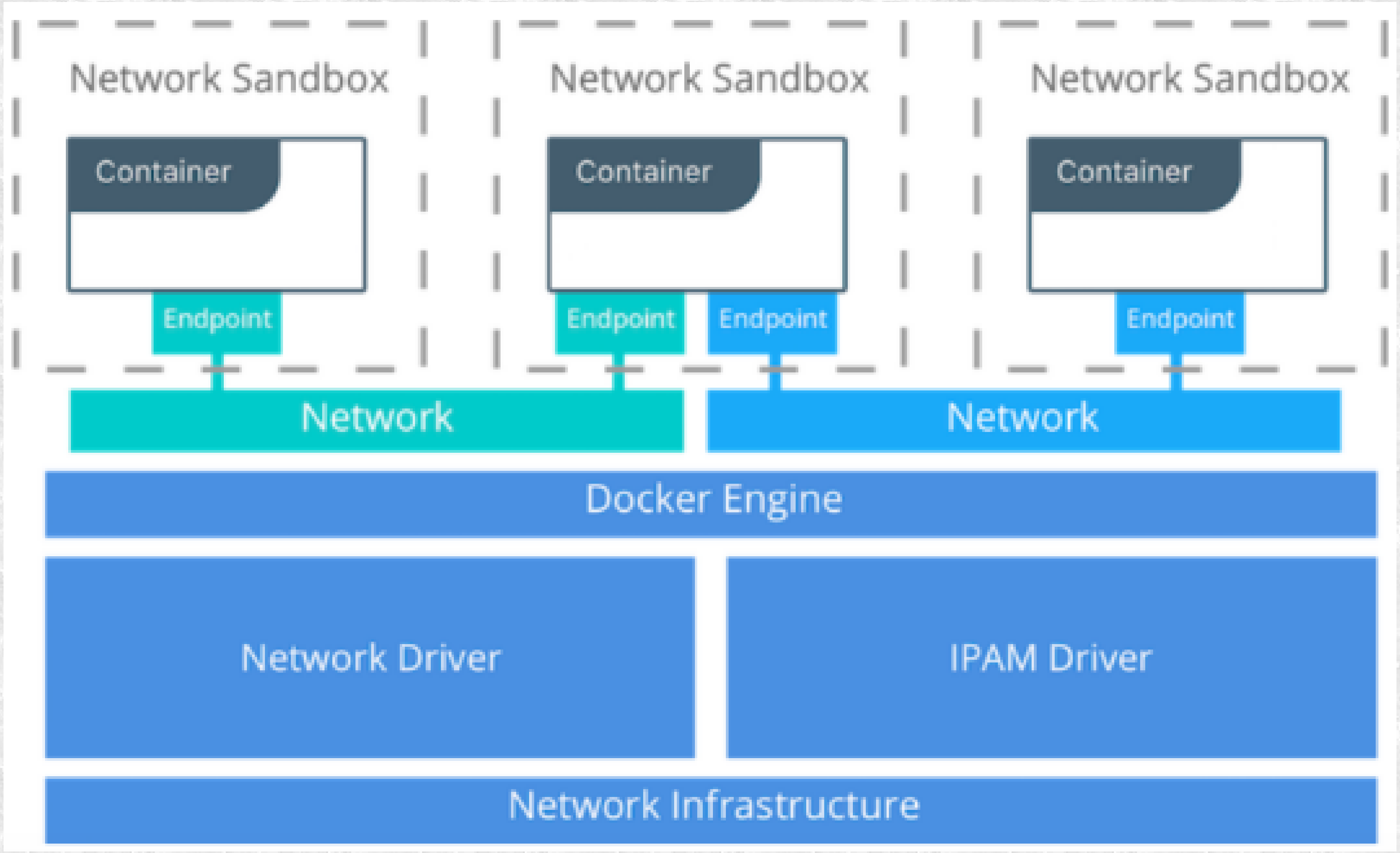
Docker command line

Available commands



Connect container to Docker networks

Container Networking Model (CNM)



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

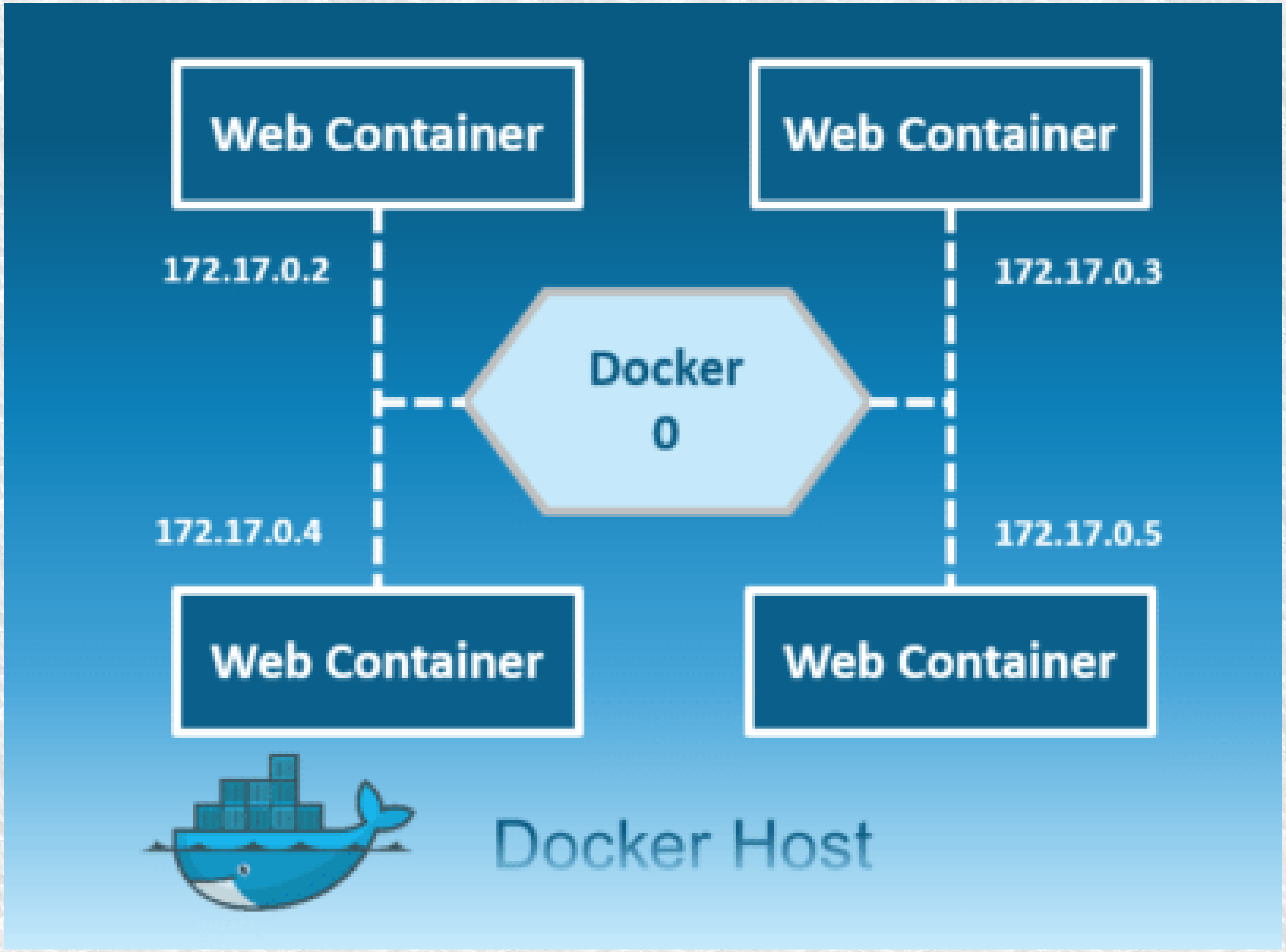
- To list all docker networks

  docker network ls

- 3 pre-defined networks (cannot be removed
- 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.

Connect container to Docker networks

Default bridge network



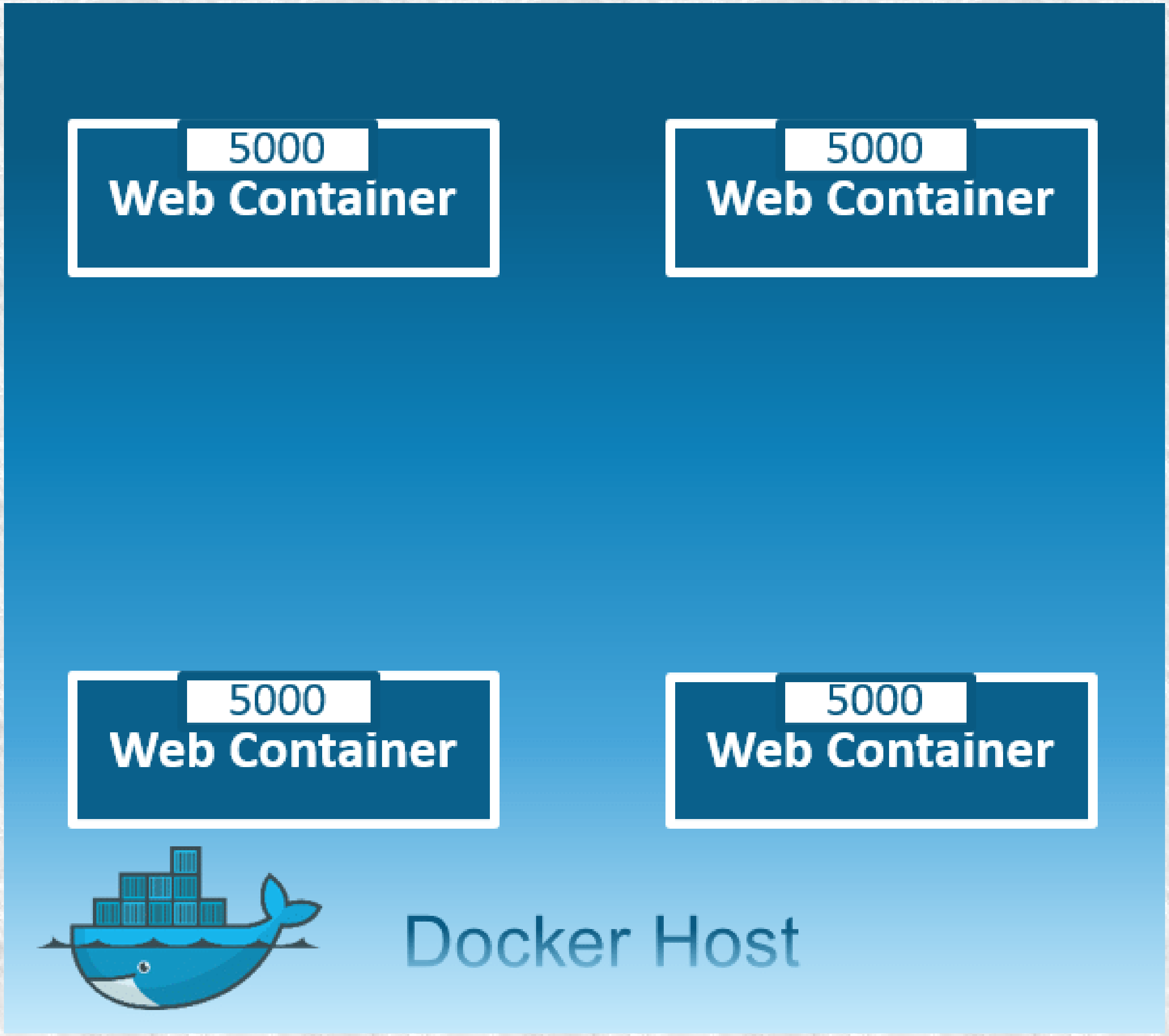
Connect container to Docker networks

User-defined bridge networks

- provide better isolation and interoperability between containerized applications
  - automatically expose all ports to each other
  - no ports exposed to the outside world
- provide automatic DNS resolution between containers.
- Containers can be attached and detached from user-defined networks on the fly.
- Commands :
  - *docker network create my-net*
  - *docker network rm my-net*
  - *docker create --name my-nginx --network my-net --publish 8080:80 \
    nginx:latest*
  - *docker network connect my-net my-nginx*
  - *docker network disconnect my-net my-nginx*

