DevOps – Let the Journey Begin

Ajit Singh

Introduction

Name -

Total Experience -

Background –

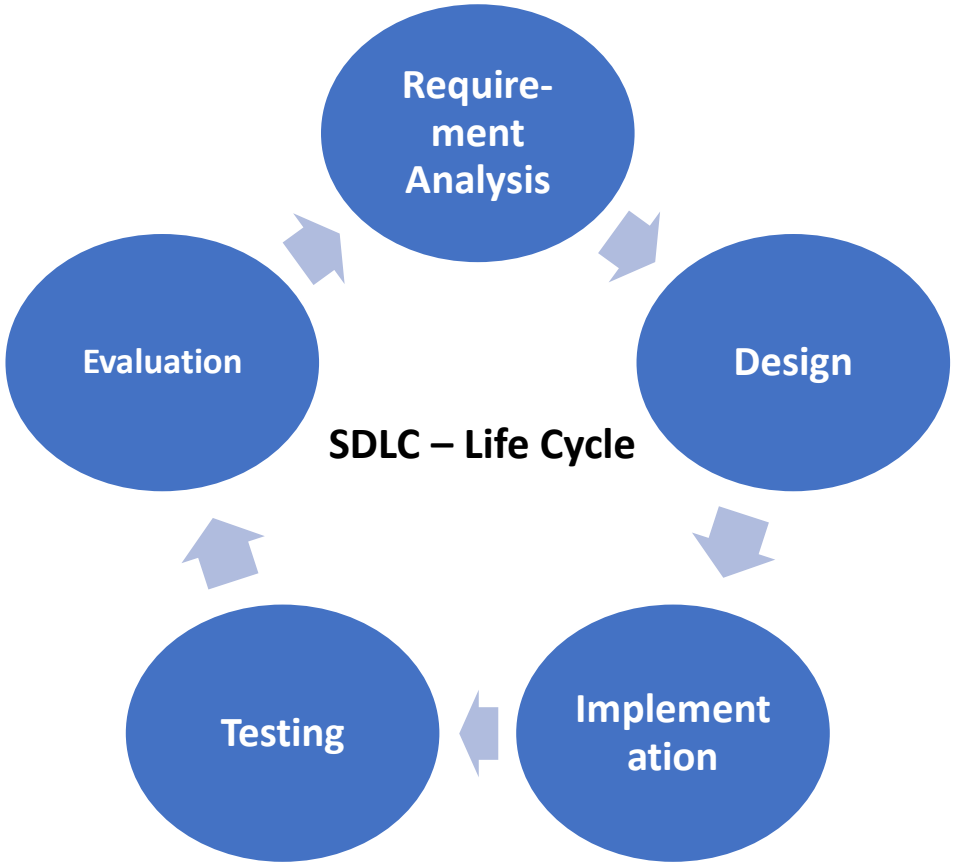
Any Exposure of AWS/Git/Docker/Kubernetes/Jenkins/Terraform/Ansible

DevOps

What is DevOps?

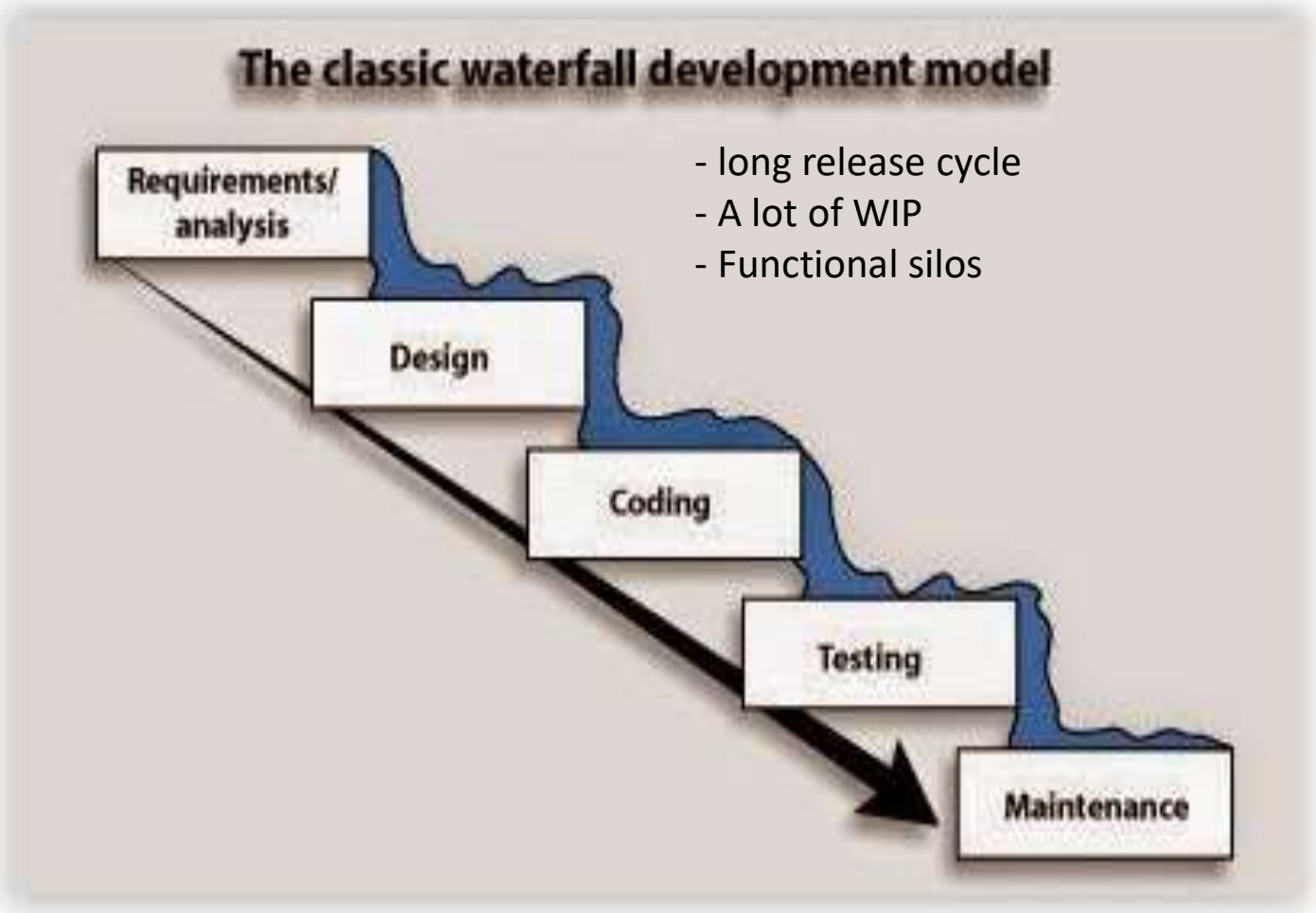
SDLC Model

- A systems development life cycle is composed of **several clearly defined and distinct work phases** which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems

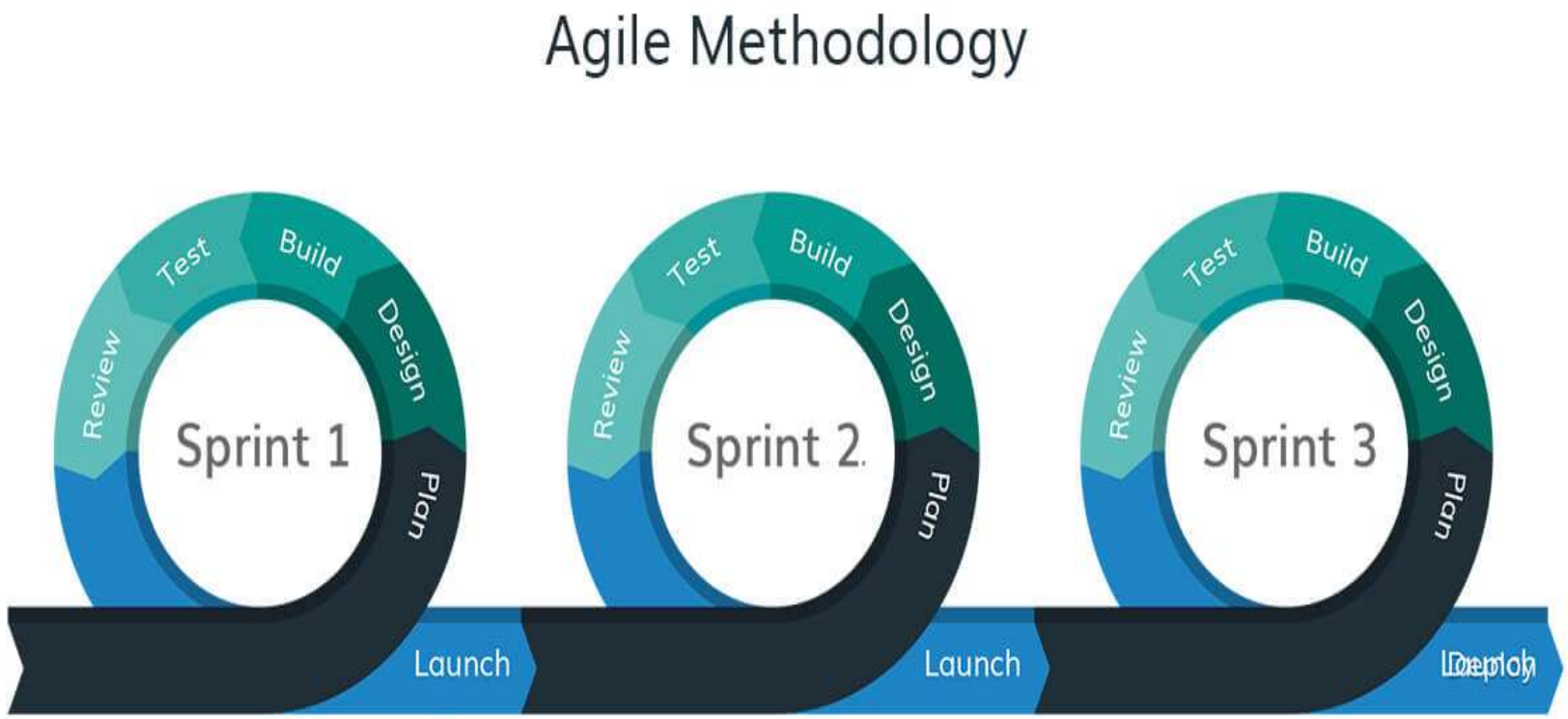


Waterfall Model

1. Determine the Requirements
2. Complete the design
3. Do the coding and testing (unit tests)
4. Perform other tests (functional tests, non-functional tests, Performance testing, bug fixes etc.)
5. At last deploy and maintain

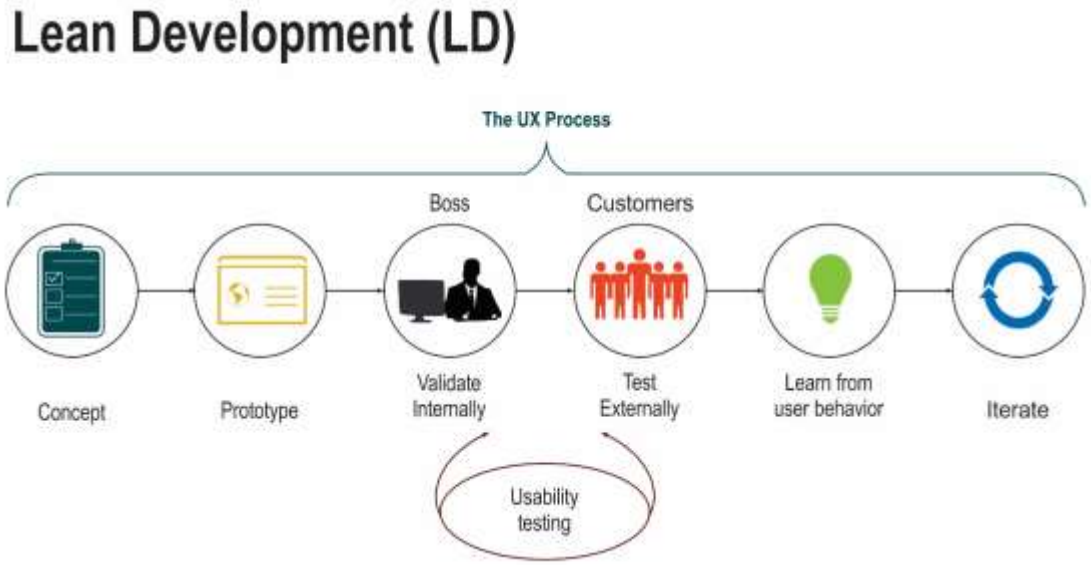


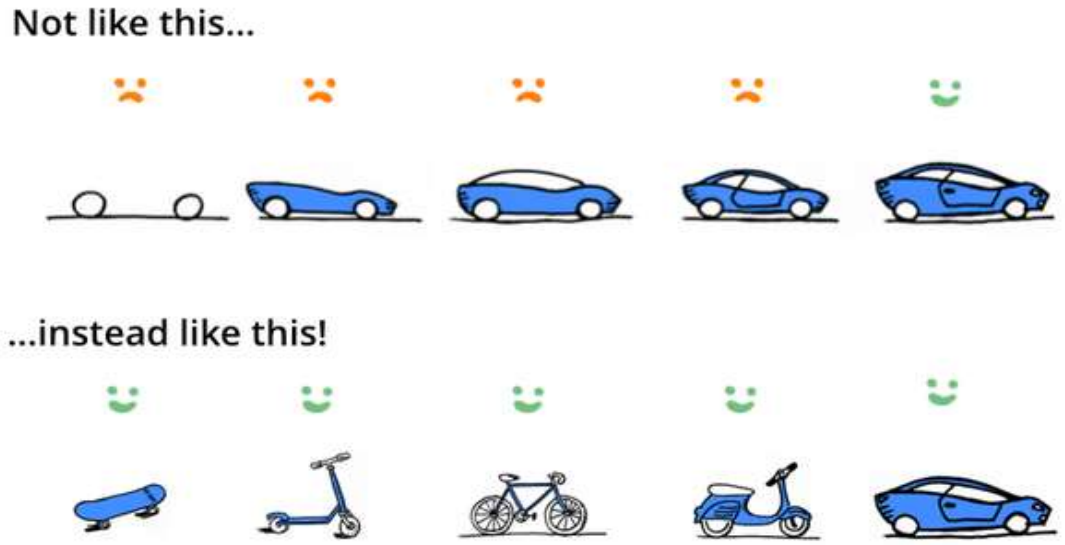
Agile



- Shorter release cycle
- Small batch sizes (MVP)
- Cross-functional teams
- Incredibly agile

Lean Development





- Suddenly ops was the bottleneck (more release less people), again WIP is more!

Challenges

Some of the challenges with the traditional teams of Development and Operations are:

A Typical Case Study

- **Development Team:**

- Monday Morning, the writing of code done, unit tests completed, code delivered to the Integration teams to get the code included in CI builds.
- To get the services tested, a ticket is opened for QA teams

- **Build/Release/Testing/Integration Team:**

- Tuesday Morning, ticket accepted, a tester put an email to the developer asking deployment instructions. There is not automated deployments, developer updated to the tester, lets come online and we will deploy the services to the QA environment together.
- Call started, developer identified the “test environment” is not compatible.
- Tuesday afternoon, a ticket raised in Ops Team with new specifications.

- **Ops Team:**

- Wednesday morning, ticket accepted, specifications checked , a new port open request was identified.
- Ticket raised for Security team, ticket accepted, change approved, port opened, email received by the Ops team the work is done.

A Typical Case Study

- **Ops Team:**

- Identified the provisioning requirements again and started work on building the environment.

- **Build/Release/Testing/Integration Team:**

- Thursday Morning, updates received - the environment is ready. Developer and Tester again on call to deploy new services. Services deployed; tester is running test scripts. Next phase is to run regression test cases. Again a new ticket is raised for new test data with production teams and day ends.

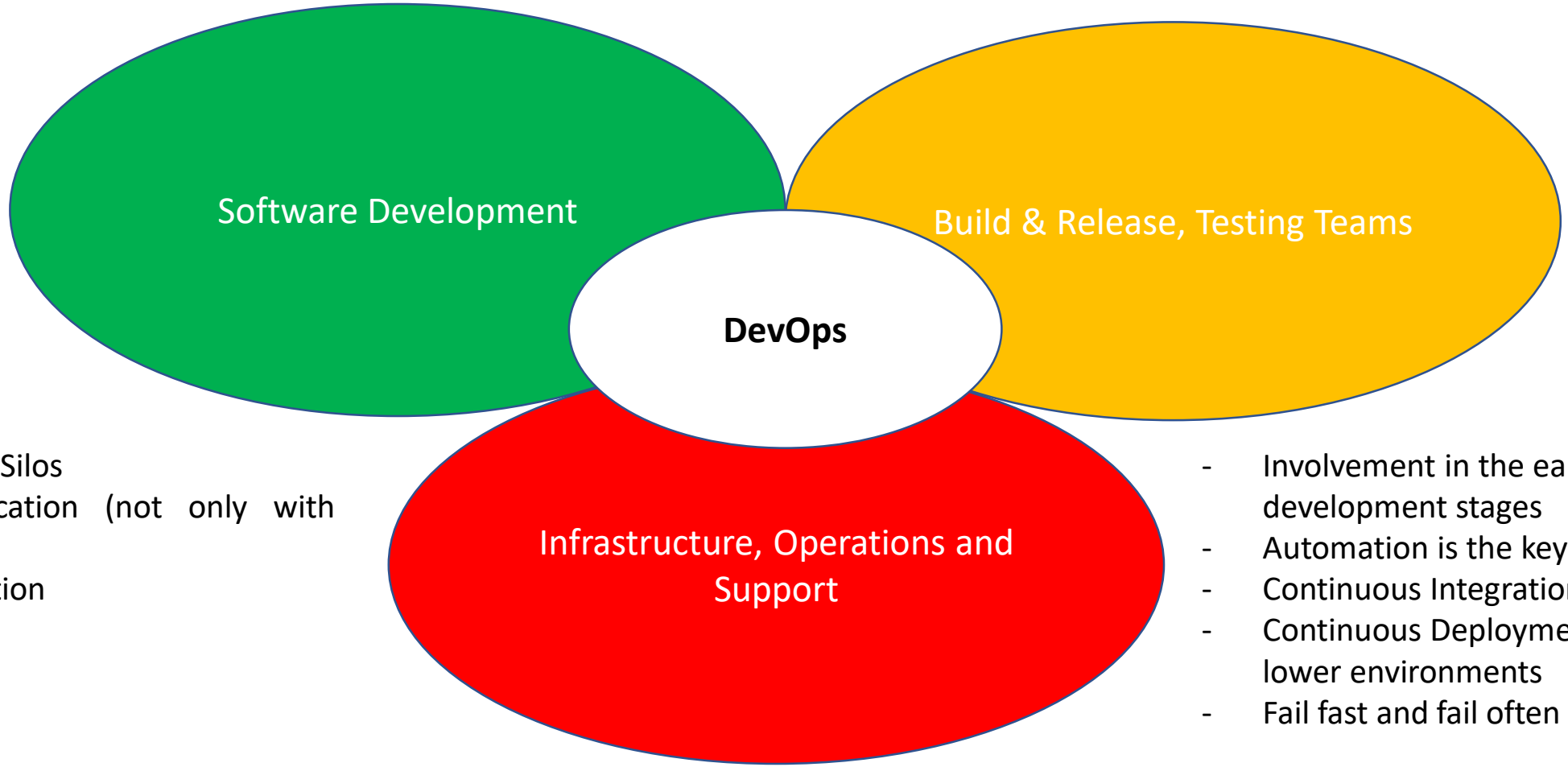
- **Ops Team:**

- Its Friday and the work is not on full swing, ticket accepted but not worked as production team has to complete rest of the works. Somehow the test data is gathered by Friday Evening.

- **Build/Release/Testing/Integration Team:**

- Monday morning, tester gets the data, regression tests run, a defect found, and ticket returned to the development team.

DevOps



- Break the Silos
- Communication (not only with emails)
- Collaboration
- Trust

- Involvement in the early development stages
- Automation is the key
- Continuous Integration
- Continuous Deployments in the lower environments
- Fail fast and fail often

DevOps Model

Collaboration

Communication

Continuity

Automation

Project Management

Customer

Planning

Build

Test

Deploy

Monitoring

Continuous Feedback

Continuous Improvement

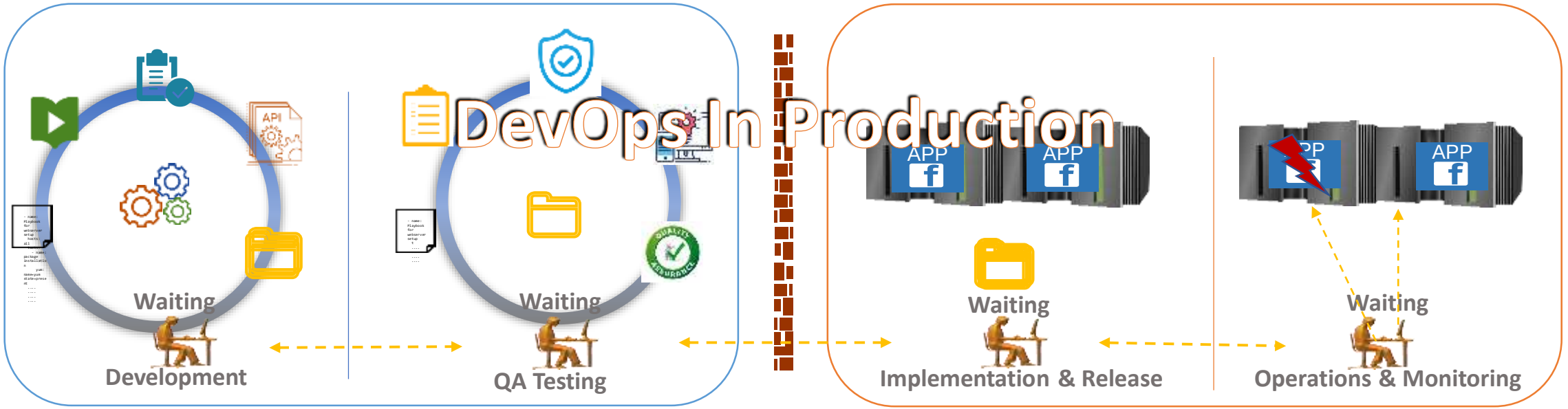
Continuous Planning

Continuous Integration

Continuous Delivery

Continuous Deployment

Continuous Monitoring



DevOps Essence

Efficiency - Faster time to market

Predictability - Lower failure rate of new releases

Reproducibility – Version everything

Maintainability - Faster time to recovery in the event of a new release crashing or otherwise disabling the current system

DevOps Core Principals

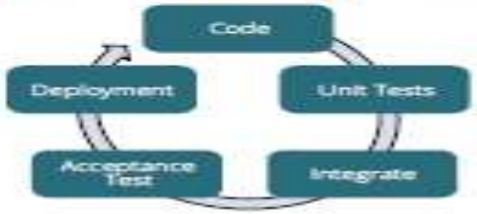
1. Customer-Centric Action

4. Cross-Functional Autonomous Teams

2. Create with the End in Mind

5. Continuous Improvement

3. End-to-End Responsibility



6. Automate Everything You can

How to Build DevOps Organization Culture

Retention is as important as recruitment

Establish Cross-functional team structure

Small teams are better

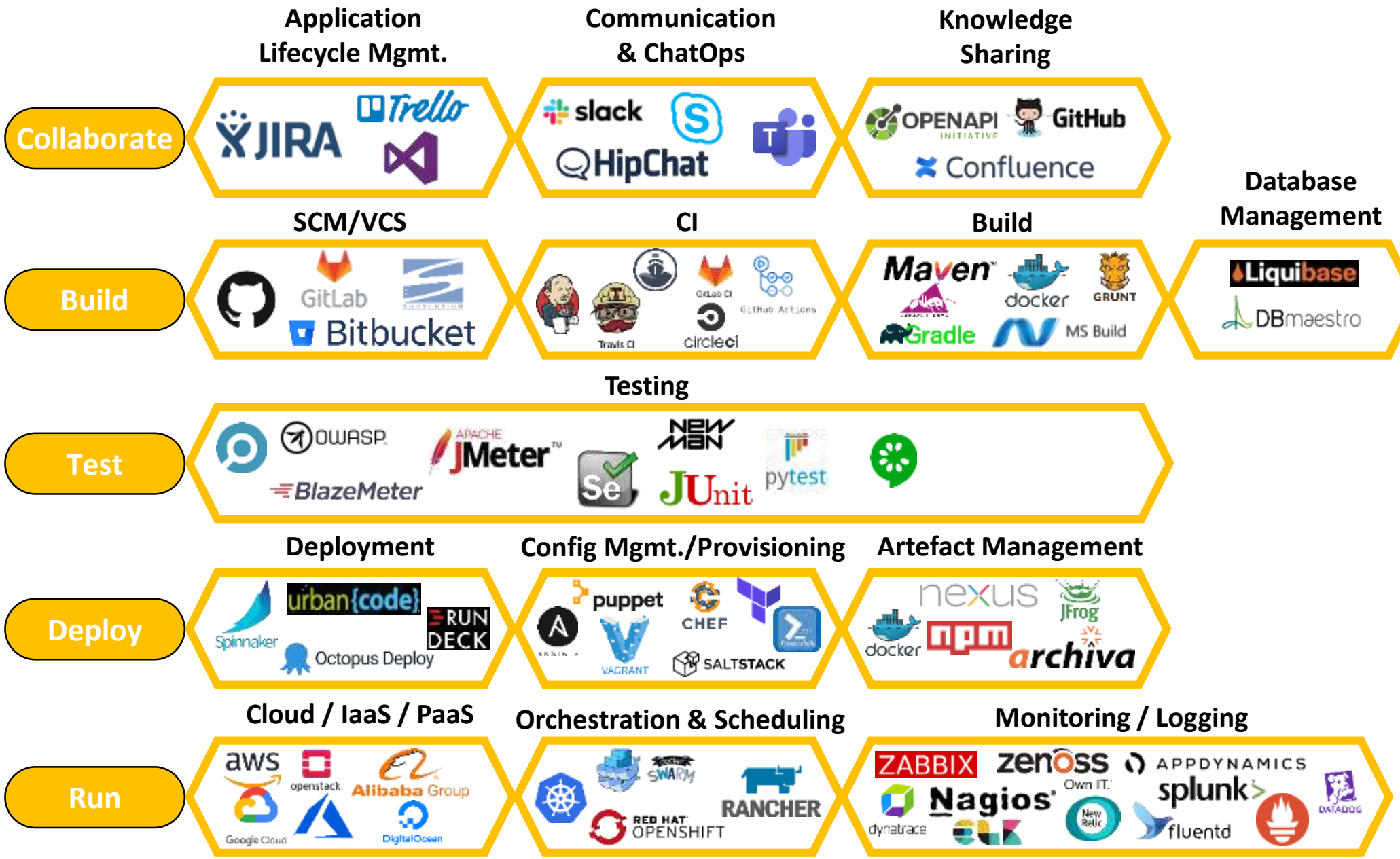
Cool tools can attract and retain

Give autonomy

Automate with existing staffs and give them a chance to learn

Take out few resources from each team, build a new virtual team for automation.