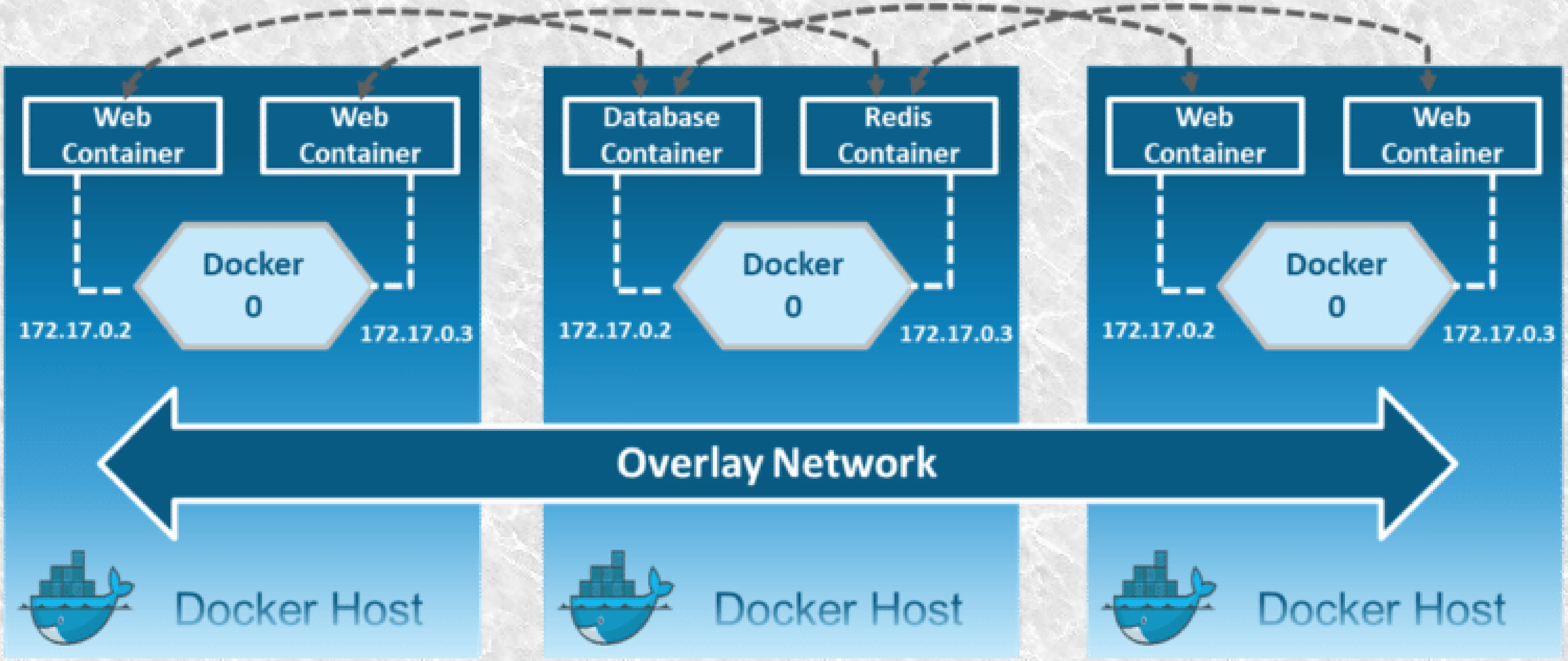
Connect container to Docker networks

Host network



Connect container to Docker networks

Overlay network



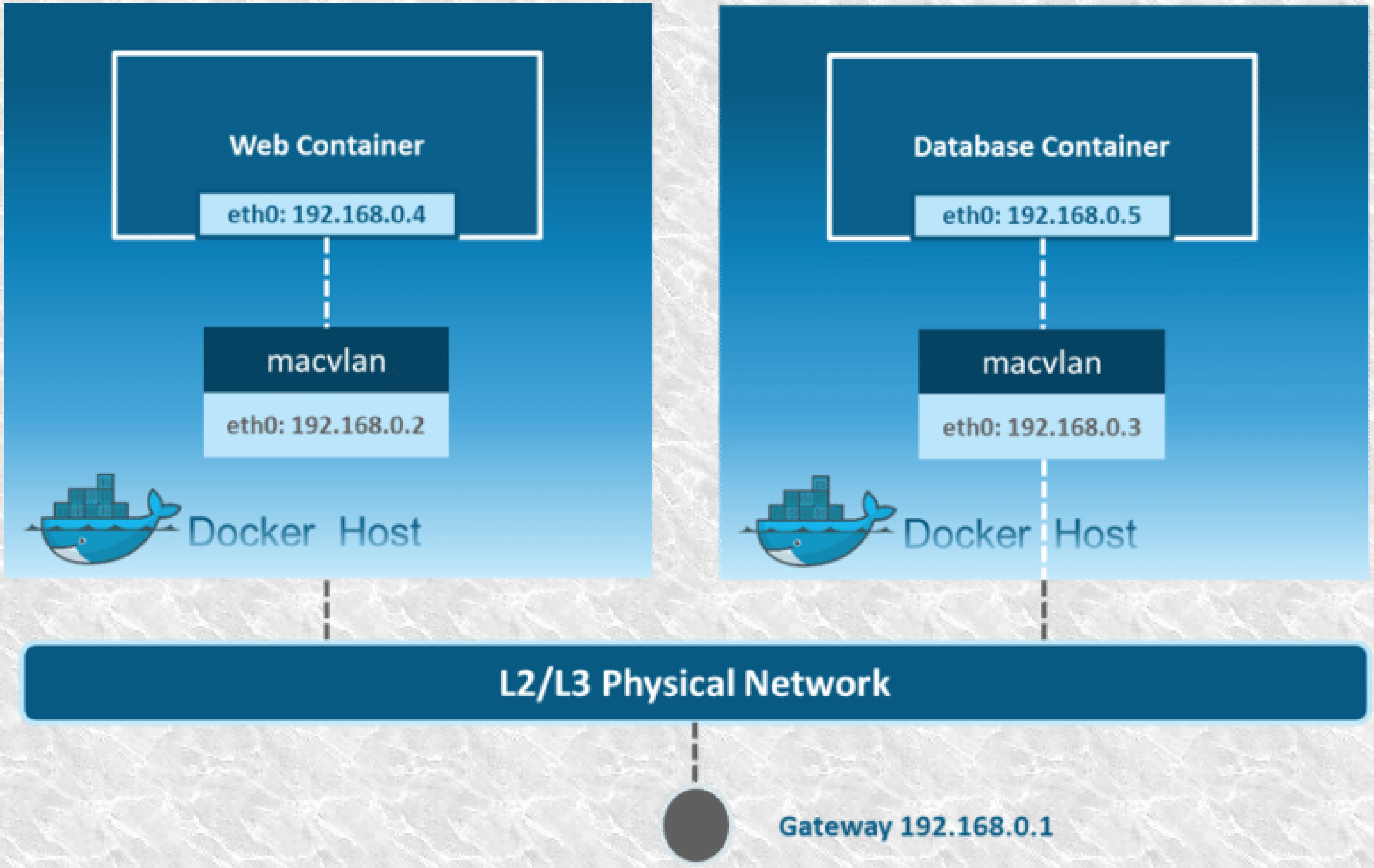
Connect container to Docker networks
overlay networks

- 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*

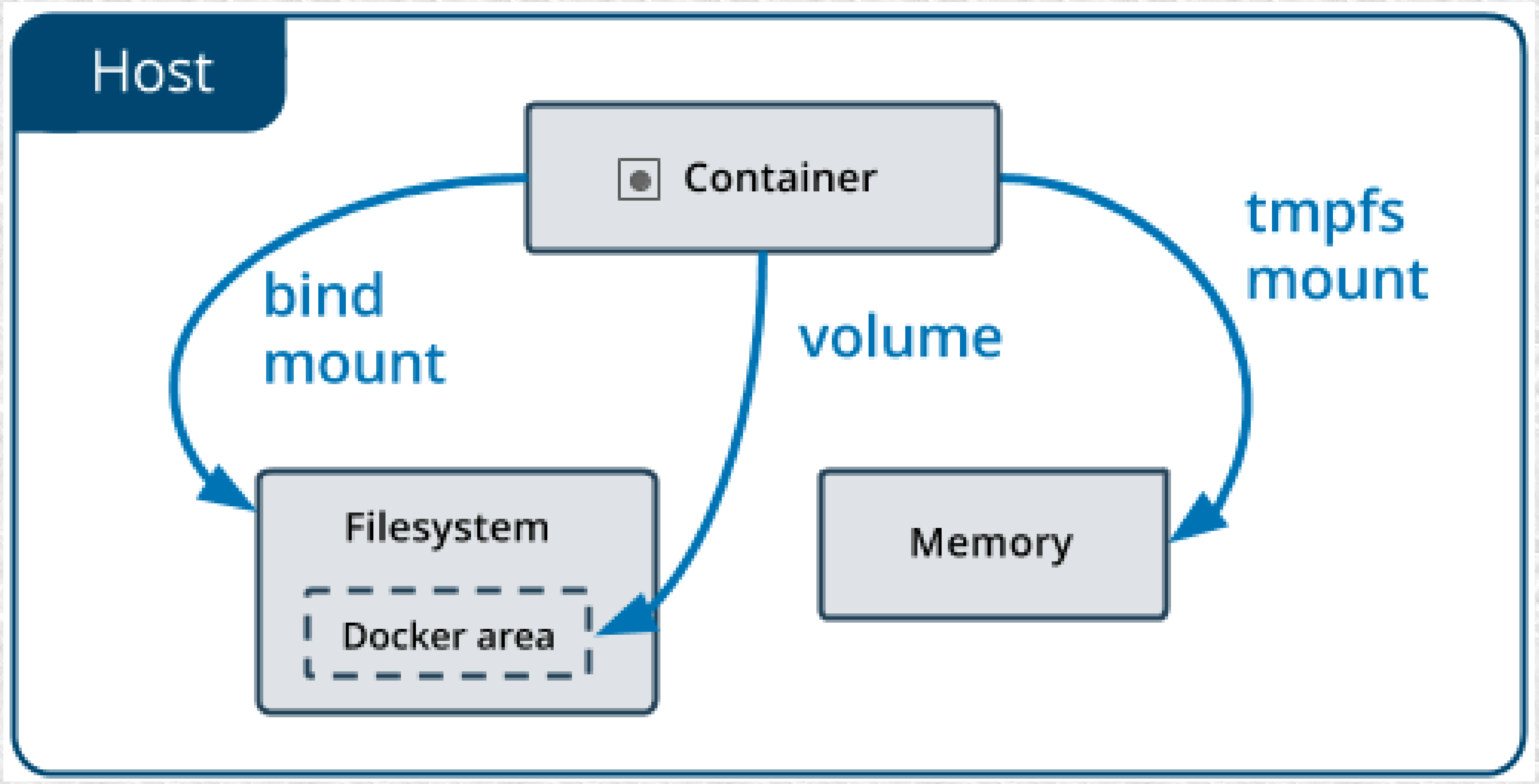
Connect container to Docker networks

Macvlan network



Manage container storage with volumes

Docker storage mecanisms



Manage container storage with volumes

Manage volumes

- Volumes created and managed by Docker.
- Some use cases for volumes include:
  - 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.
- Commands :
  - docker volume create my-vol
  - docker volume ls
  - docker volume inspect my-vol
  - docker volume rm my-vol
  - docker run -v /dbdata --name dbstore2 ubuntu /bin/bash
  - docker run -d --name devtest --mount source=myvol2,target=/app \
    nginx:latest

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)

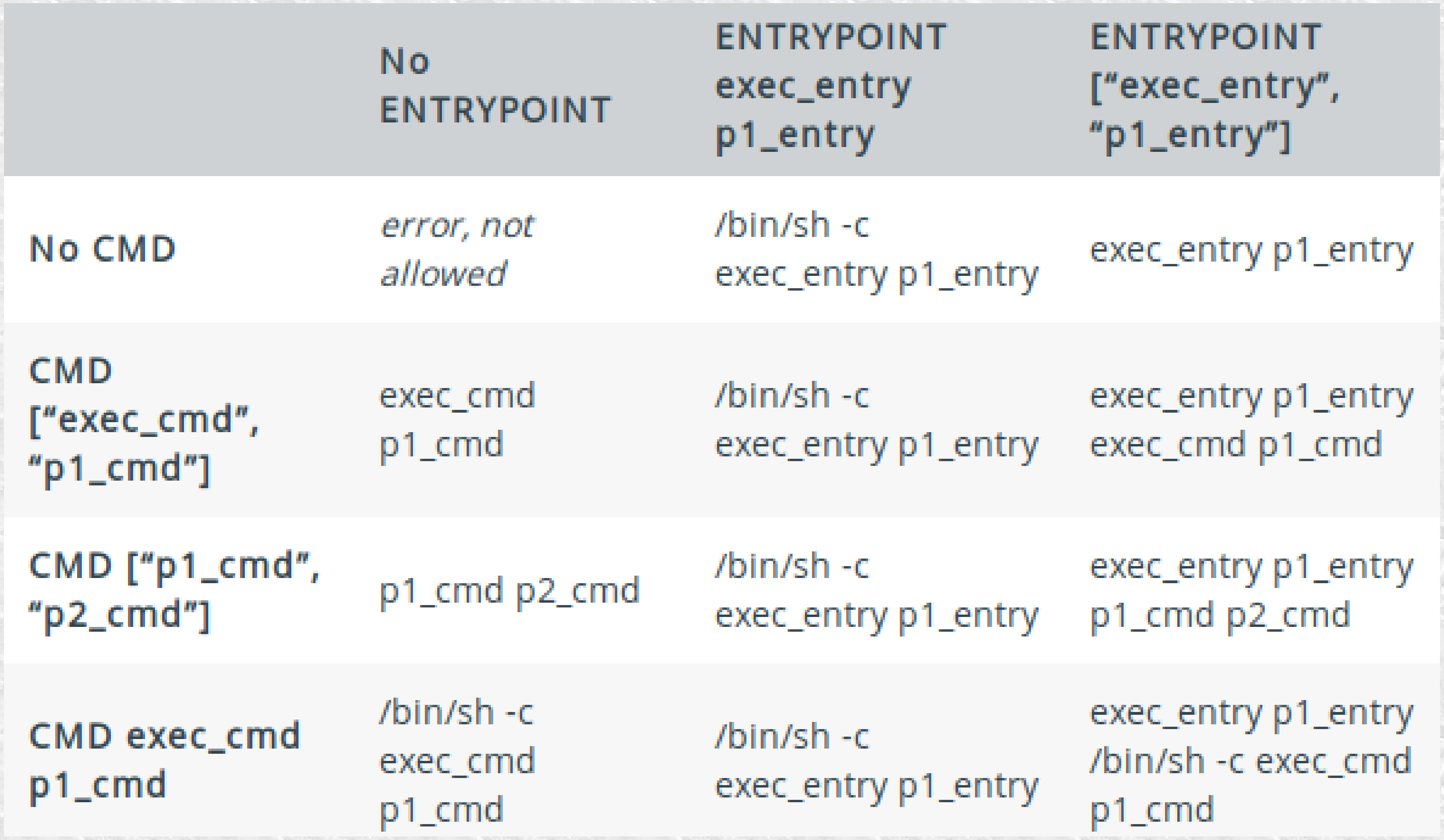
Create Dockerfiles and build images

CMD & ENTRYPOINT instructions

- Provide defaults for an executing container.
- There can only be one CMD/ENTRYPOINT instruction in a Dockerfile.
- If you list more than one CMD/ENTRYPOINT then only the last will take effect.
- 2 forms:
  - Shell form
  - Exec form
- In the shell or exec formats, the instruction sets the command to be executed when running the image.
- When running image CMD intruction can be overrided, 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

How CMD and ENTRYPOINT interact



	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT [“exec_entry”, “p1_entry”]
No CMD	*error, not allowed*	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD [“exec_cmd”, “p1_cmd”]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD [“p1_cmd”, “p2_cmd”]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Create Dockerfiles and build images

COPY, ADD, WORKDIR structions

- 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 volumes from native host or other containers.
- 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 6
Container Infrastructure

Plan

- Docker machine
- Container infrastructure tools
- Service discovery

What is Docker Machine?