DevOps Toolsets



Introduction to Cloud Computing

What is Cloud?

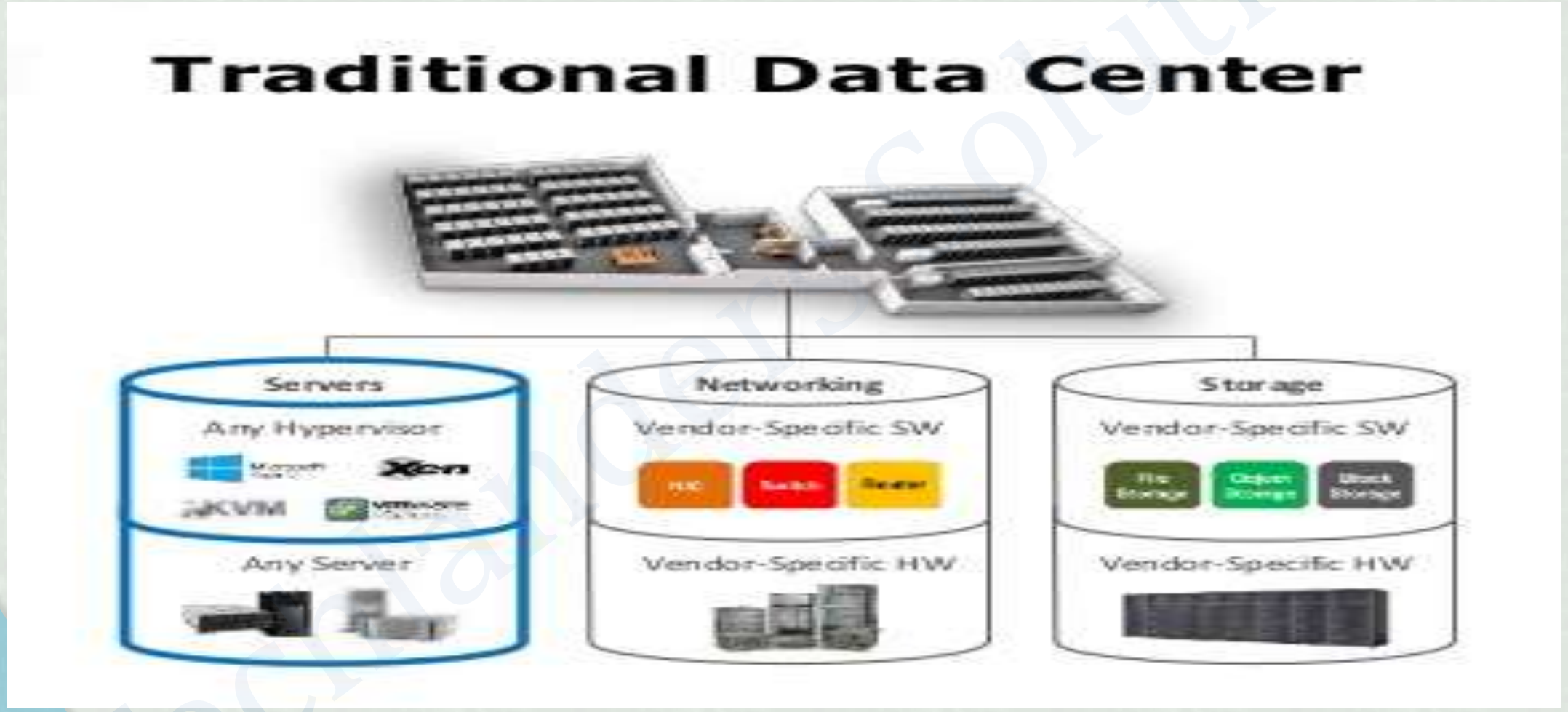
Introduction to Cloud Computing

In simple words, Cloud computing is – Placing your data on someone else's datacenter, letting them manage underline hardware Infrastructure (optionally underline Database or applications too); while having your full control on the data, and accessing that data through Internet or dedicated network.

Cloud computing is a model for enabling **universal**, **on-demand** access to a **shared pool** of configurable computing resources (e.g., computer networks, servers, storage, applications and services), which can be **rapidly provisioned** and **released** with **minimal management effort** on **Pay-per-use basis.**

Free available cloud Examples:	Gmail, IRCTC, WhatsApp/Facebook
Paid available cloud Examples:	AWS, Azure(Microsoft), Oracle Cloud

Traditional DataCenters



Traditional DataCenters

- Main issues with Traditional IT Infrastructure.

- Infrastructure is not a core business
- Hard to Scale
- Dedicated Infrastructure teams
- Dedicated Datacenters
- Dependency on vendors (servers, switches, cables etc.)
- Underutilized Resources
- High Cost
- Difficult Capacity Planning
- On-Spot demands were hard to manage
- Provisioning resources was very time consuming

Why cloud?

- To overcome all of the discussed challenges, IT infrastructure domain drifted towards Service based model which is a real “cloud computing”

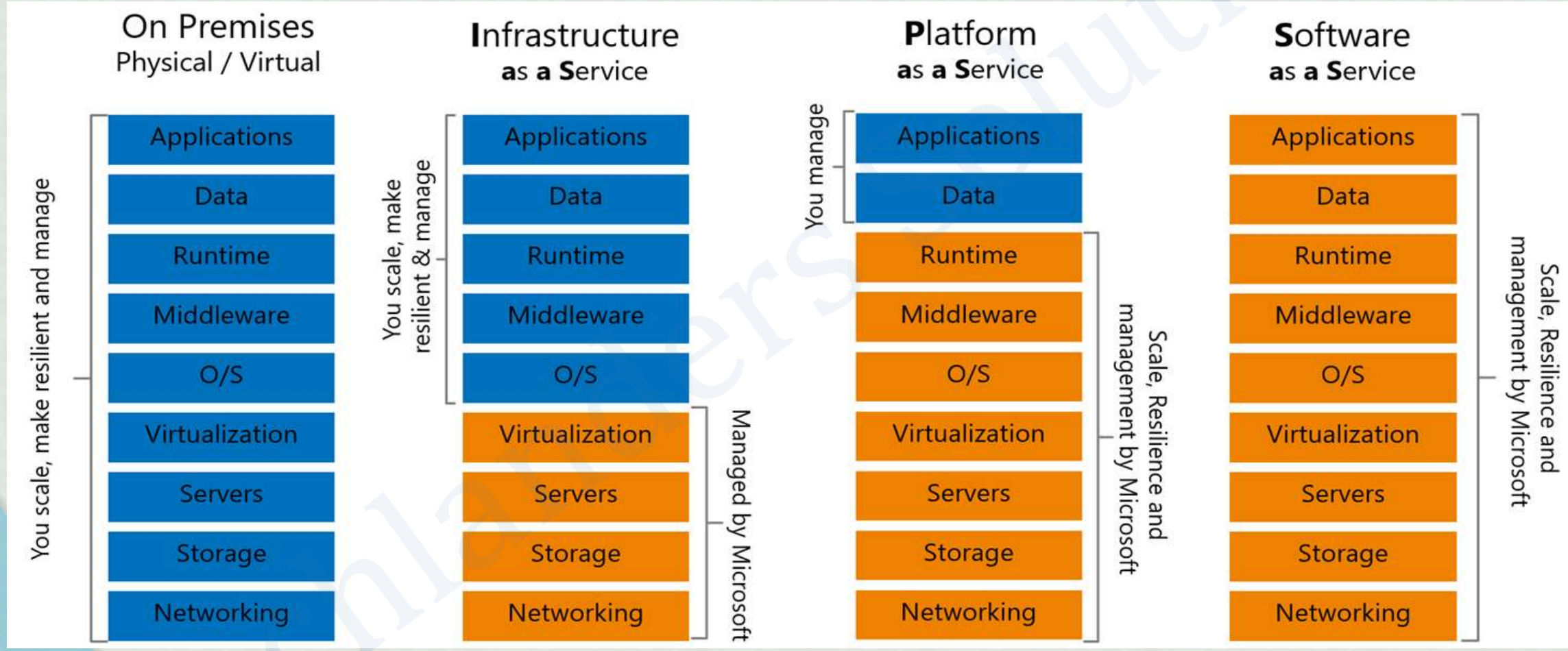
- No Dedicated Datacenter
- No Different Infrastructure Teams
- Higher/Faster Scalability
- Elasticity
- Pay per use model
- Option to adopt high availability
- Better performance
- Instant provisioning
- Optimized use of resources
- On demand scaling to any extent
- No to worry about capacity planning

Cloud Advantages

Cloud Service Models

- There are three Cloud Computing Service Models:
  - Infrastructure as a Service (IaaS)
  - Platform as a Service (PaaS)
  - Software as a Service (SaaS)

Responsibility- Who owns What?



Responsibility- Who owns What?

Cloud Service Models - IaaS

IaaS is the most basic Cloud Service Model

It offers Underline Infrastructure for Compute, Storage and Networking

Infrastructure can be selected by customers as per their choice and Pay-per-use model.

- Examples: Bare metal servers, virtual Instances, Load balancers

IaaS - Benefits

- Drastic reduction in capital investment
- Easily Scalable
- Pay only for the used resources
- High Flexibility
- Reduced infrastructure support teams

Cloud Service Models - PaaS

Another service model, where cloud provider manages the OS & middleware part, along with IaaS

Provide capability to deploy applications on cloud infrastructure without managing underline Infra

Consumers are responsible for managing deployed applications and their environment specific configurations

- Examples: webservers and databases

PaaS - Benefits

- Includes all IaaS benefits
- No upfront licensing cost
- More reduction in Infrastructure support team
- Rapid time to market

Cloud Service Models - SaaS

SaaS deliver complete application to the consumers over the internet.

Consumers are not responsible for managing any application or underlying infrastructure.

SaaS application are delivered as “one-to-many” model.

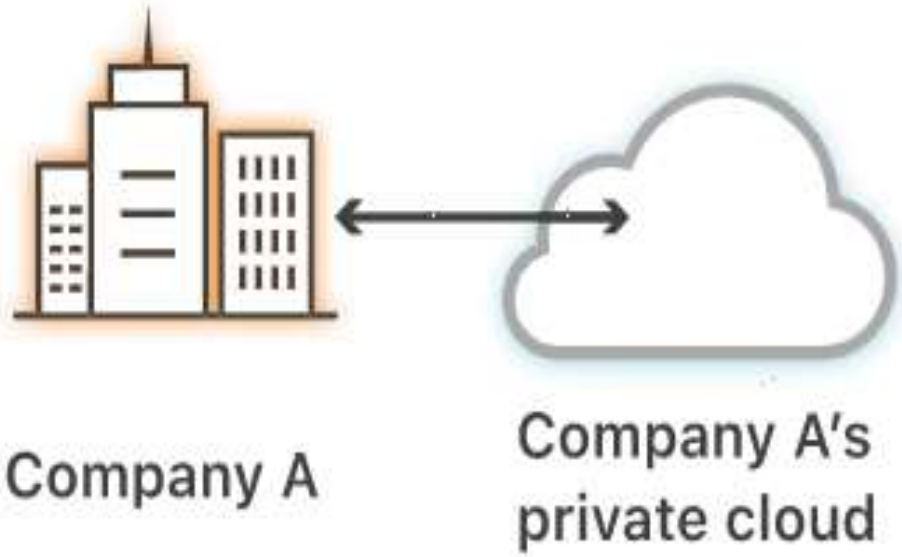
- Examples: office365, Gmail, WhatsApp, JIRA, GIT, Service Now

SaaS - Benefits

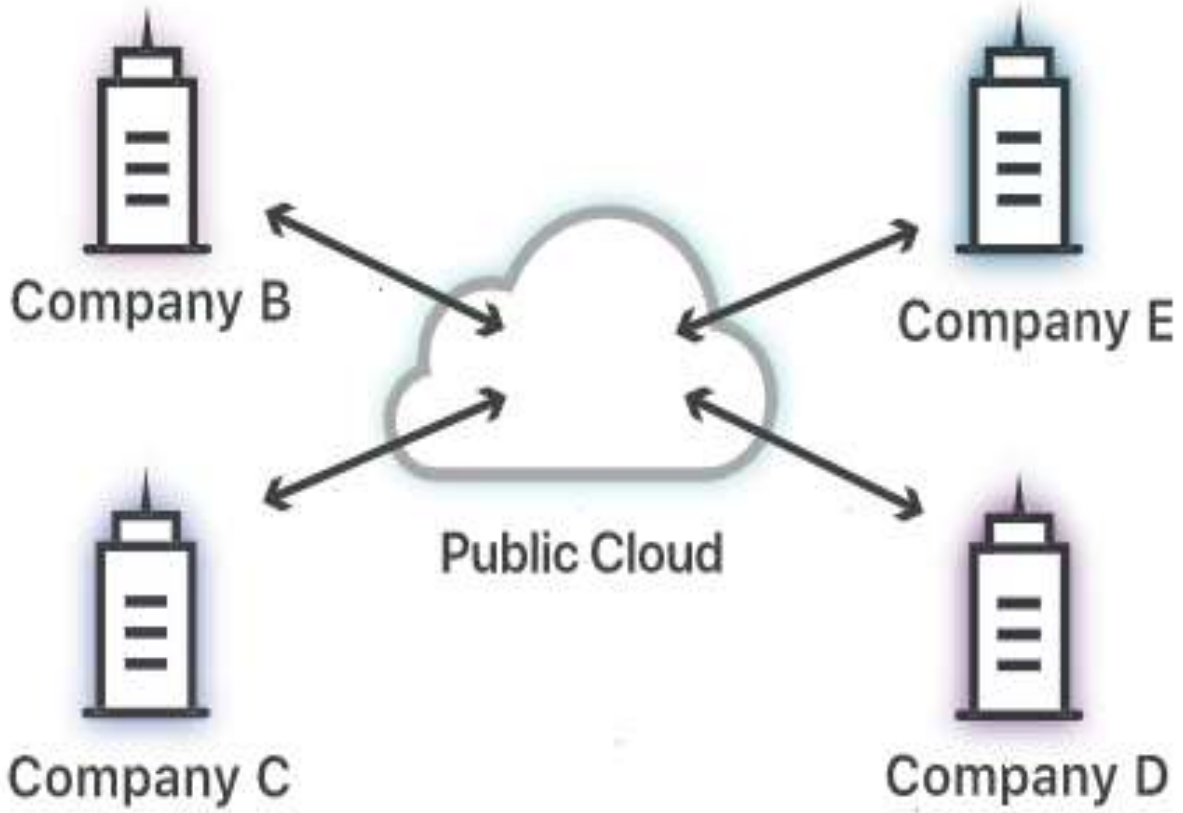
- Includes all discussed benefits which we get in PaaS
- Ability to access from anywhere
- Ability to access from multiple devices
- No installations and maintenance requirements
- No Application management/Licensing Required

Cloud Essentials Characteristics

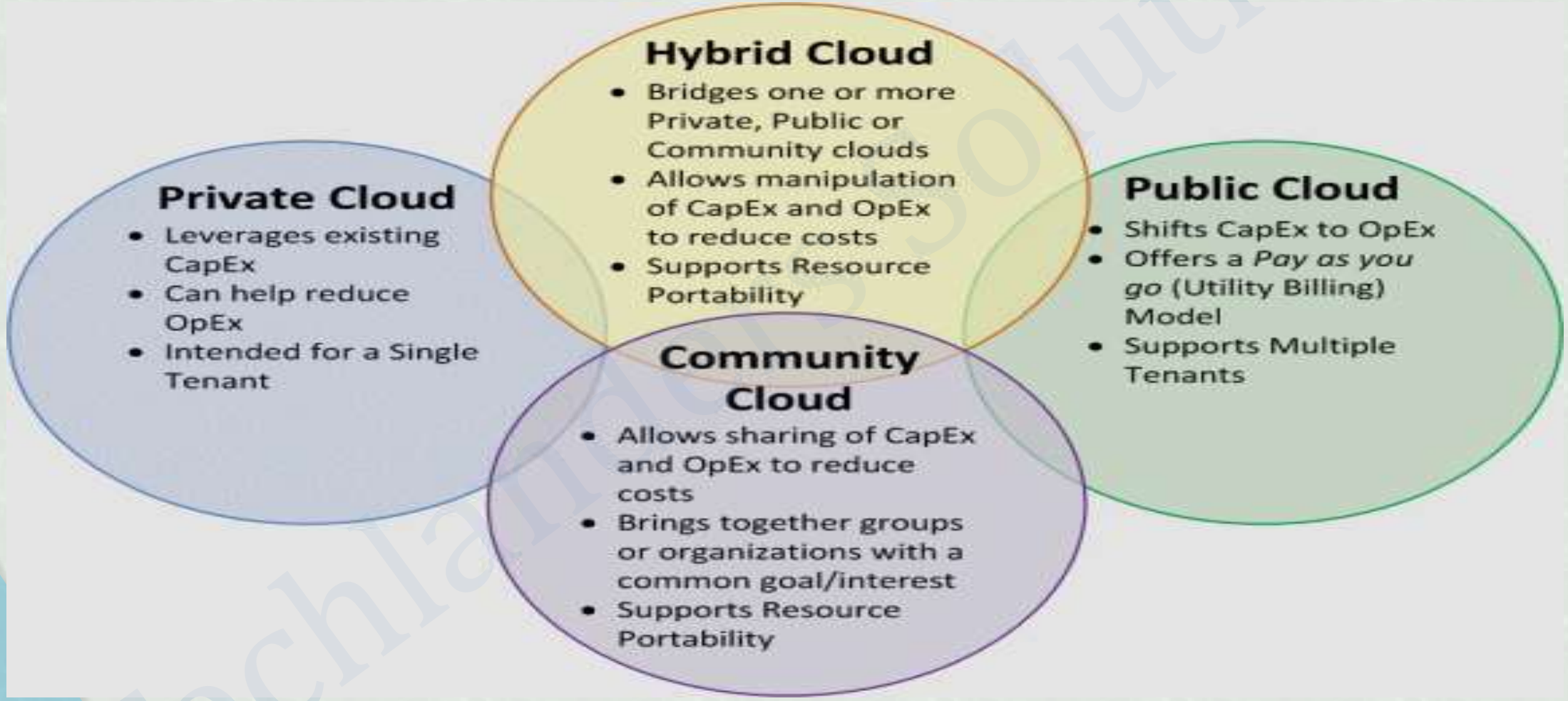
Private cloud



Public cloud shared by multiple companies



Cloud Deployments Types



Cloud's Major Use Cases

On and Off

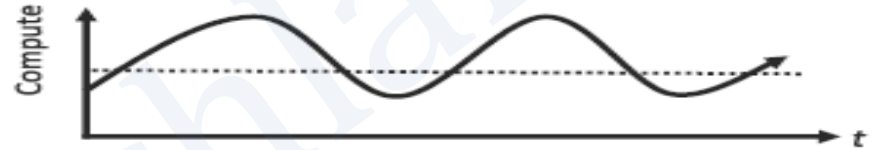
On and off workloads (e.g. batch job)
Over provisioned capacity is wasted
Time to market can be cumbersome

Growing Fast

Successful services needs to grow/scale
Keeping up with growth is a big IT challenge
Cannot provision hardware fast enough

Unpredictable Bursting

Unexpected/unplanned peak in demand
Sudden spike impacts performance
Cannot over provision for extreme cases

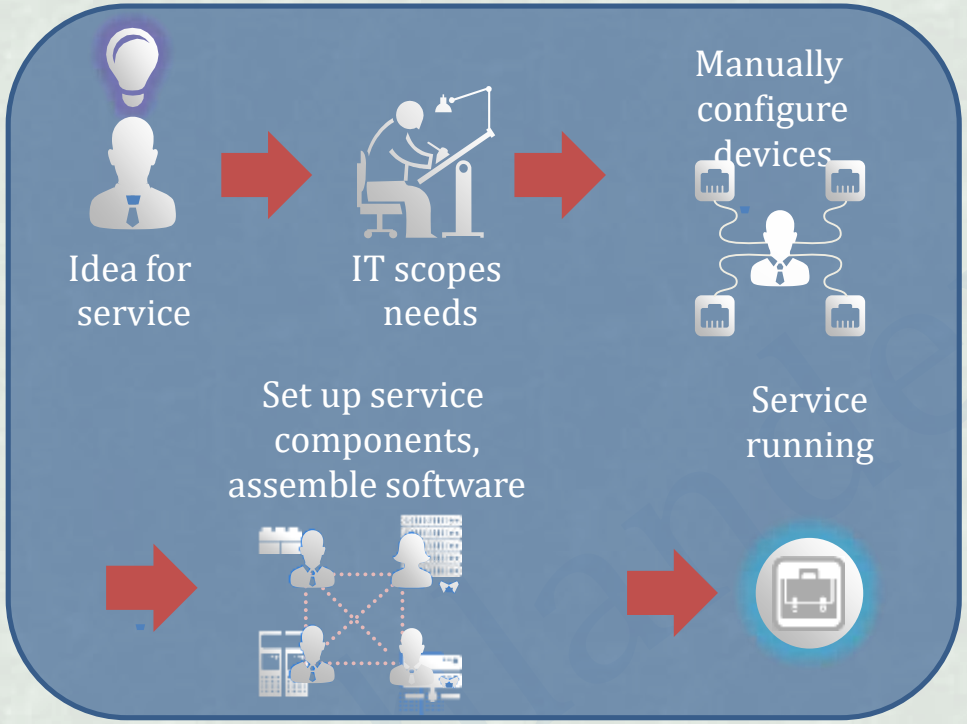


Predictable Bursting

Services with micro seasonality trends
Peaks due to periodic increased demand
IT complexity and wasted capacity