- Docker Machine create hosts with Docker Engine installed on them.
- Machine can create Docker hosts on
  - local Mac
  - Windows box
  - company network
  - data center
  - cloud providers like Azure, AWS, or Digital Ocean.
- docker-machine commands can
  - start, inspect, stop, and restart a managed host,
  - upgrade the Docker client and daemon,
  - configure a Docker client to talk to host.

Docker machine

docker-machine create

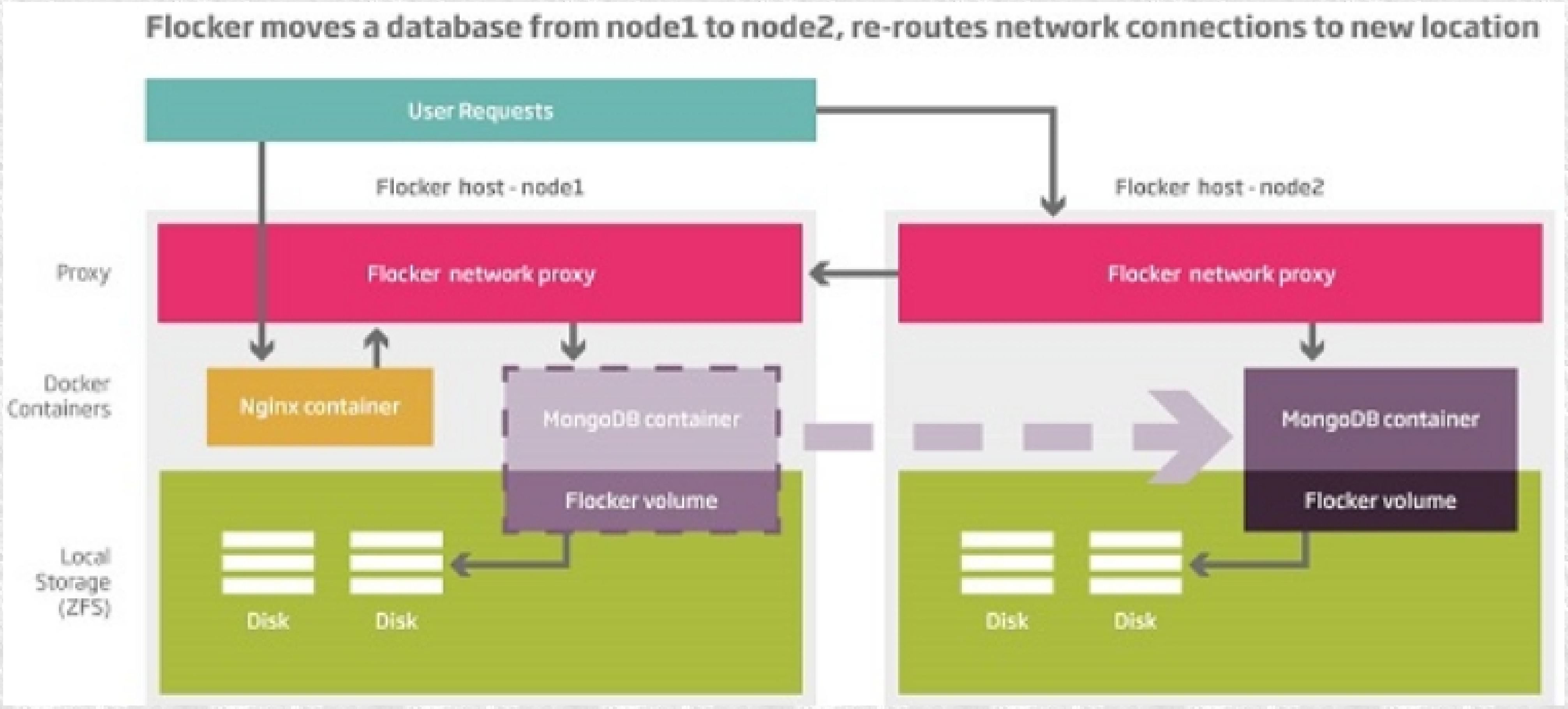
- Create a machine. Requires the --driver flag to indicate which provider (VirtualBox, DigitalOcean, AWS, etc.)
- Examples :
  - docker-machine create --driver virtualbox dev
  - docker-machine create --driver digitalocean --\ digitalocean-access-token xxxxx docker-sandbox
  - docker-machine create --driver amazonec2 --\ amazonec2-access-key AKI******* --amazonec2-\ secret-key 8T93C******* aws-sandbox

Container infrastructure tools
Flocker

- Flocker is an open-source container data volume manager for your Dockerized applications.
- By providing tools for data migrations, Flocker gives ops teams the tools they need to run containerized stateful services like databases in production.
- Unlike a Docker data volume which is tied to a single server, a Flocker data volume can be used with any container in the cluster.
- Flocker manages Docker containers and data volumes together.

Container infrastructure tools

Flocker



Container infrastructure tools

Flannel

- Flannel is a networking technology used to connect Linux Containers.
- It is distributed and maintained by CoreOS
- Flannel is a virtual network that gives a subnet to each host for use with container runtimes.
- Used by docker container orchestration tools (docker swarm, kubernetes, ...) to ensure that all containers on different hosts have different IP addresses.

Container infrastructure tools

etcd

- etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines.
- etcd gracefully handles leader elections during network partitions and will tolerate machine failure, including the leader.
- applications can read and write data into etcd.
- Use-case : to store database connection details or feature flags in etcd as key value pairs.

Container infrastructure tools

rkt

- Rkt (Rocket) : Open source container runtime, developed by CoreOS.
- An alternative to Docker daemon.
  - Docker evolve into a complex platform that serves a variety of needs and functions
  - Rocket is designed to serve as a simple but secure re-usable component for deploying applications.
- Registration isn’t necessary to distribute the image.
- It is possible to access an ACI (App Container Image) hosted on any server by direct URL.

Service discovery
The challenge

- The idea behind zeroconf is to
  - automatically create and manage a computer network by automatically assigning network addresses,
  - automatically distributing and resolving hostnames,
  - automatically managing network services.
- Maintaining a mapping between a running container and its location (IP address, …)
- This mapping has to be done in a timely manner and accurately across relaunches of the container throughout the cluster.
- Docker and kubernetes mainly use DNS

Module 7

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

docker-compose use cases

- Compose can be used in many different ways
- Development environments :
  - create and start one or more containers for each dependency (databases, queues, caches, web service APIs, etc) with a single command.
- Automated testing environments :
  - create and destroy isolated testing environments in just a few commands.
- Cluster deployments :
  - Compose can deploy to a remote single docker Engine.
  - The Docker Engine may be a single instance provisioned with Docker Machine or an entire Docker Swarm cluster.

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.

Docker-compose
Service configuration reference

- The Compose file is a YAML file defining services, networks and volumes (no containers, no nodes).
- A service definition contains configuration that is applied to each container started for that service, much like passing command-line :

  docker container create

- Likewise, network and volume definitions are analogous to :

  docker network create

  And

  docker volume create

- Options specified in the Dockerfile, such as CMD, EXPOSE, VOLUME, ENV, are respected by default (don't need to specify them again in docker-compose.yml)

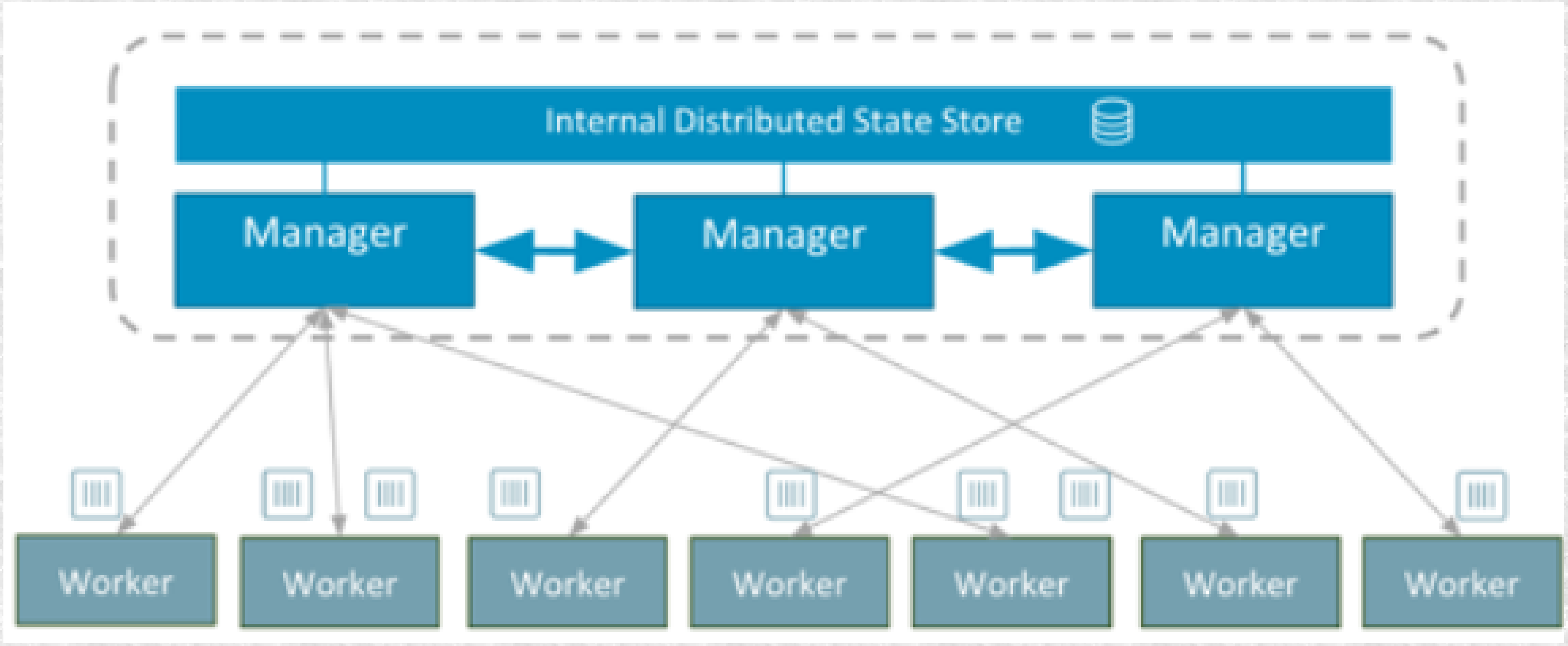
What’s docker swarm mode

- 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

Manager nodes

- Manager nodes handle cluster management tasks:
  - maintaining cluster state
  - scheduling services
  - serving swarm mode HTTP API endpoints
- managers maintain a consistent internal state of the entire swarm and all the services running on it.
- If the manager in a single-manager swarm fails, your services continue to run, but you need to create a new cluster to recover.
- It’s recomended to implement more than one manager for high-availability requirements.

Docker swarm

Worker nodes

- Worker nodes are also instances of Docker Engine whose execute containers.
- You can create a swarm of one manager node, but you cannot have a worker node without at least one manager node.
- By default, all managers are also workers ⇒ In a single manager node cluster scheduler places all tasks on the local Engine.
- To prevent the scheduler from placing tasks on a manager node, set the availability for the manager node to *Drain*.
- It’s possible to promote a worker node to be a manager by running : *docker node promote*

Docker swarm

Swarm Services networks

- The following three network concepts are important to swarm services:
  - Overlay networks : 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 the host machines.

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:

```
docker node ls
```

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

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 in a consistent and maintainable way.
- 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.