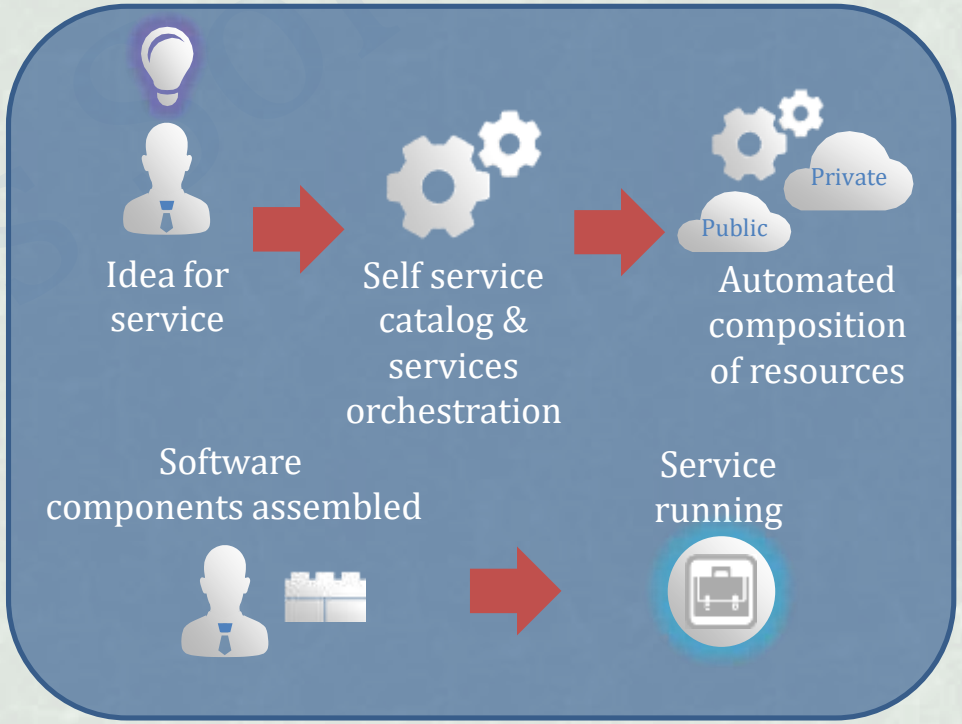
Business Impact of Cloud

Traditional Datacenter



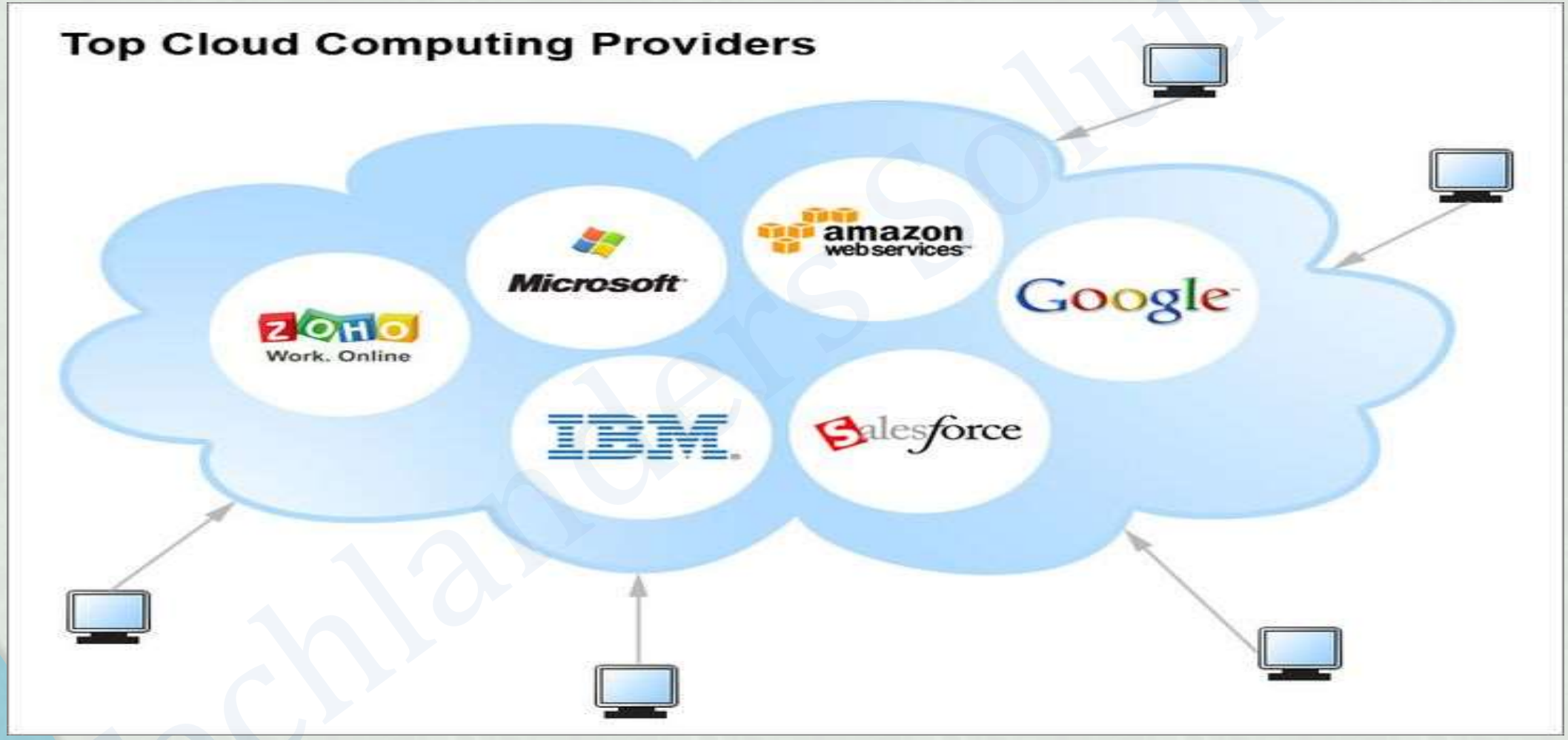
Time to Provision New Service: Months

Cloud Infrastructure



Time to Provision New Service: Minutes

Major Cloud Vendors



Knowledge Checks

- Which Service Level (IaaS, PaaS, SaaS) provides you most control?
- What is Hybrid Cloud?
- Can two public clouds be connected?
- Connecting two public clouds, will be know as public cloud or Hybrid?
- Cloud provided Database, is a PaaS or SaaS?

AWS
(Amazon Cloud)

Amazon Web Services

- AWS (Amazon Web Services) is a group of web services (also known as cloud services) being provided by Amazon since 2006.
- AWS provides huge list of services starting from basic IT infrastructure like CPU, Storage as a service, to advance services like Database as a service, Serverless applications, IOT, Machine Learning services etc..
- Hundreds of instances can be build and use in few minutes as and when required, which saves ample amount of hardware cost for any organizations and make them efficient to focus on their core business areas.
- Currently AWS is present and providing cloud services in more than 190 countries.
- Well-known for IaaS, but now growing fast in PaaS and SaaS.

Why AWS?

- **Low Cost:** AWS offers, pay as you go pricing. AWS models are usually cheapest among other service providers in the market.
- **Instant Elasticity:** You need 1 server or 1000's of servers, AWS has a massive infrastructure at backend to serve almost any kind of infrastructure demands, with pay for what you use policy.
- **Scalability:** Facing some resource issues, no problem within seconds you can scale up the resources and improve your application performance. This cannot be compared with traditional IT datacenters.
- **Multiple OS's:** Choice and use any supported Operating systems.
- **Multiple Storage Options:** Choice of high I/O storage, low cost storage. All is available in AWS, use and pay what you want to use with almost any scalability.
- **Secure:** AWS is PCI DSS Level1, ISO 27001, FISMA Moderate, HIPAA, SAS 70 Type II passed. In-fact systems based on AWS are usually more secure than in-house IT infrastructure systems.

AWS Global Infrastructure

AWS Regions:

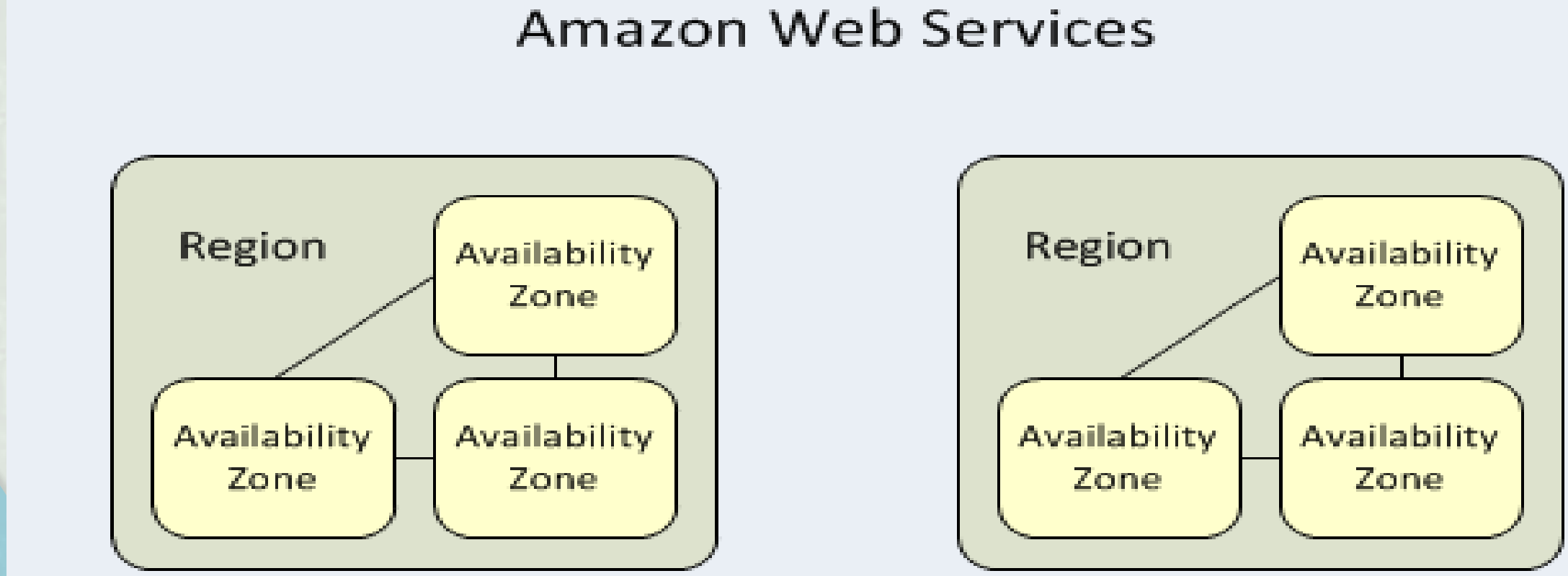
- Geographic Locations
- Consists of at least two Availability Zones(AZs)
- All of the regions are completely independent of each other with separate Power Sources, Cooling and Internet connectivity.

AWS Availability Zones

- AZ is a distinct location within a region
- Each zone is insulated (with low-latency links) from other to support single point of failures
- Each Region has minimum two AZ's
- Most of the services/resources are replicated across AZs for HA/DR purpose.

Note: Resources aren't replicated across regions unless you do so specifically.

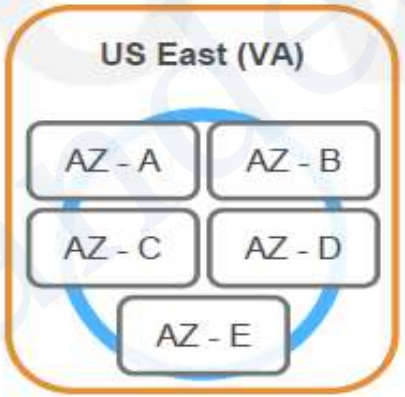
AWS Global Infrastructure

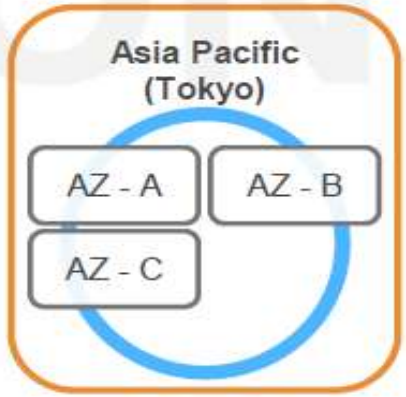


AWS Global Infrastructure

At least 2 AZs per region.

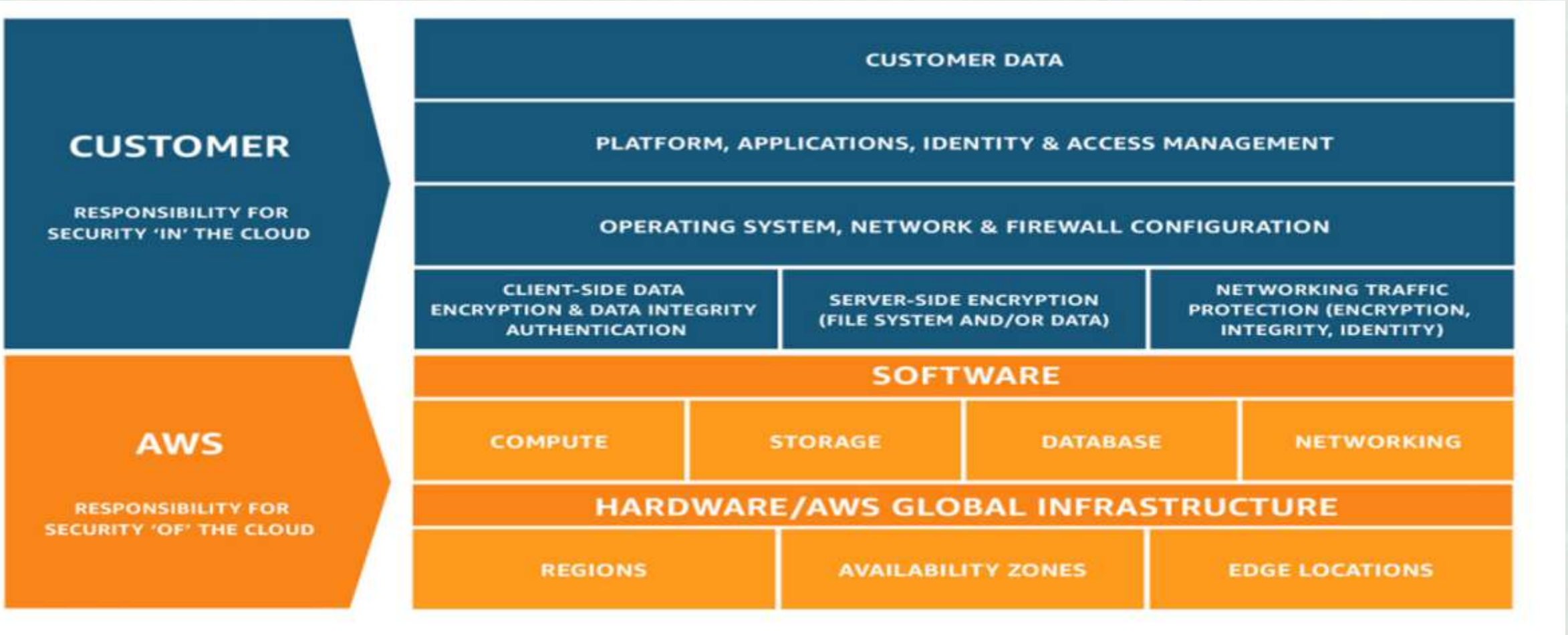
- Examples:
  - US East (N. Virginia)
    - us-east-1a
    - us-east-1b
    - us-east-1c
    - us-east-1d
    - us-east-1e
  - Asia Pacific (Tokyo)
    - ap-northeast-1a
    - ap-northeast-1b
    - ap-northeast-1c