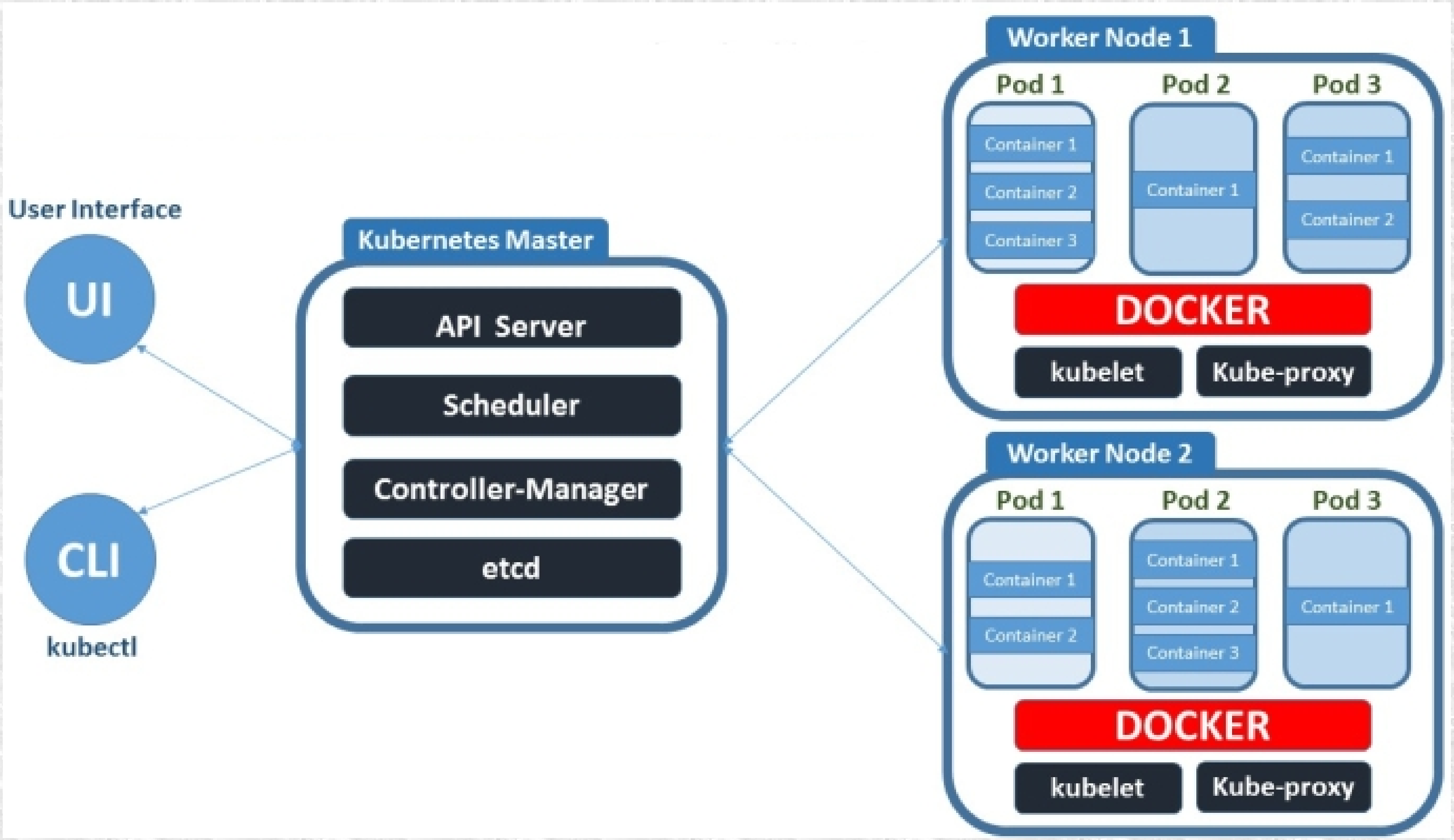
Kubernetes vs docker swarm : Terminology

	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 vs docker swarm : Features

Features	**Docker Swarm**	**Kubernetes**
Installation & Cluster configuration	Installation very simple, but cluster not very strong	Insttallation 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 trafic between containers in the cluster	Manual intervention needed for load balancing trafic 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
  - Kube-scheduler
  - Kube-controller-manager
  - etcd

- Might contain:
  - Kube-proxy
  - a network management utility

Kubernetes
Node

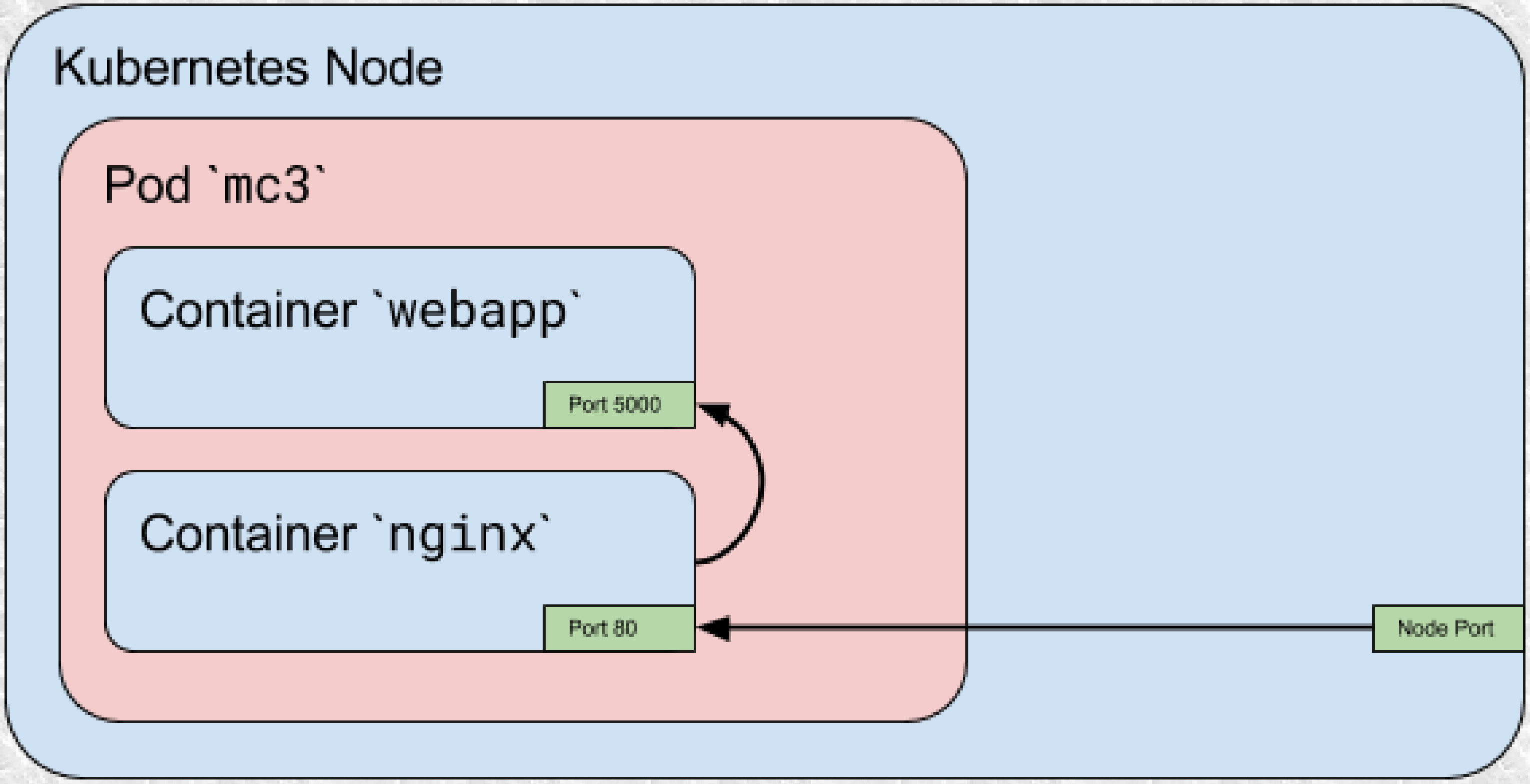
- Typically consists of:
  - Kubelet
  - Kube-proxy
  - cAdvisor

- Might contain:
  - a network management utility

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
  - Rollback to older Deployment versions.
  - Scale Deployment up or down.
  - Pause and resume the Deployment.
  - Canary Deployment.
  - ...

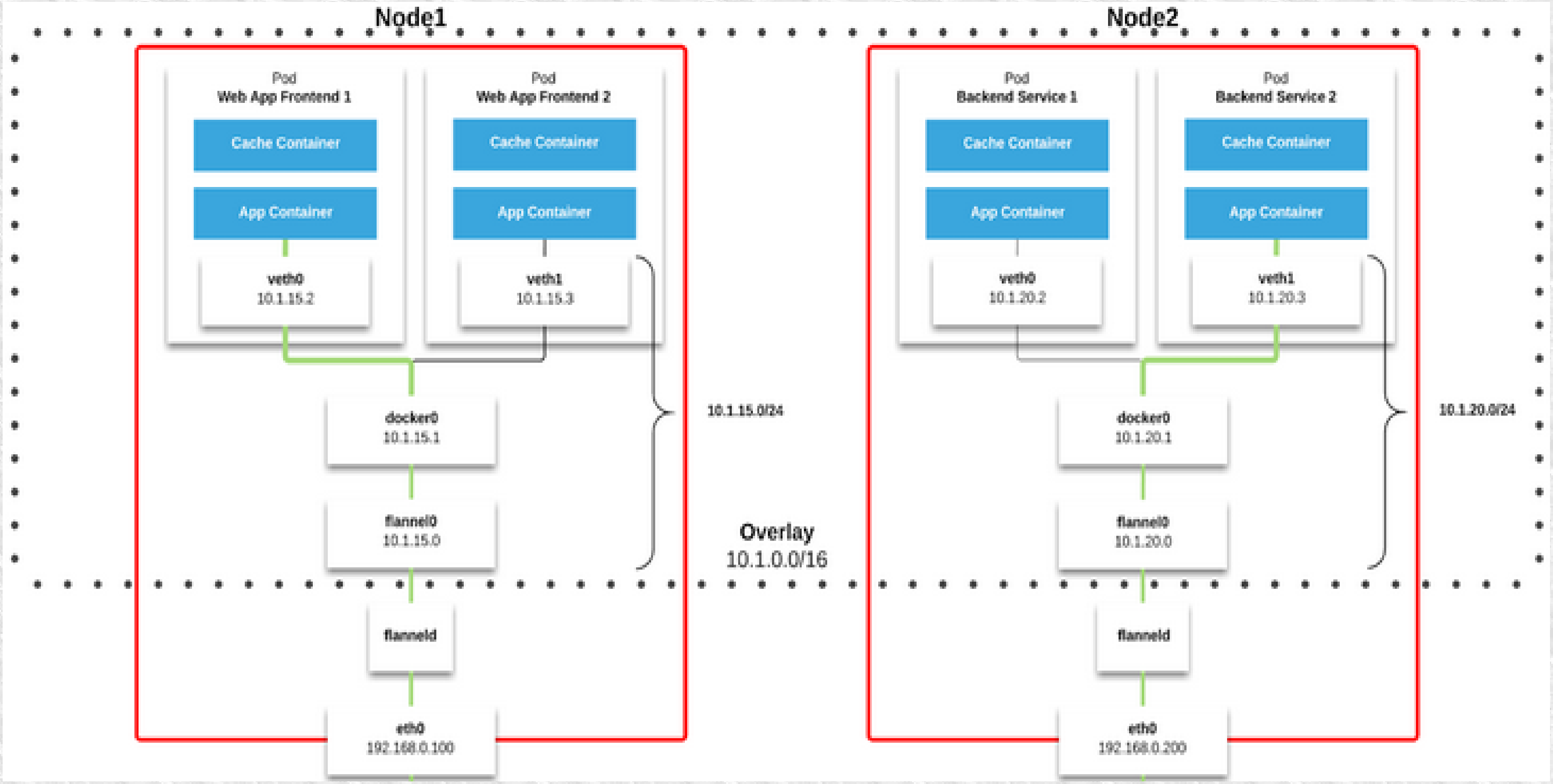
Kubernetes
Services

- 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
Noetwork Model

- Every Pod get its own IP address
  - Not need to explicitly create links between Pods
  - Almost, never need to deel 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



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

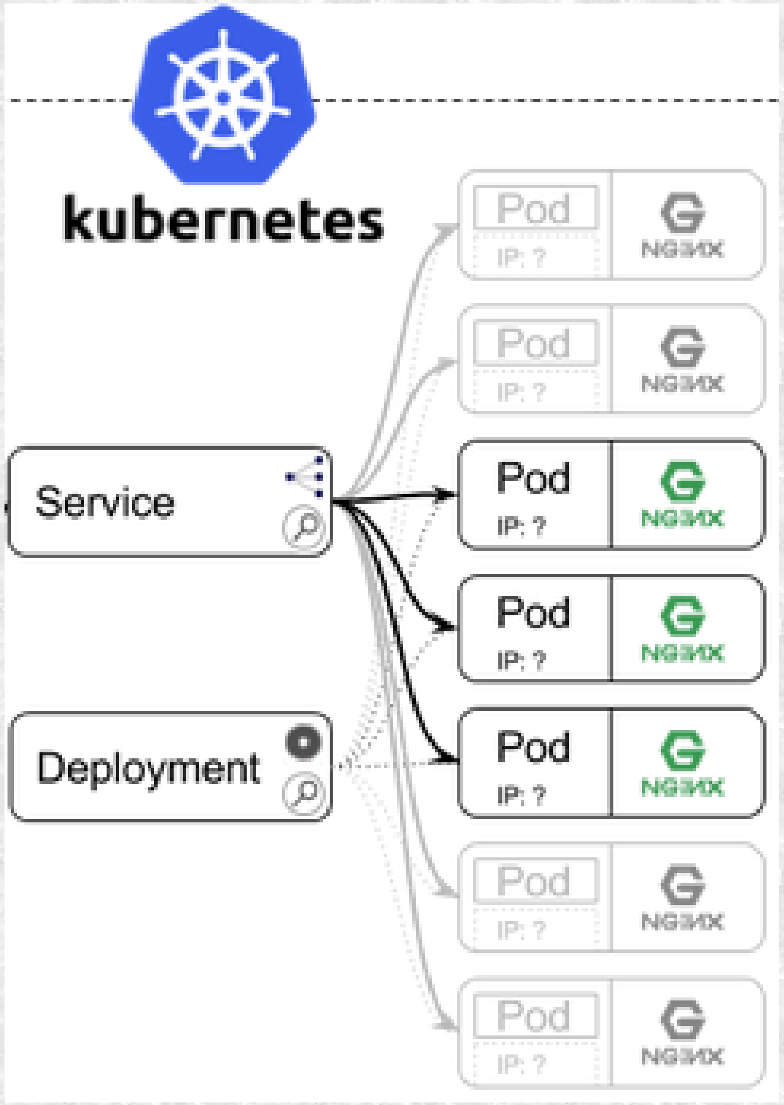
- Running commands against Kubernetes clusters.
- Syntax :

  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.

Kubernetes

Service-deployment-pod relationship



Brahim HAMDI

LPI DevOps Tools Engineers

Module 8

Ansible and configuration management tools

Plan

- Configuration management tools
- Ansible
- Inventory
- Playbook
- Variables
- Template module (Jinja2)
- Roles
- ansible-vault
- Puppet
- Chef

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

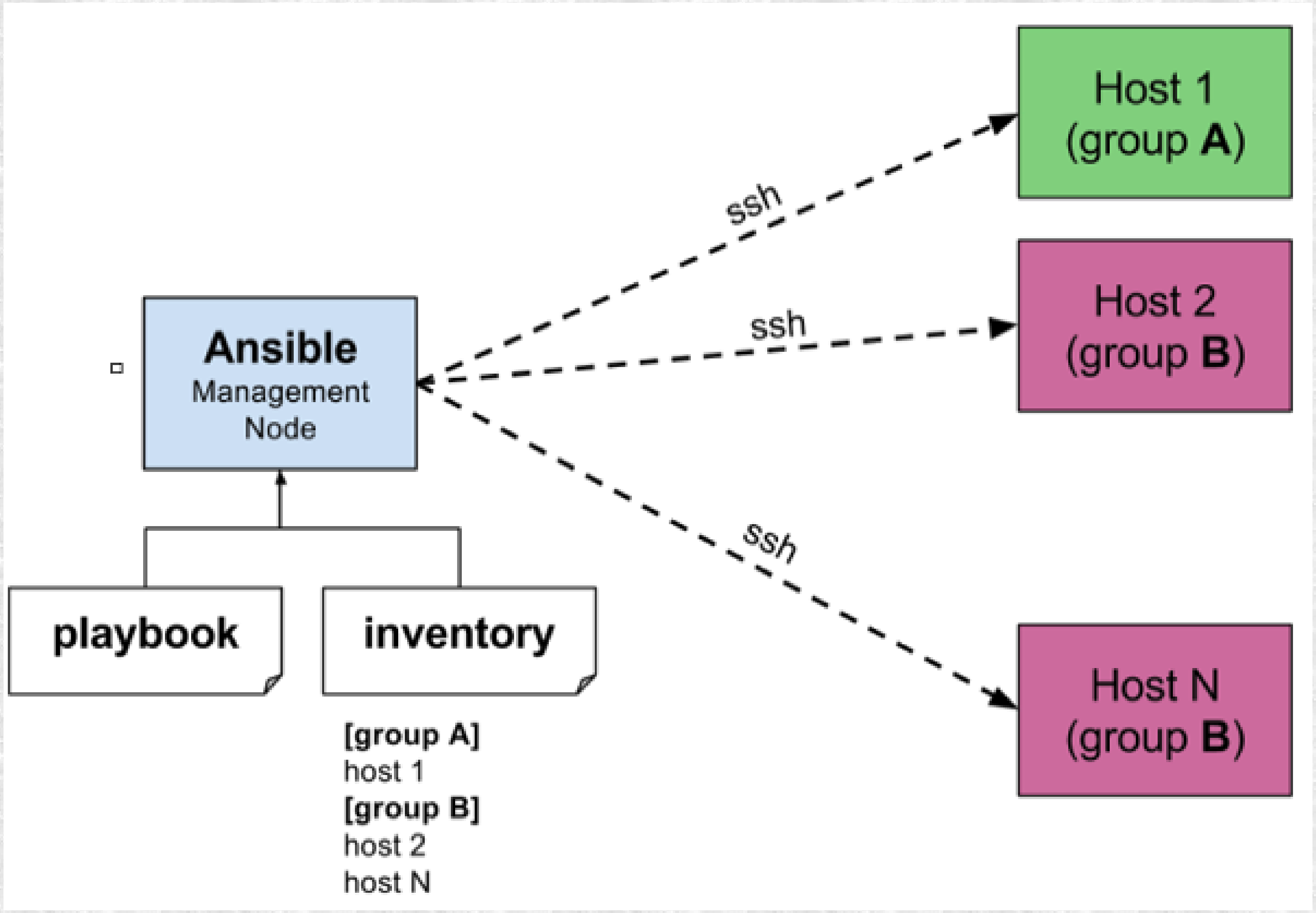
Configuration management tools

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, groups and aspects of hosts 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 incuding or excluding groups and single hosts
- Selection can be done by passing :
  - all / *
  - Groups names
  - Exclusion (all:!CentOS)
  - Intersection (webservers:&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

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
  - hort
  - inventory_file
  - playbook_dir
  - play_hosts
  - hostvars
  - groups
  - group_names
  - ansible_ssh_user

Template module
Jinja2

- Templates allow to create dynamic configuration files using variables.
- All templating happens on the Ansible controller before the task is sent and executed on the target machine.
- Ansible uses Jinja2 templating to enable dynamic expressions and access to variables.
- Example of using Jinja2 template :

```
- template:
      src=/mytemplates/foo.j2
      dest=/etc/file.conf
      owner=bin
      group=wheel
      mode=0644
```

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

Puppet

What is puppet ?

- Puppet is designed to manage the configuration of Unix-like and Microsoft Windows systems declaratively
- Describes system resources and their state using the Puppet DSL
- The Puppet DSL is based on Ruby
- Resource types are used to manage system resources
- Resource types are declared in manifests files

Puppet

Puppet ressources

Resource Type Format:

```
<TYPE> { '<TITLE>':
<ATTRIBUTE> => <VALUE>,
}
```

Example:

```
user { ‘username’:
 ensure => present,
 uid => ‘102’,
 gid => ‘wheel’,
 shell => ‘/bin/bash’,
 home => ‘/home/username’,
 managehome => true,
}
```

Puppet commands

- puppet apply: manages systems without needing to contact a Puppet master server
- puppet agent: manages systems, with the help of a Puppet master
- puppet cert: helps manage Puppet’s built-in certificate authority (CA)
- puppet module: is a multi-purpose tool for working with Puppet modules
- puppet resource: lets you interactively inspect and manipulate resources on a system
- puppet parser: lets you validate Puppet code to make sure it contains no syntax errors

What is Chef ?

- Chef is both the name of a company, and the name of a configuration management tool written in Ruby.
- It uses a pure Ruby DSL.
- Use Chef Development Kit (Chef DK) to get the tools to test your code.
- Chef uses a client-server model.
- It utilizes a declarative approach to configuration management.
- Resources are idempotent.

What is Chef ?

- Chef testing tools:
  - Cookstyle : code linting – automatically correct style, syntax and logic mistakes.
  - Foodcritic : deprecated (use cookstyle instead)
  - ChefSpec : test resources and recipes
  - InSpec : test and audit infrastructures by comparing the actual and desired state
  - Test Kitchen :

What is Chef ?

- Use resources to describe your infrastructure.
- A Chef recipe is a file that groups related resources.
- Chef cookbook provides structure to your recipes.
- Use the knife command for interacting with the Chef server.

Chef
Chef-client

- A chef-client is an agent that runs nodes managed by Chef.
- The agent will bring the node into the expected state:
  - Registering and authenticating the node with the Chef server
  - Building the node object
  - Synchronizing cookbooks
  - Taking the appropriate and required actions to configure the node
  - Looking for exceptions and notifications

Chef
chef-server-ctl

- This is used to:
  - Start and stop individual services
  - Reconfigure the Chef server
  - Gather Chef server log files
  - Backup and restore Chef server data

Chef
chef-solo

- chef-solo:
  - A command that executes chef-client to converge cookbooks in a way that does not require the Chef server
  - Uses chef-client’s Chef local mode
- Does not support:
  - Centralized distribution of cookbooks
  - A centralized API that interacts with and integrates infrastructure components
  - Authentication or authorization

Chef
Cookbooks

- A cookbook is the fundamental unit of configuration and policy distribution. A cookbook defines a scenario and contains everything that is required to support that scenario:
  - Recipes that specify the resources to use and the order in which they are to be applied
  - Attribute values
  - File distributions
  - Templates
  - Extensions to Chef, such as custom resources and libraries

Chef

Cookbooks commands

- knife cookbook
- knife cookbook generate COOKBOOK_NAME (options)
- knife cookbook delete COOKBOOK_NAME [COOKBOOK_VERSION] (options)
- knife cookbook download COOKBOOK_NAME [COOKBOOK_VERSION] (options)
- knife cookbook list (options)
- knife cookbook metadata (options)
- knife cookbook show COOKBOOK_NAME
- knife cookbook upload [COOKBOOK_NAME...] (options)

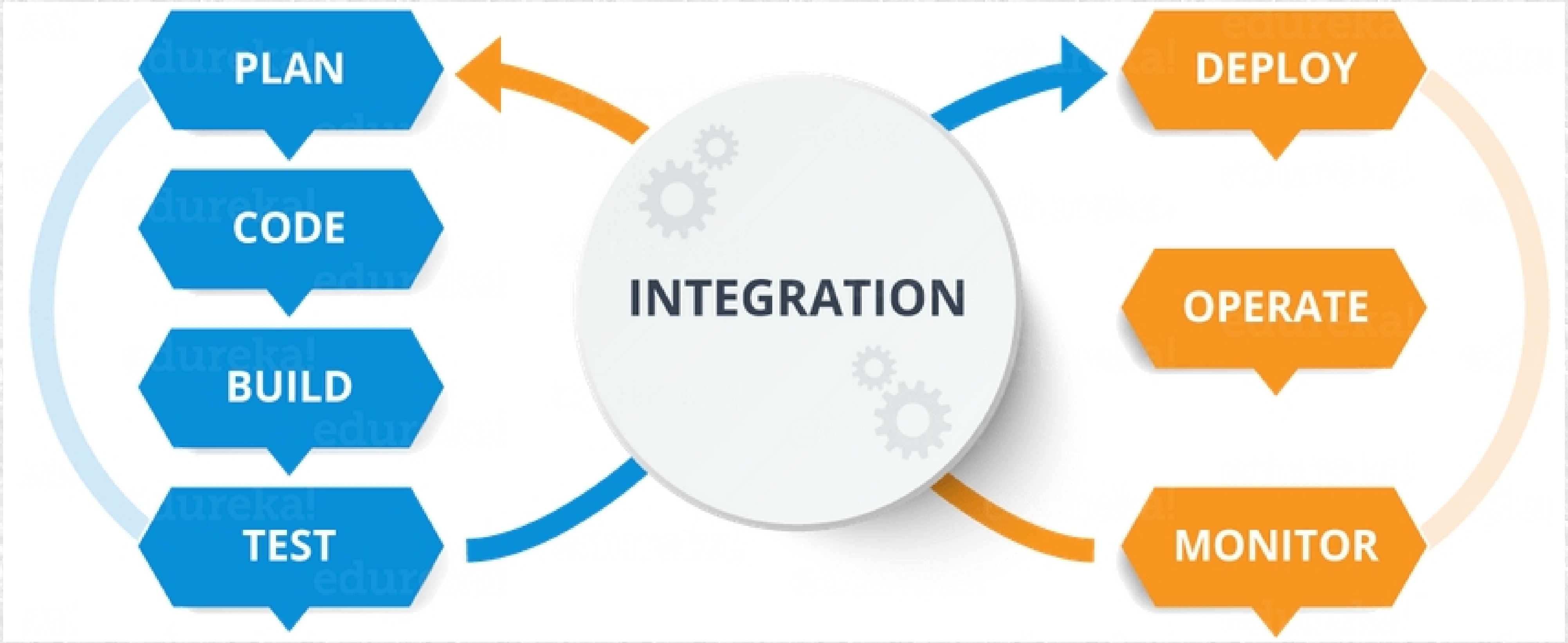
LPI DevOps Tools Engineers

Module 9
CI/CD with Jenkins

Plan

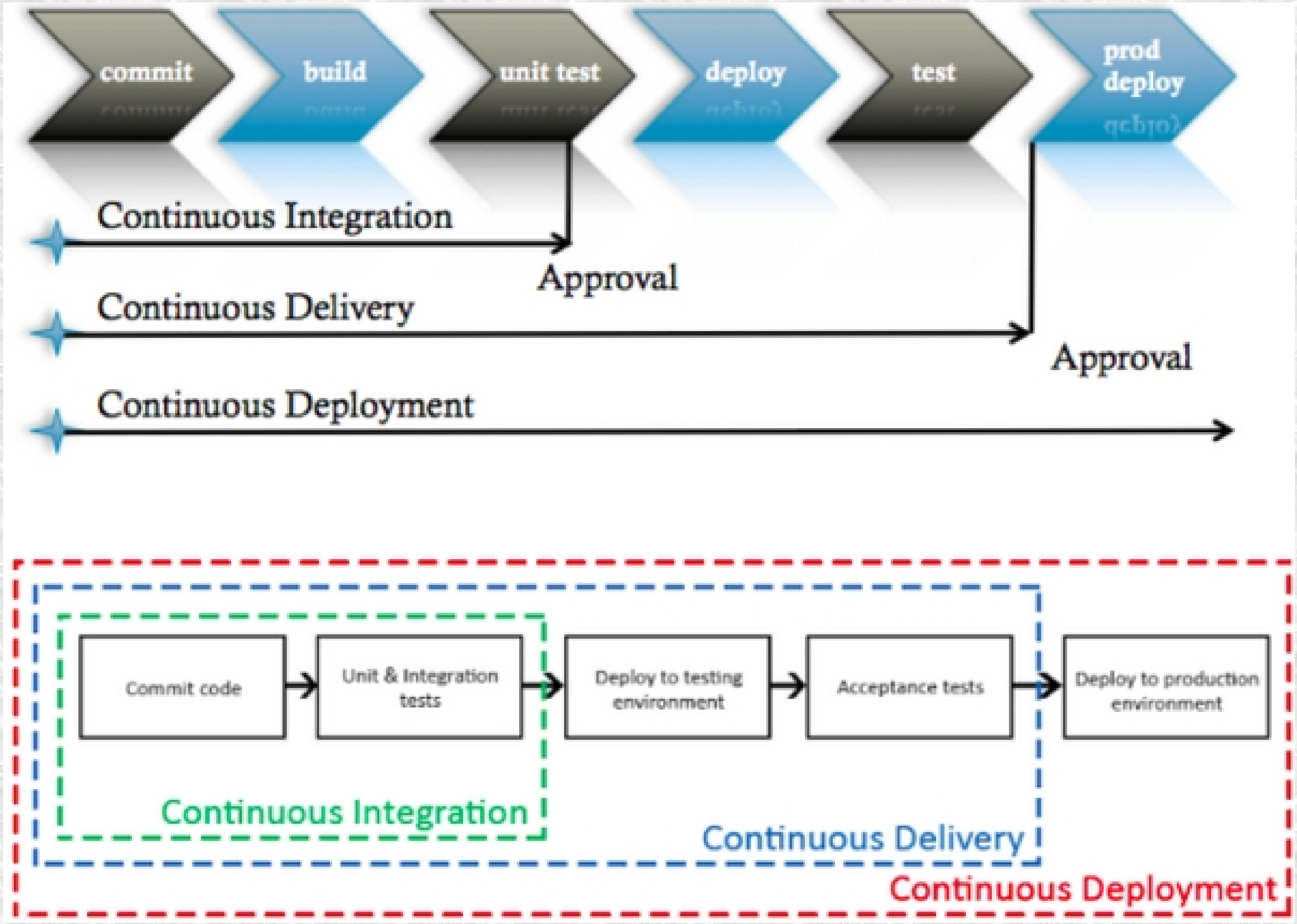
- CI/CD
- Jenkins
- Building a CI/CD Pipeline Using Jenkins