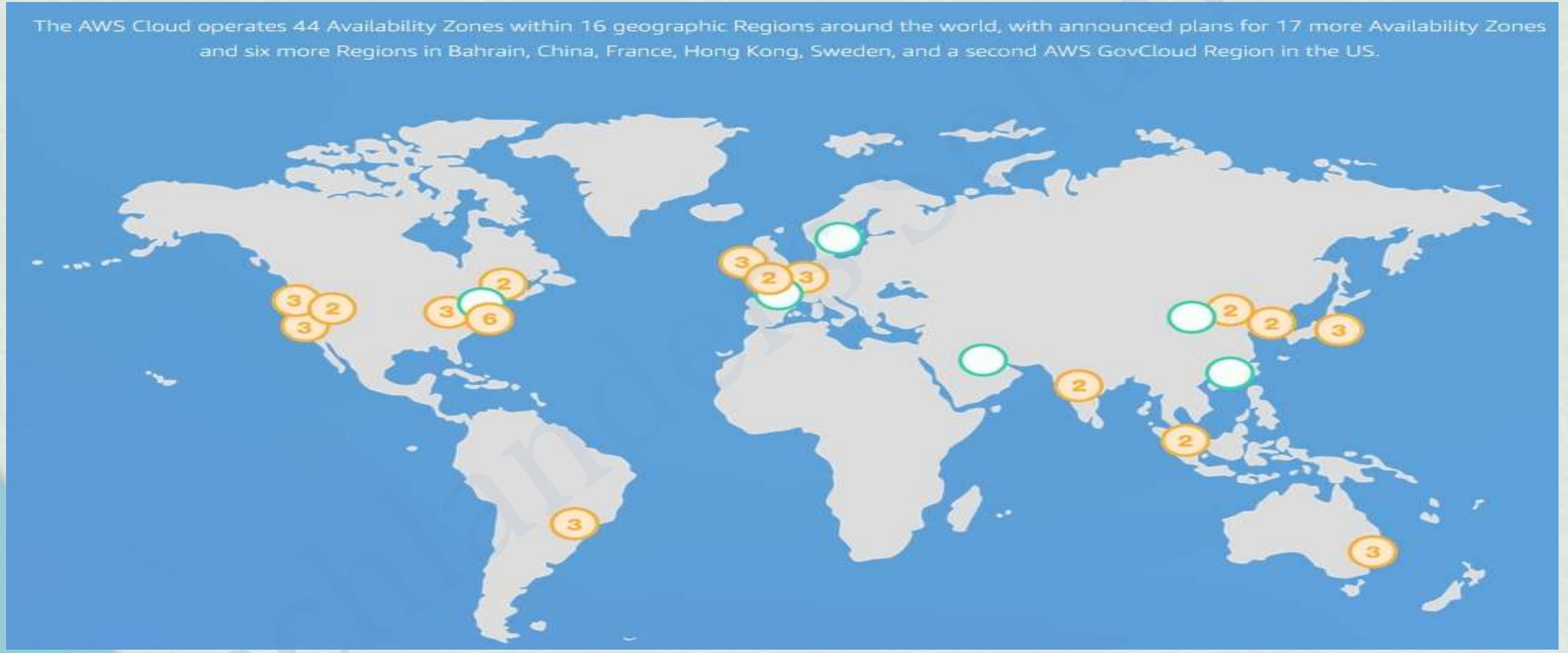


Note: Conceptual drawing only. The number of Availability Zones (AZ) may vary.

Shared Responsibility

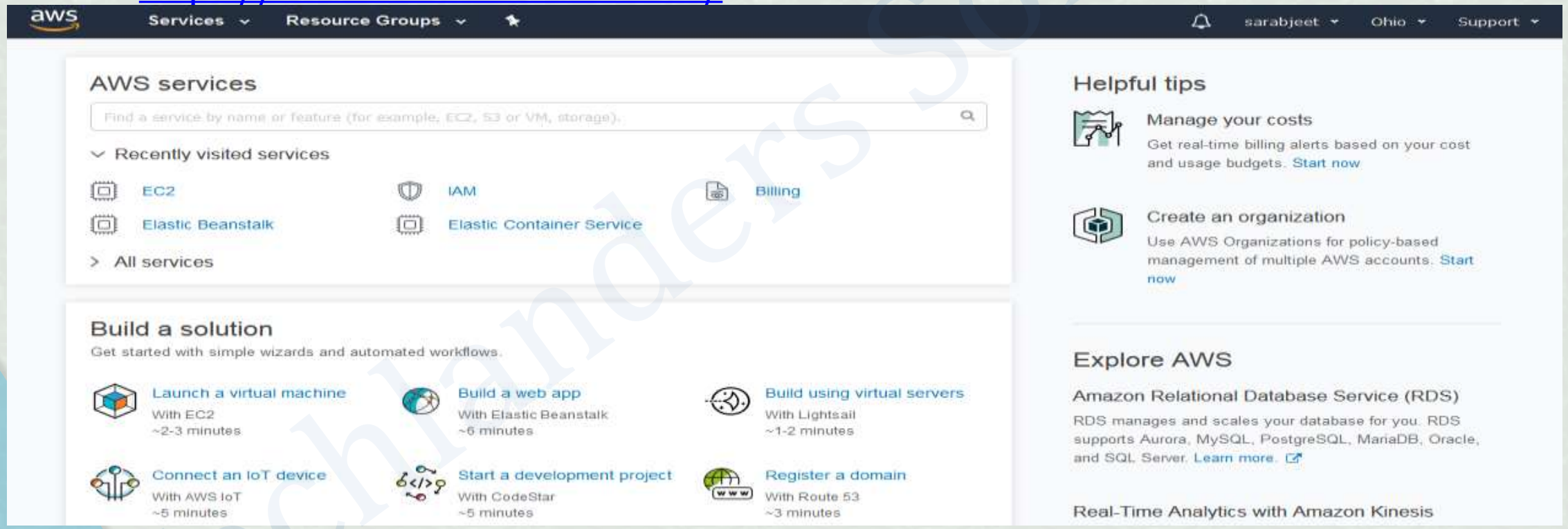


AWS Global Infrastructure



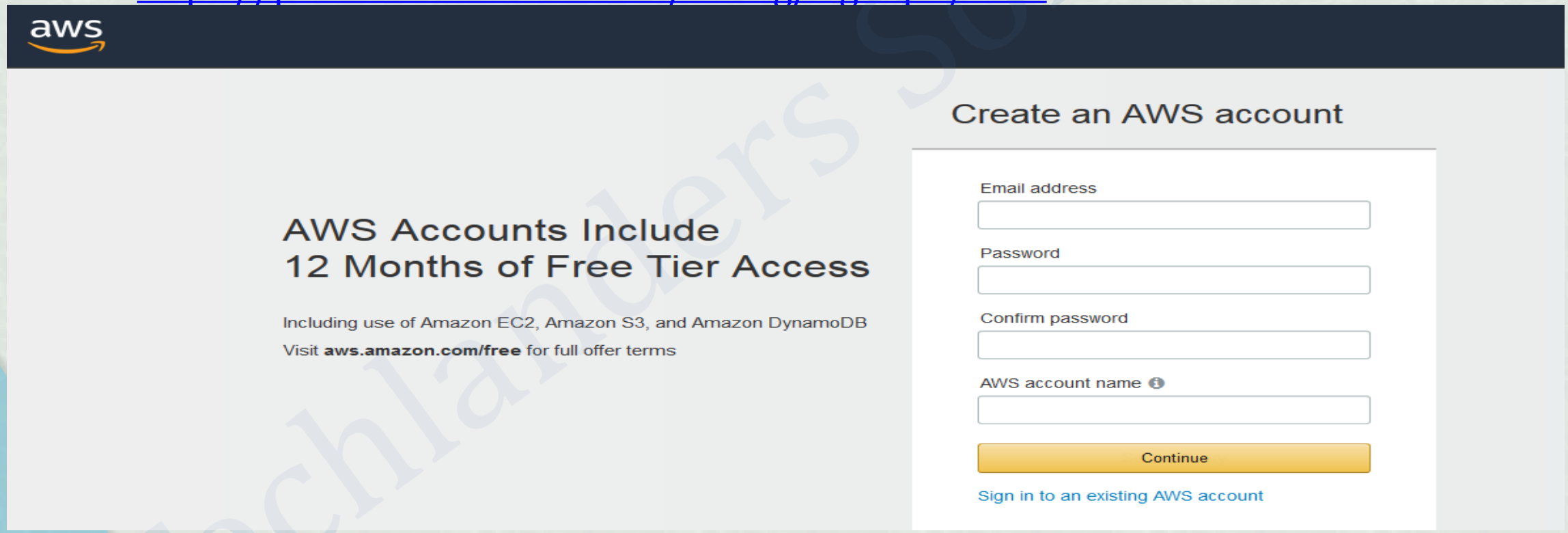
AWS Management Console

- Simple and intuitive web-based user interface.
  - https://console.aws.amazon.com/