CI/CD
DevOps lifecycle



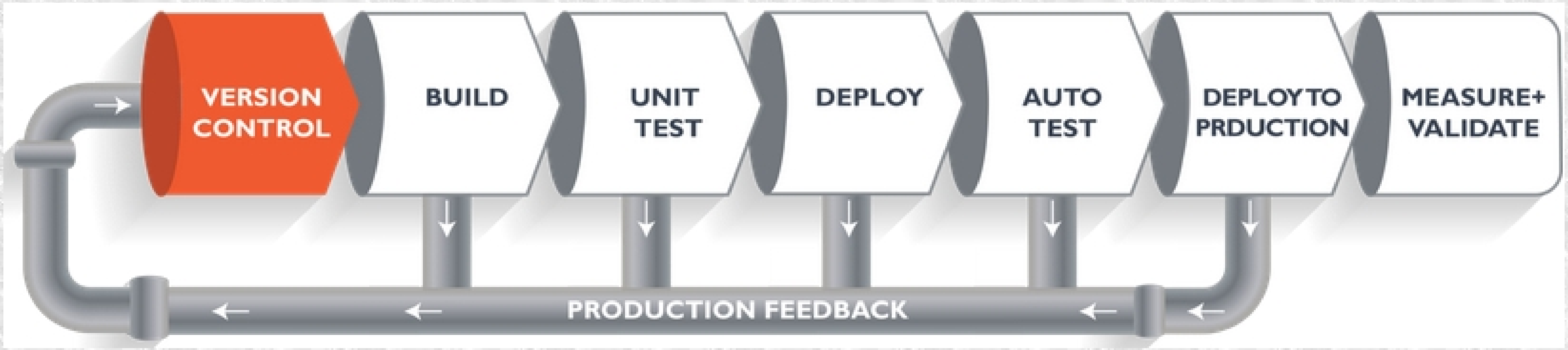
CI/CD

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
- Jenkins can automate the entire process.
  - with various interfaces and tools (Git, docker, etc ...)
- Git used by development team to commit the code.
  - From Git, Jenkins pulls the code and then Jenkins moves it into the commit phase
- Tools like maven 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.

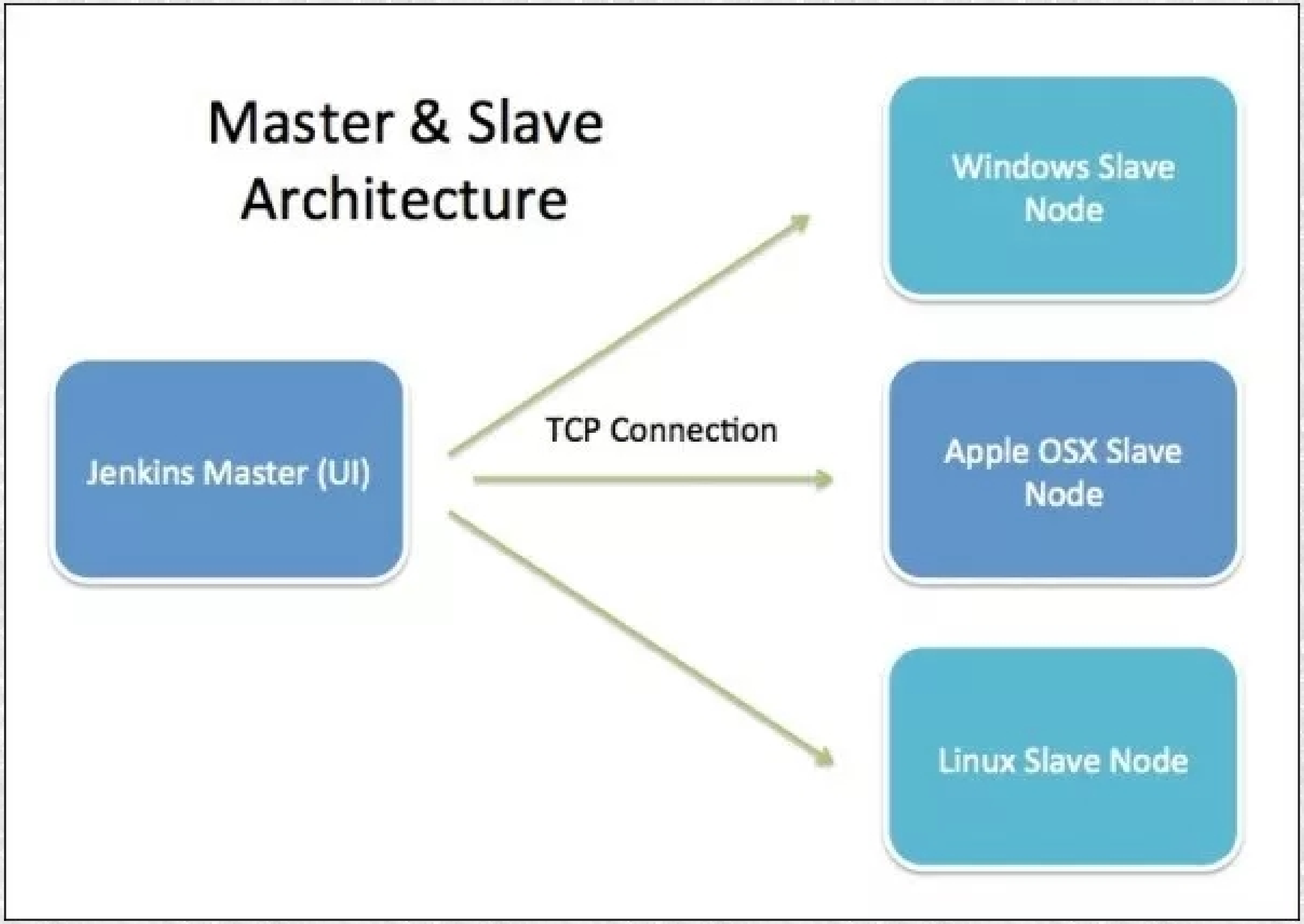
Jenkins
Plugins and mailer

- 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



Jenkins SDL
What is it ?

- SDL – Specific Domain Language
- SDL allows to write code in order to create jobs.
- Job SDL scripts are written in Groovy
  - a dynamic language built on top of Java.
- The next example shows creating of four jobs:
  - PROJ-unit-tests
  - PROJ-sonar
  - PROJ-integration-tests
  - PROJ-release

Jenkins SDL Examples

```
def gitUrl = 'git://github.com/jenkinsci/job-dsl-plugin.git'
job('PROJ-unit-tests') {
    scm {
        git(gitUrl)
    }
    triggers {
        scm('*/15 * * * *')
    }
    steps {
        maven('-e clean test')
    }
}

job('PROJ-sonar') {
    scm {
        git(gitUrl)
    }
    triggers {
        cron('15 13 * * *')
    }
    steps {
        maven('sonar:sonar')
    }
}
```

```
job('PROJ-integration-tests') {
    scm {
        git(gitUrl)
    }
    triggers {
        cron('15 1,13 * * *')
    }
    steps {
        maven('-e clean integration-test')
    }
}

job('PROJ-release') {
    scm {
        git(gitUrl)
    }
    // no trigger
    authorization {
        // limit builds to just Jack and Jill
        permission('hudson.model.Item.Build', 'jill')
        permission('hudson.model.Item.Build', 'jack')
    }
    steps {
        maven('-B release:prepare release:perform')
        shell('cleanup.sh')
    }
}
```

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 10
IT monitoring

Plan

- Monitoring
- Prometheus
- Prometheus’ configuration file
- Exposing metrics for Prometheus
- Prometheus alertmanager
- Dashboarding whith grafana

Monitoring

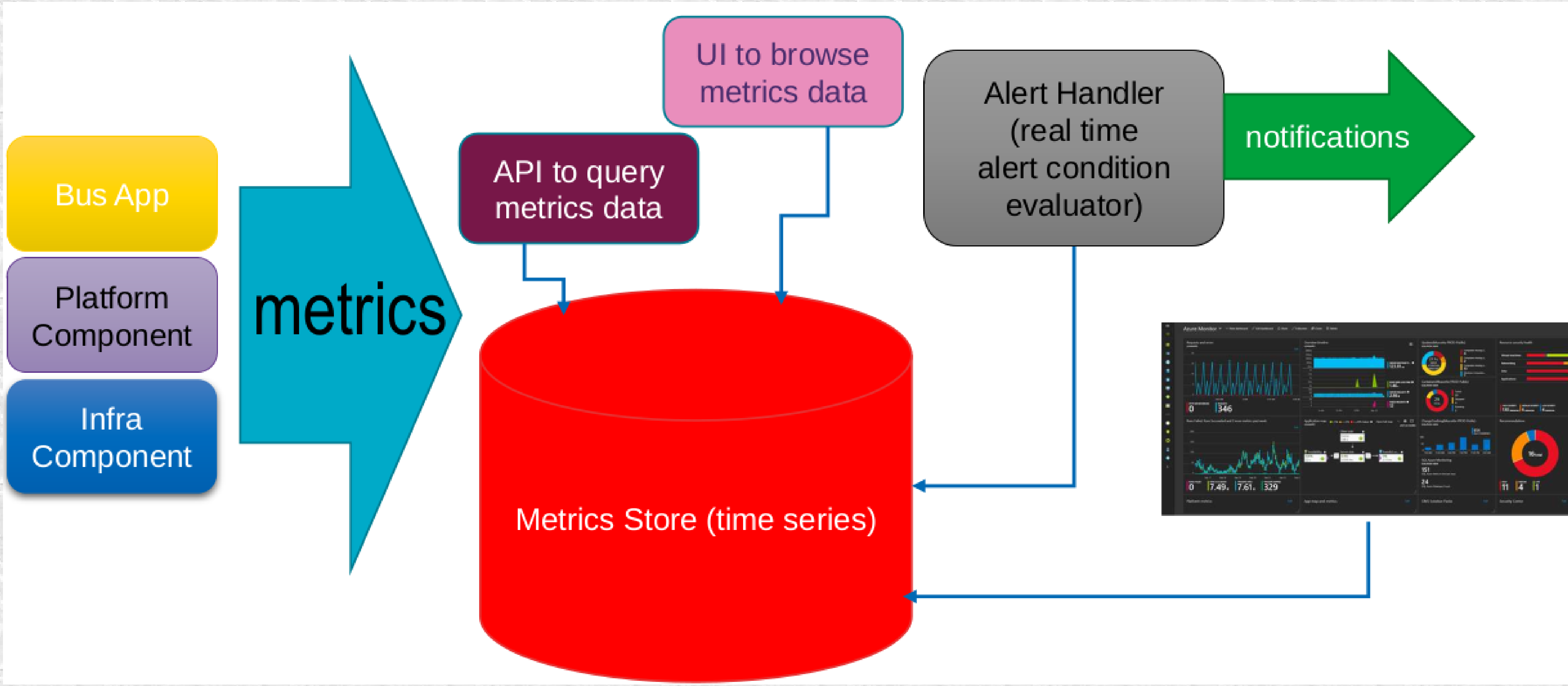
Why monitor ?

- 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.

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

Monitoring
Process


Bus App
Platform Component
Infra Component
metrics
API to query metrics data
UI to browse metrics data
Alert Handler (real time alert condition evaluator)
notifications
Metrics Store (time series)

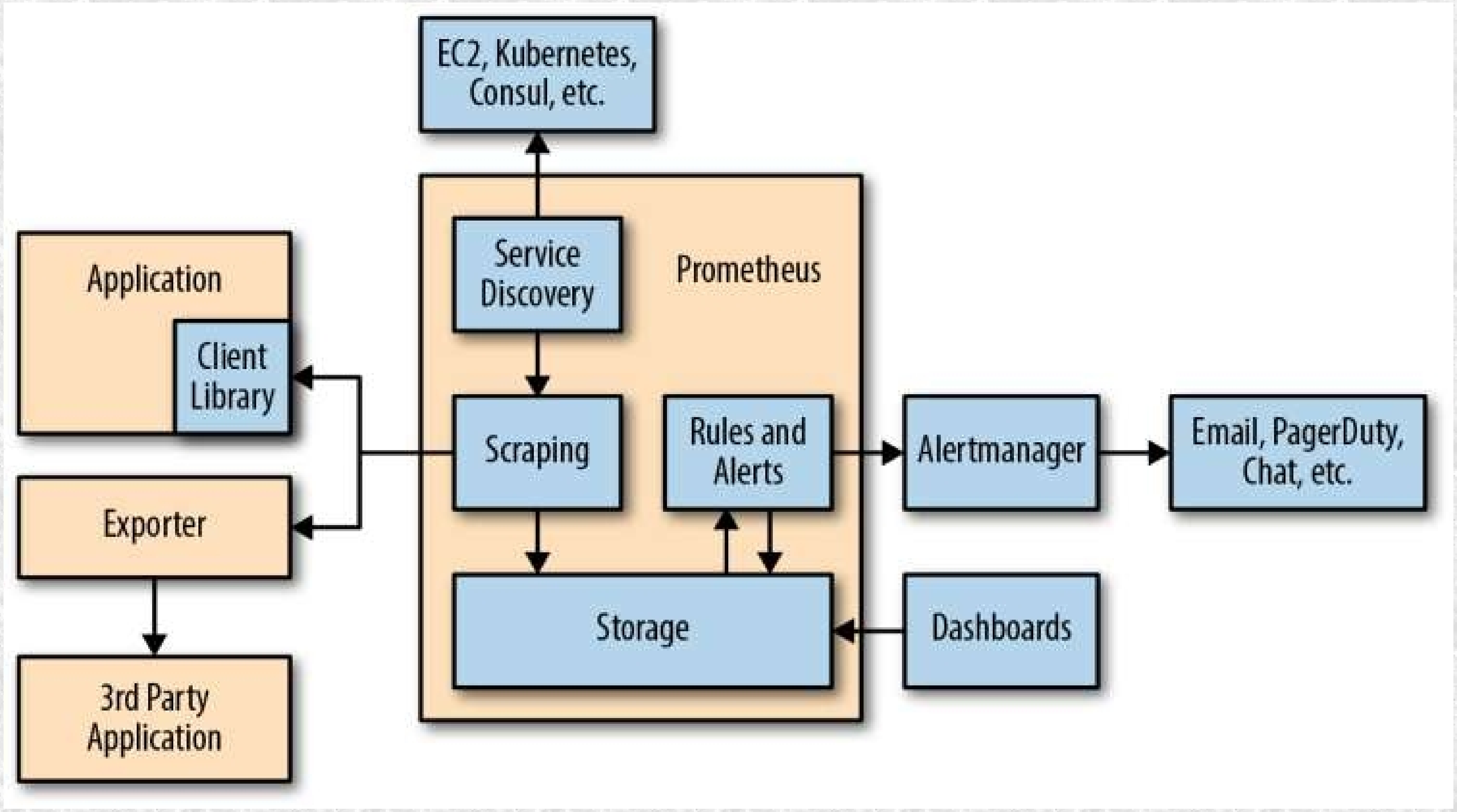
Prometheus

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’ 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.
    - Rules are used to create new time series and for the generation of alerts.
- rule_files : information of the location of any rules we want the Prometheus server to load.
- scape_configs : which resources Prometheus monitors.

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

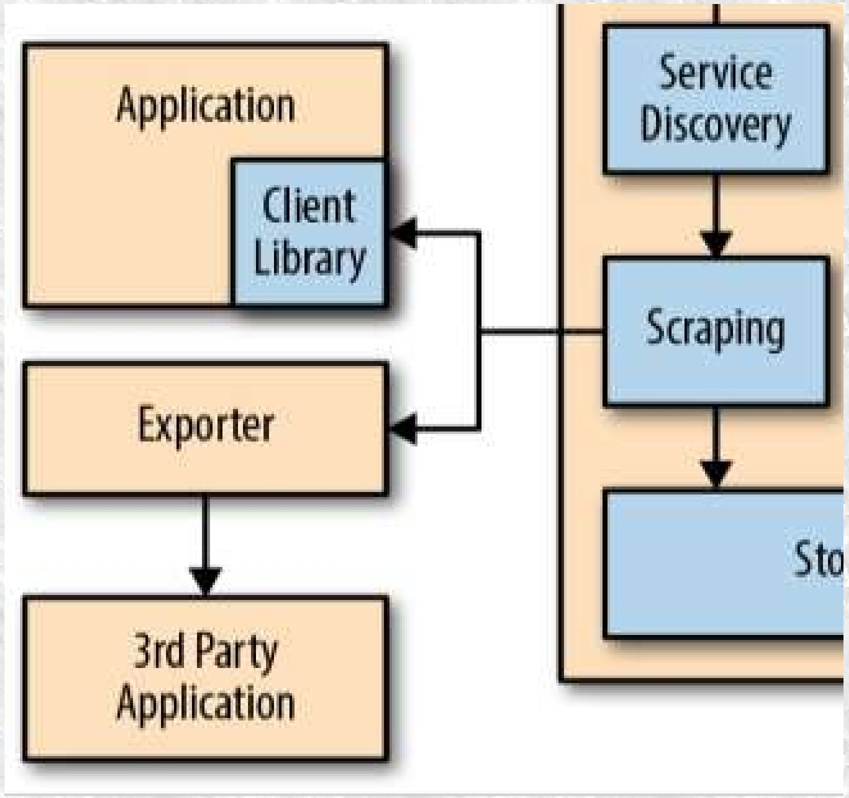
Scrape Metrics for Prometheus

- 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

Client Libraries for Exposing Metrics

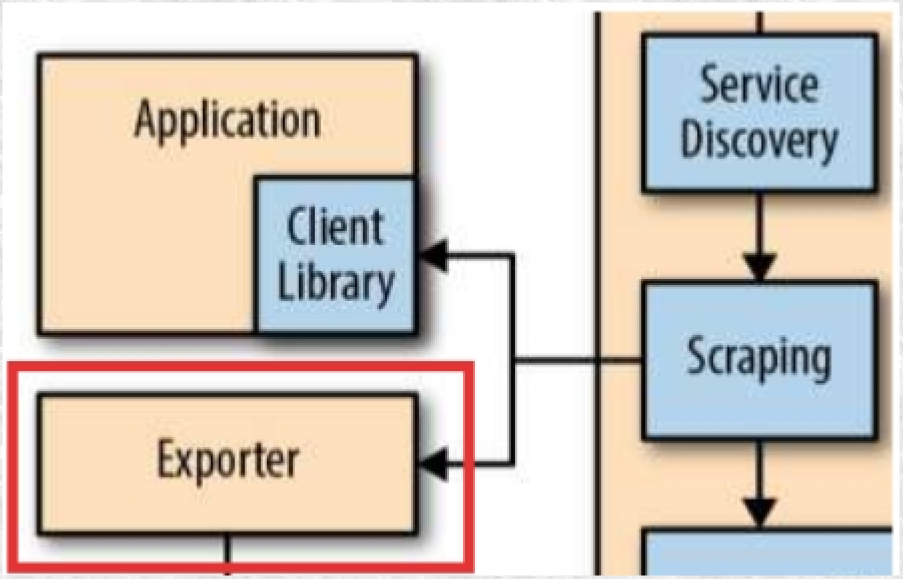
- Go
- Java
- Python
- Ruby
- Bash
- C++
- Node.js
- PHP
- .Net / C#
- etc …



Exposing metrics for Prometheus

Prometheus Exporters

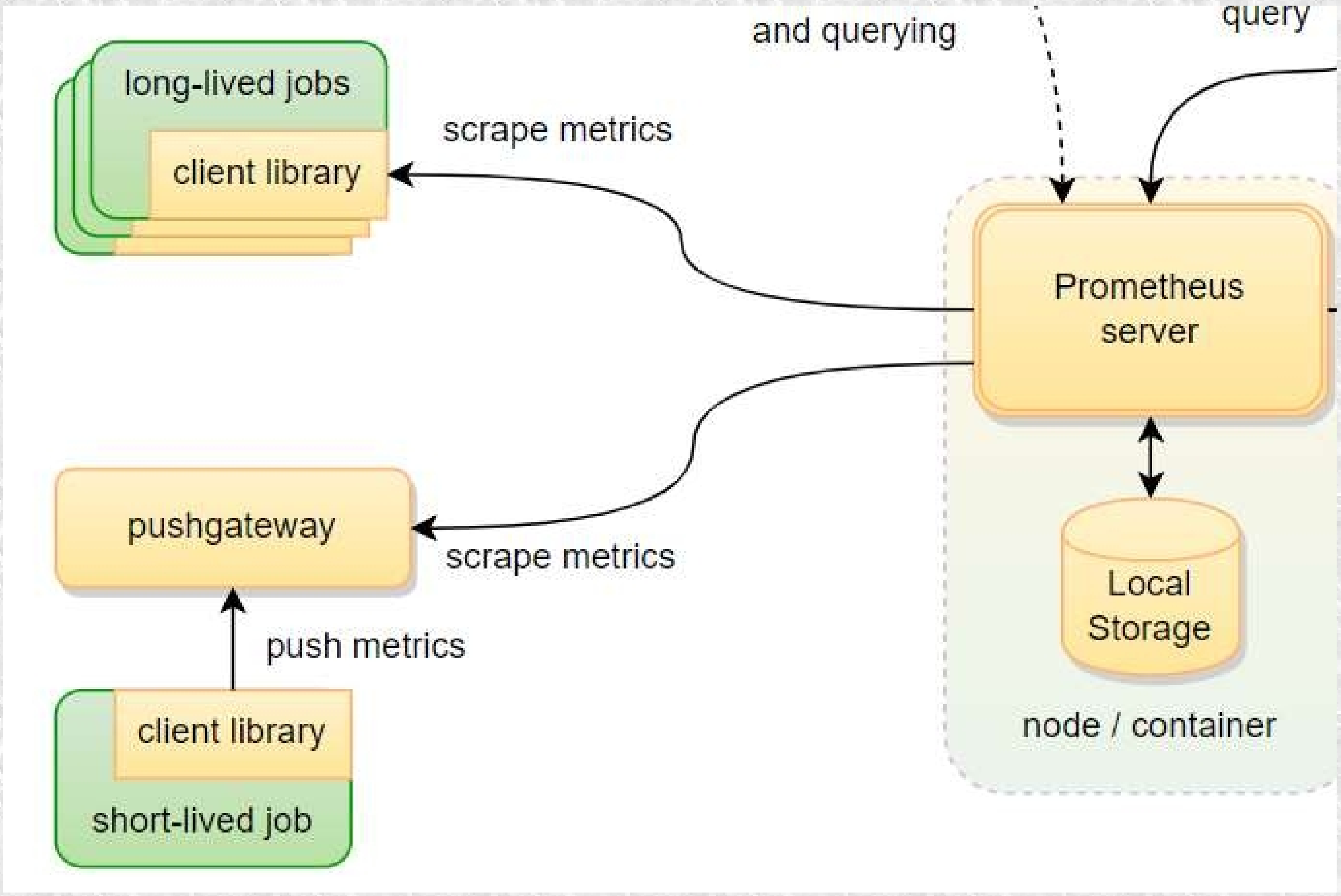
- Specialized adapters to expose metrics for specific technology components.
  - Installed and configured for a specific component
  - Scraped by Prometheus based on config file prometheus.yml that references the endpoint exposed by the exporter.
- Exporters (https://prometheus.io/docs/instrumenting/exporters/)
  - 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, ...



Exposing metrics for Prometheus

Pushgateway for Short-Lived Jobs

- Jobs that may be gone before their metrics are scraped.



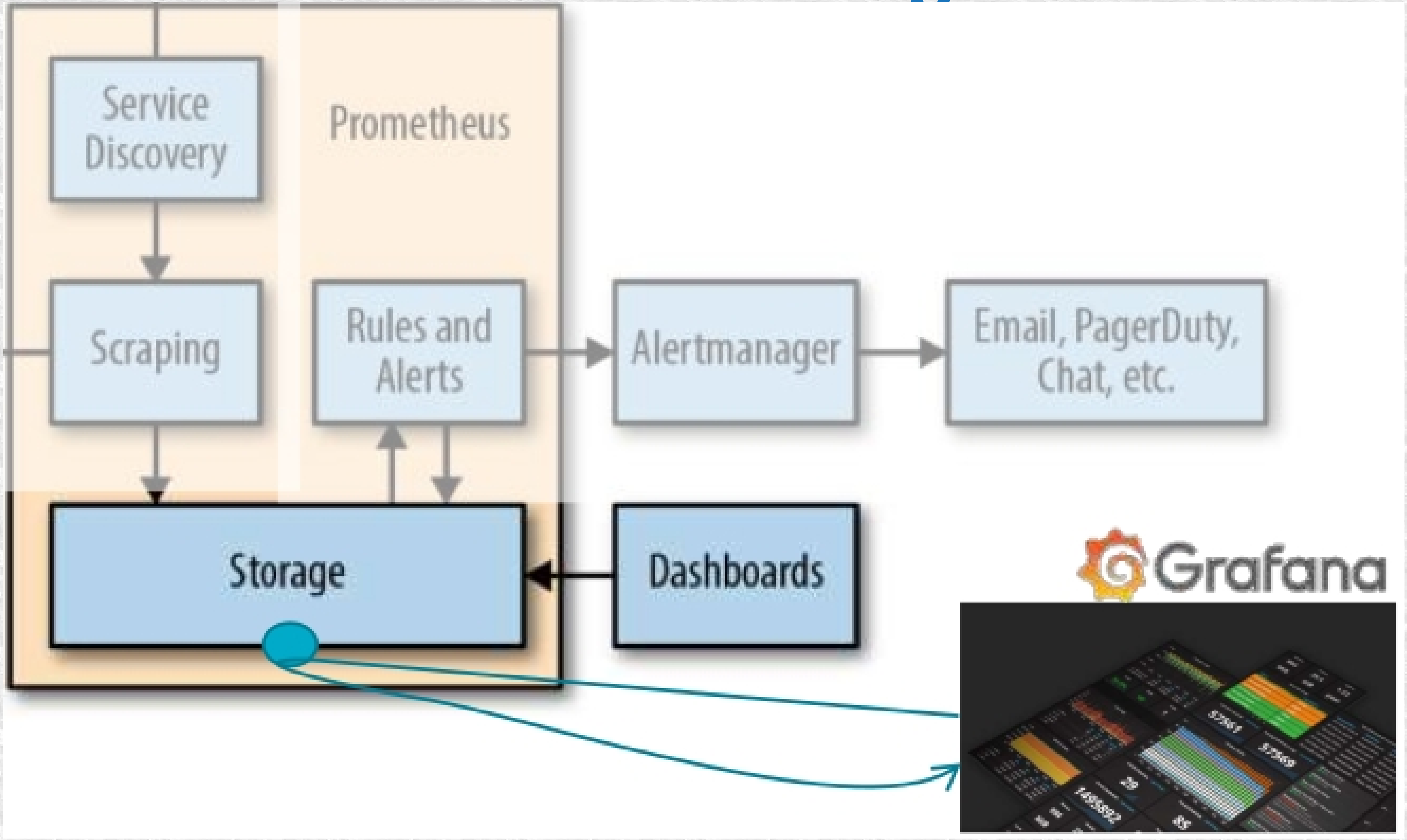
Prometheus Alertmanager

Prometheus alerting

- Rules specify alert conditions
  - Expressed in Prometheus metrics
- Prometheus evaluates these rules –
  - As time progresses
  - As metrics get updated
- An alert is be triggered
  - When the expr evaluates to true
  - For at least as long as is specified in the for condition
- A triggering alert
  - Can be found in the Prometheus Web UI
  - Is notified to the Alert Manager
- The Alert Manager is configured for action
  - For example: send notification via specific communication channels

Grafana

Dashboarding



Grafana
What is it ?