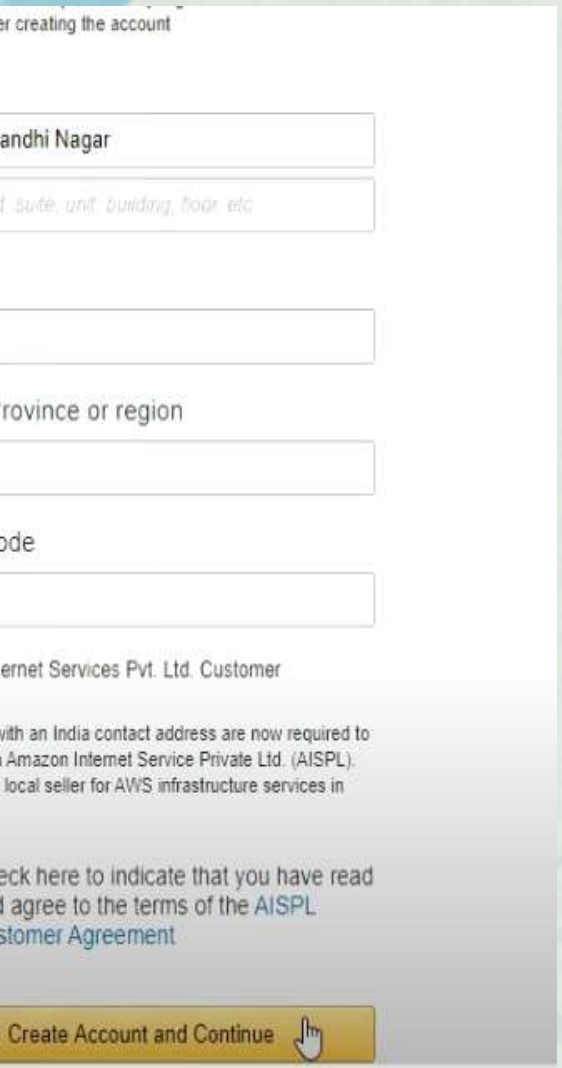


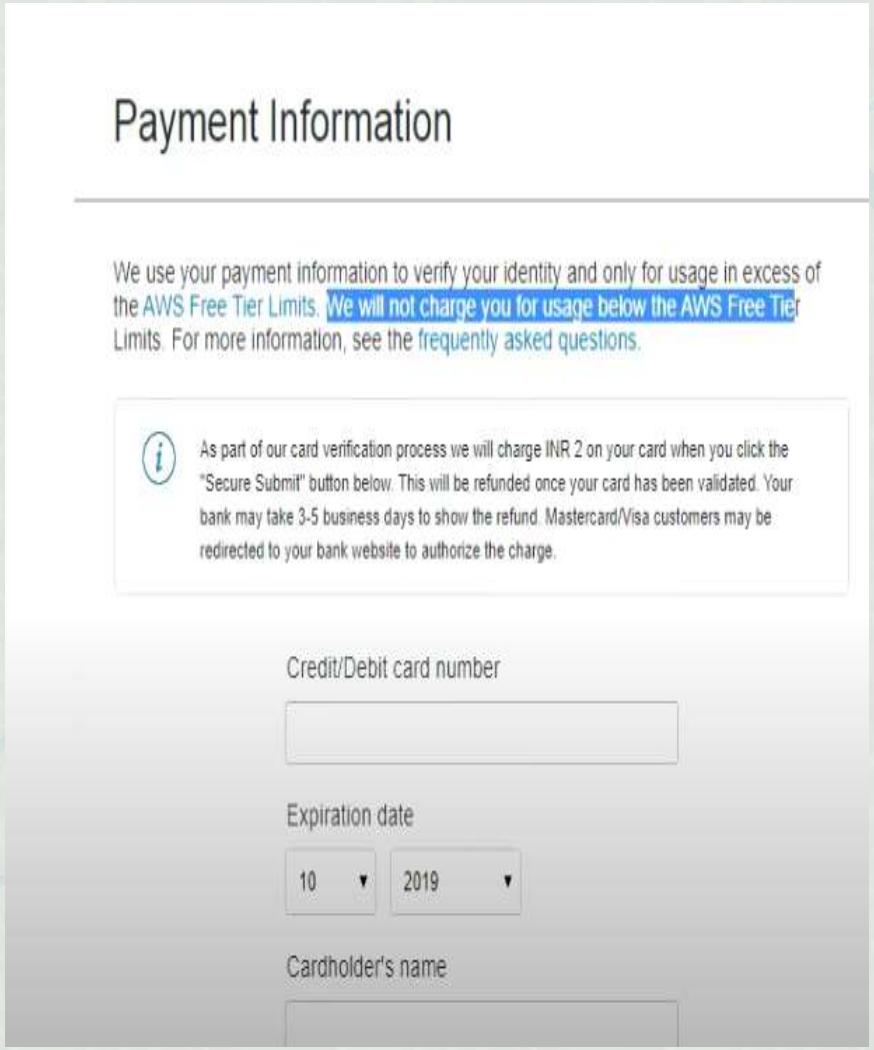
LAB 1 : AWS Signup

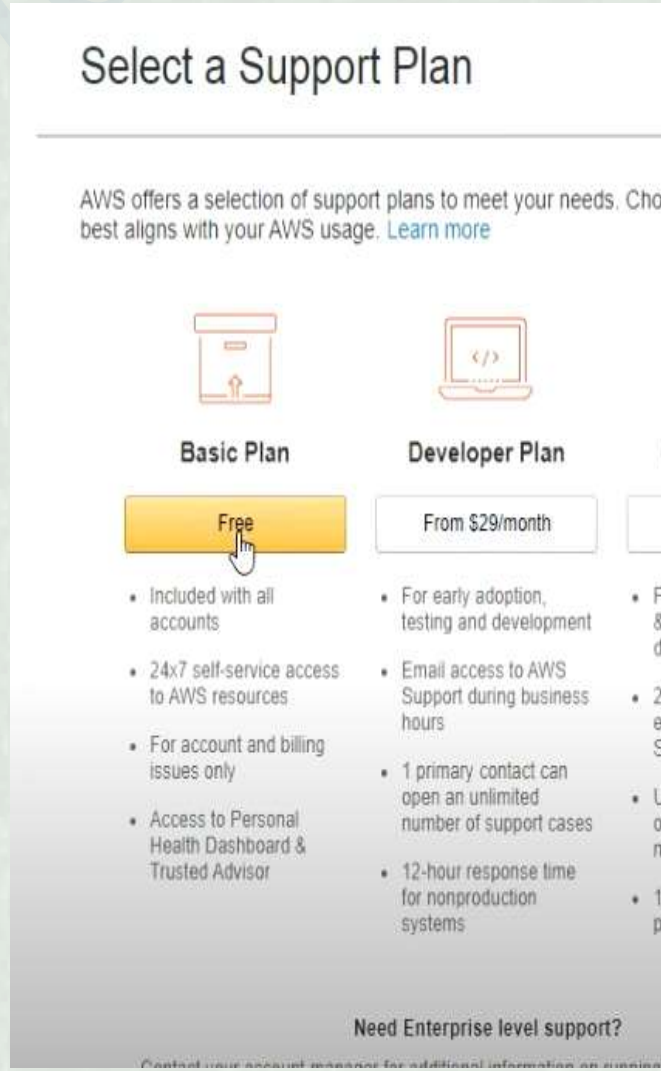
- Create a new account at
  - https://portal.aws.amazon.com/billing/signup#/start



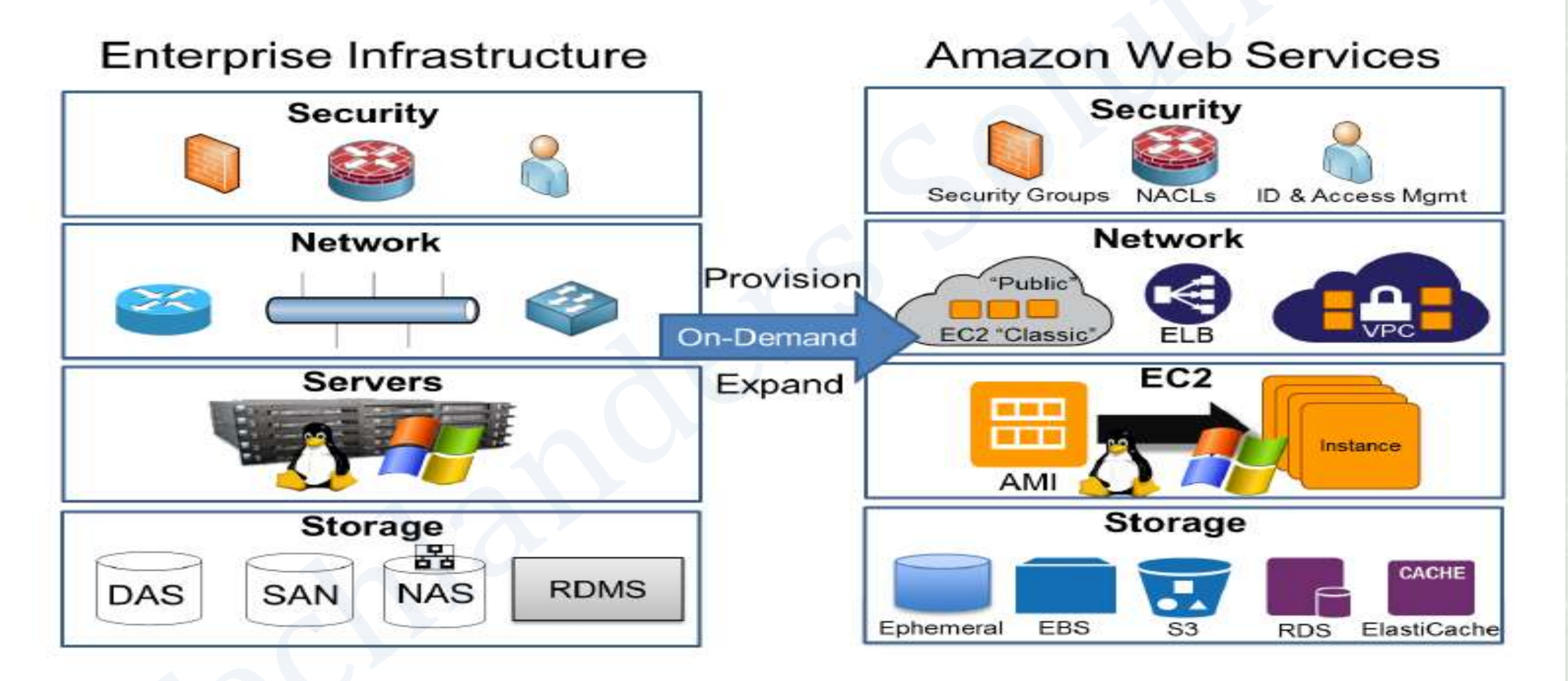
AWS SIGNUP







AWS Core Infrastructure Services



AWS Security

- **Physical Security:**
- 24/7 trained security staff
- AWS data centers in nondescript and undisclosed facilities
- Two-factor authentication for authorized staff
- Authorization for data center access
- Multiple approval based change process

AWS Security

- **Hardware, Software, and Network :**
- Authentication and authorization in place
- RBAC based access control mechanism
- Firewall and other boundary devices
- Security at Server level, Application level and Network level
- AWS monitoring tools
- Services to log AWS resources access

AWS Security

Amazon Resource Names (ARNs)

Amazon Resource Names (ARNs) uniquely identify AWS resources.

We require an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies, API calls etc.

ARN have a specific format:
arn:partition:service:region:account-id:resourcetype/resource

- IAM user name
  arn:aws:iam::123456789012:user/David
- IAM instance id:
  arn:aws:ec2:*region*:*account-id*:dedicated-host/*host_id*
  Eg. arn:aws:ec2:us-east-1:123456789012:dedicated-host/h-12345678

AWS Access Credentials

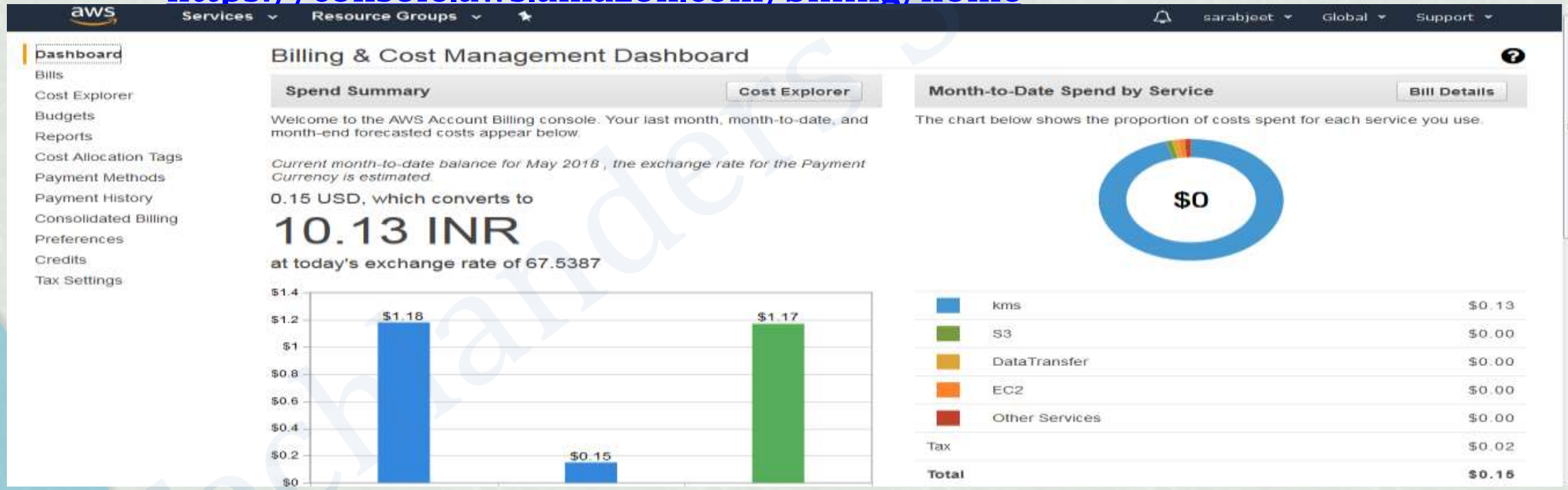
AWS resources can be access using several authentication methods:

- IAM User-id / Password
- Account ID/ AK (Access Key)/ SK (Security Key)
- Certificates
- Key pairs

AWS Billing

AWS Billings history, Previous payments, Current month cost, Budget fixing, Setting usage alarms etc, can be managed from AWS billing page :

https://console.aws.amazon.com/billing/home



AWS Pricing calculators

Simple Monthly Calculator:
You can Estimate your expected monthly bill using Simple Monthly Calculator.
http://calculator.s3.amazonaws.com/index.html

TCO Calculator:
You can Quickly compare the total cost of ownership (TCO) of your **on-premises infrastructure with a comparable AWS deployment** using TCO Calculator and estimate savings you can realize by moving to AWS. https://awstcocalculator.com/#

Cost Explorer:
With Cost Explorer, you can track your actual account usage and bill, at any time using the billing portal. You can view data for up to the last 13 months, forecast how much you are likely to spend for the next three months, and get recommendations for what Reserved Instances to purchase.
https://console.aws.amazon.com/billing/home#/costexplorer

Resource Management Tools

- **AWS Management Console**
- AWS Console Mobile App (View resources)
- **AWS Command line interface**
- AWS Toolkit for PowerShell
- AWS-Shell

Version Control Systems with GIT

What is Version Control System

As name states Version Control System is the **“Management of changes to anything”.**

Version Control is way of storing files in central location accessible to all team members and enabling them to keep track of changes being done in the source code by whom, when & why. It also help teams to recover from some inevitable circumstances.