- A generic open source dashboard product
  - Supporting many types of data sources, of which Prometheus is but one
- Grafana queries data sources (such as Prometheus) periodically
  - Does not store any data
  - Refreshes visualizations
  - Evaluates alert conditions and triggers alerts/sends notifications
- Extensive library of pre-built dashboards available
  - Also plugins
- Supports user authentication and authorization and multi-tenancy.

Module 11

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.

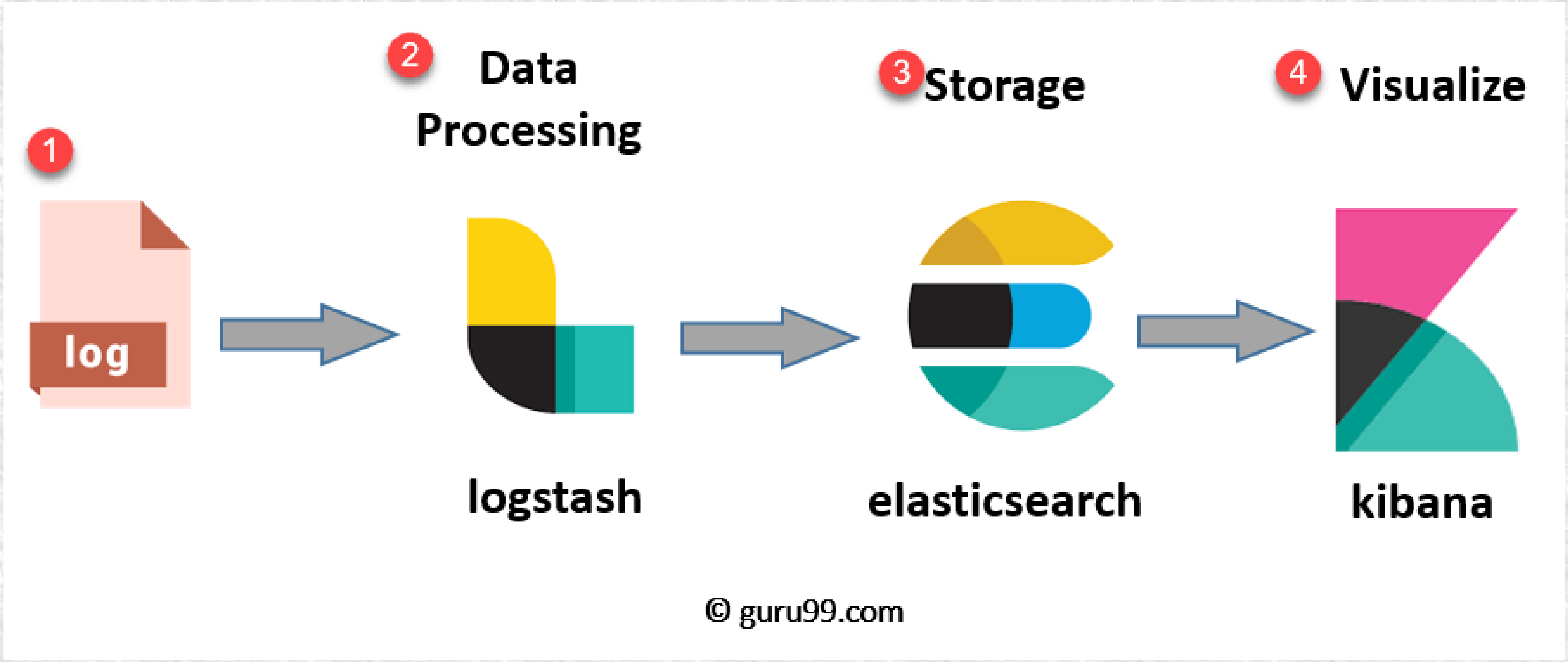
ELK stack

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 visutalization 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

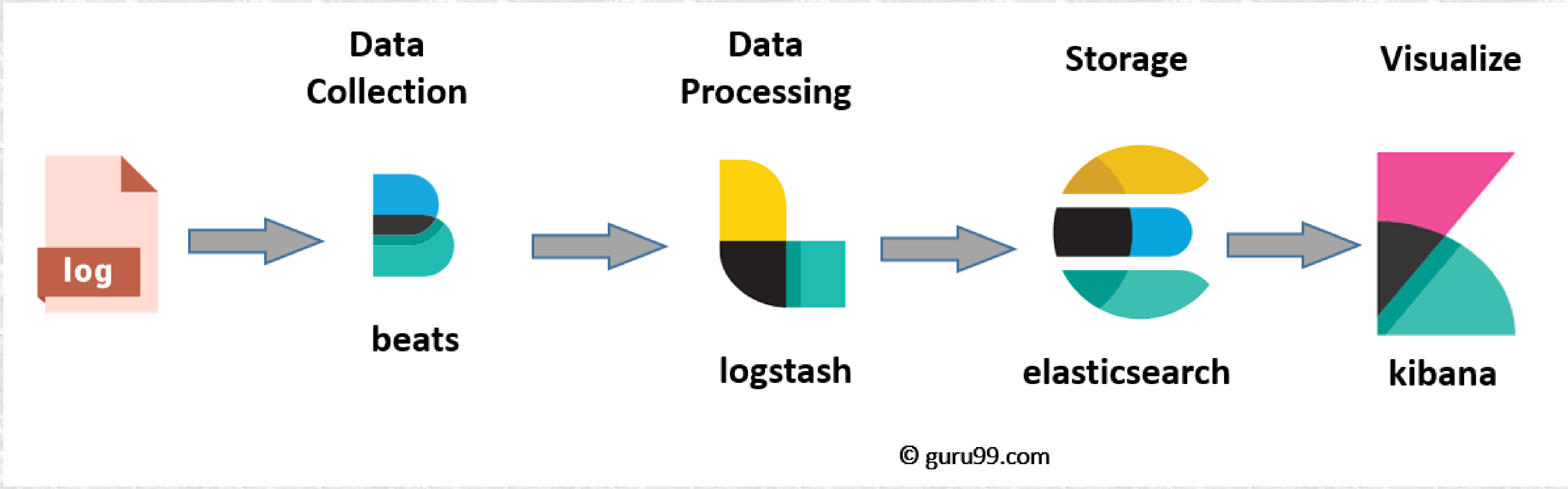
Architecture



ELK stack

Architecture

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



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.

Elasticsearch
Used terms

- Cluster : A collection of nodes which together holds data and provides joined indexing and search capabilities.
- Node : An elasticsearch Instance. It is created when an elasticsearch instance begins.
- Index : A collection of documents which has similar characteristics. e.g., customer data, product catalog.
  - It is very useful while performing indexing, search, update, and delete operations.
- Document : The basic unit of information which can be indexed. It is expressed in JSON (key: value) pair. '{"user": "nullcon"}'.
  - Every single Document is associated with a type and a unique id.

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.

Logstash
Grok filter

- Logstash grok filter used to parse unstructured data into structured data.
- It match a line against a regular expression, map specific parts of the line into dedicated fields, and perform actions based on this mapping.
- basic syntax format for a Logstash grok filter:

  %{PATTERN:FieldName}

- Example :
  - Log :

    ```
    2016-07-11T23:56:42.000+00:00 INFO [MySecretApp.com.Transaction.Manager]:Starting transaction for session -
    464410bf-37bf-475a-afc0-498e0199f008
    ```

  - Grok pattern :

    ```
    grok {
        match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:log-level} \[%          {DATA:class}\]:%
        {GREEDYDATA:message}" }
    }
    ```

  - Results :

    ```
    {
     "message" => "Starting transaction for session -464410bf-37bf-475a-afc0-498e0199f008",
     "timestamp" => "2016-07-11T23:56:42.000+00:00",
     "log-level" => "INFO",
     "class" => "MySecretApp.com.Transaction.Manager"
    }
    ```

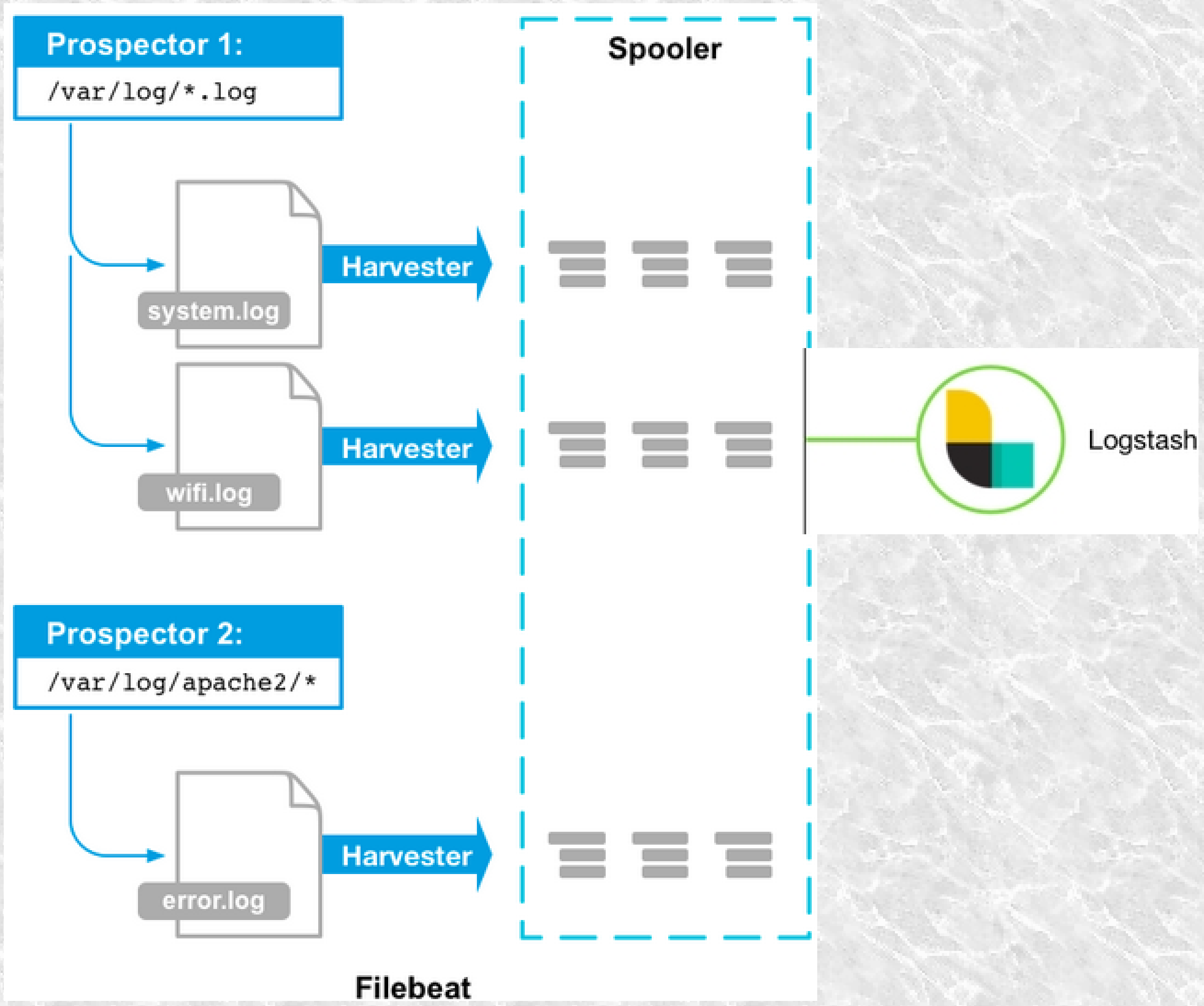
What is Kibana?

- A data visualization which completes the ELK stack.
- Dashboard offers various interactive diagrams, geospatial data, and graphs to visualize complex quires.
- 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.

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}" }
        }
}
```