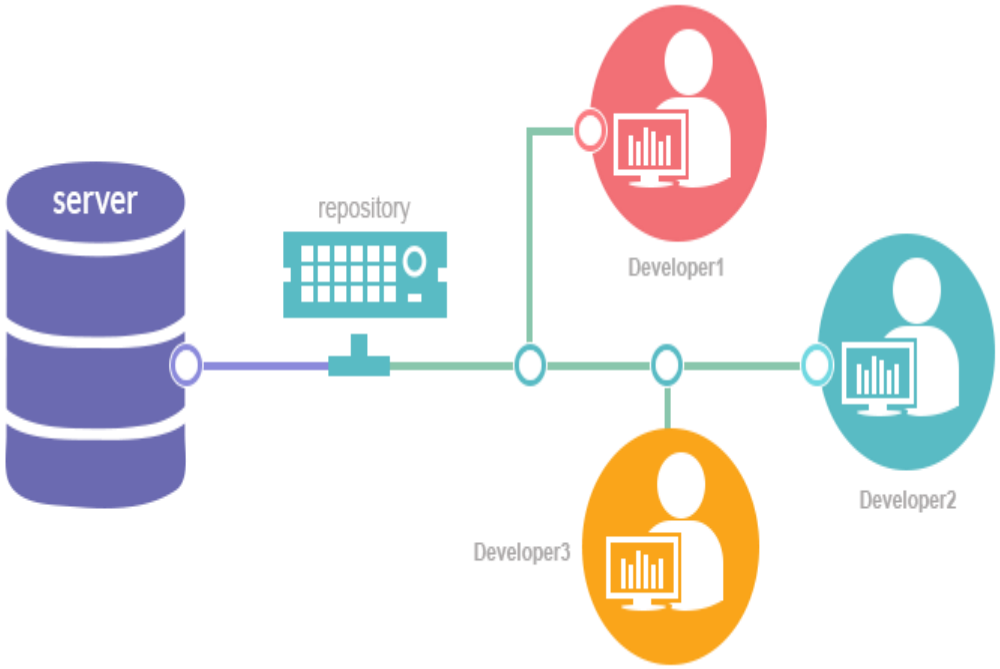
Think about traditional versioning of file with names – Login001.java, Login002.java, Login_final.java.

Its not just for code, it also helps in

- Backups & Restoration
- Synchronization
- Reverts
- Track Changes
- Most importantly in Parallel Development

What is The Need of Version Control System

How Version Control System Will Work ?



Basic Terminology of Version Control System

- Working Directory
- Repository
- Commit
- Checkout

Advantage of Version Control System

- With a distributed system, you can work on your copy of the code without having to worry about ongoing work on the same code by others.
- Identify the ownership of changes
- you can **sync your repositories among yourselves**, bypassing the central location.
- **Managing access is easier** in distributed systems.

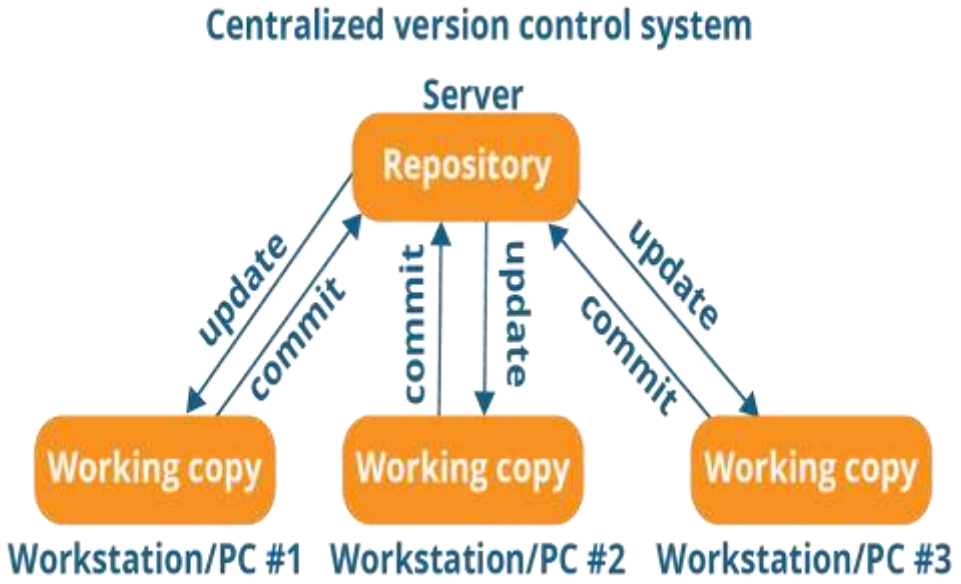
Types of Version Control System

There are two types of version control systems (VCS).

- Centralized version control systems (CVCS)
- Distributed version control systems (DVCS).

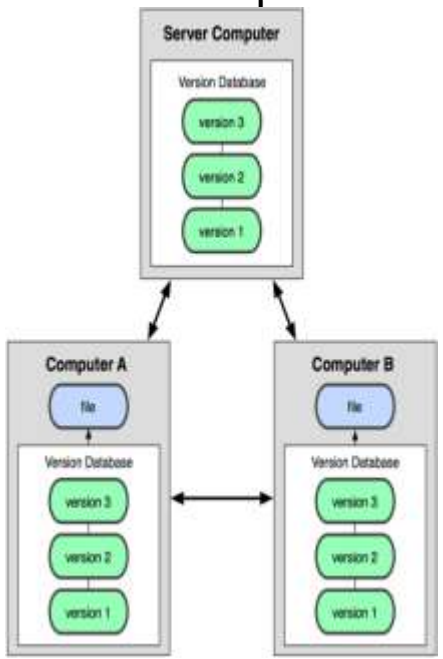
Centralized version control systems (CVCS)

Centralized systems have a copy of the project hosted on a centralized server, to which everyone connects to in order to make changes. Here, the “first come, first served” principle is adopted: if you’re the first to submit a change to a file, your code will be accepted.



Distributed version control systems (DVCS)

In a distributed system, every developer has a copy of the entire project. Developers can make changes to their copy of the project without connecting to any centralized server, and without affecting the copies of other developers. Later, the changes can be synchronized between the various copies.



Introduction of Git

Git History

Linus uses BitKeeper to manage Linux code

Ran into BitKeeper licensing issue

Liked functionality

Looked at CVS as how not to do things

- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control

Why Git?

- **Branching:** gives developers a great flexibility to work on a replica of master branch.
- **Distributed Architecture:** The main advantage of DVCS is **“no requirement of network connections to central repository”** while development of a product.
- **Open-Source:** Free to use.
- **Integration with CI:** Gives faster product life cycle with even faster minor changes.

What is Git

- Git is a distributed version control system that is used by developers to manage changes to a codebase.
- Git uses a branching model that allows developers to work on different features and changes to the codebase without affecting the main branch.
- Git is fast, scalable, and efficient, making it an essential tool for software development.
- Git is used by millions of developers and companies worldwide, and it can be used with a wide range of programming languages and operating systems.
- Git provides a range of tools and commands for managing changes to the codebase, including committing changes, branching, merging, and resolving conflicts.

Git Installation

Open a terminal: You can use the shortcut Ctrl + Alt + T on most distributions.

Update package lists: Run the following command to update the package lists and ensure you are installing the latest version of Git:

- Update The system

  sudo apt update –y / sudo yum update -y

- Install Git: Run the following command to install Git on your system:

  sudo apt install git / sudo yum install git -y

- Verify the installation: After the installation is complete, you can verify it by checking the Git version:

  git --version

Git Commands

- git –version // to check the version

// set the global user name and email

- [root@techlanders ~]# git config --global user.email “sandeep@techlanders.com”
- [root@techlanders ~]# git config --global user.name “sandeep”
- [root@techlanders ~]# git config --global -l
- user.name=sandeep
- user.email= sandeep@techlanders.com

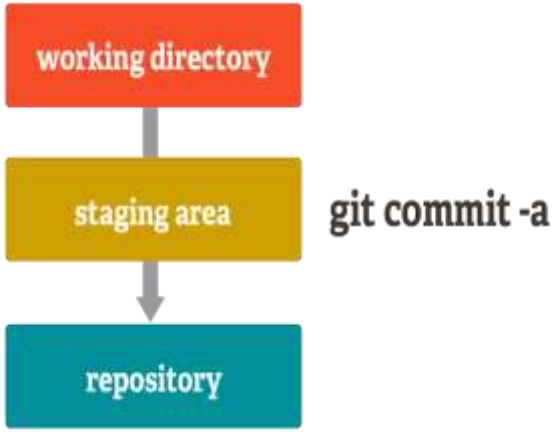
****
****

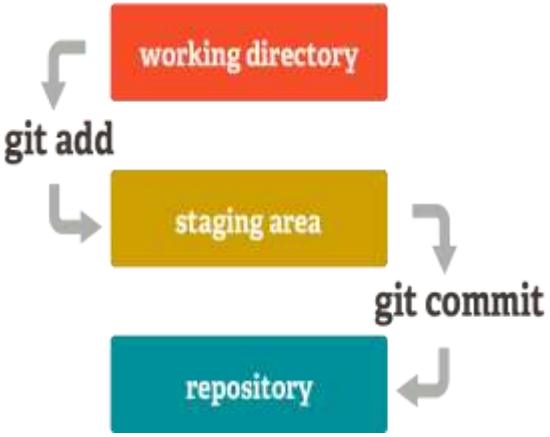
//Initializing a repo

- [root@master git]# mkdir /Repo1
- [root@master git]# cd /Repo1/
- [root@master Repo1]# git init
- Initialized empty Git repository in /Repo1/.git/
- [root@master Repo1]#

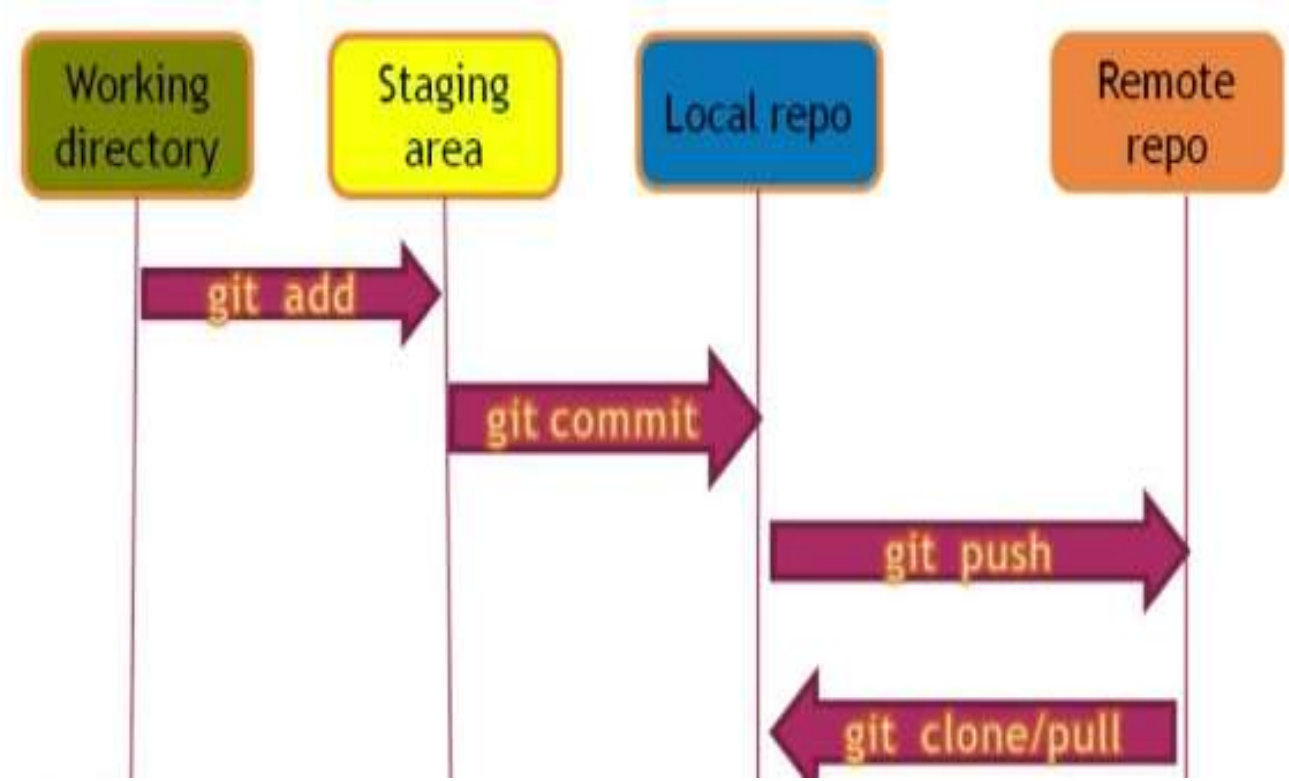
Staging Area

- Unlike the other systems, Git has something called the "staging area" or "index". This is an intermediate area where commits can be formatted and reviewed before completing the commit.

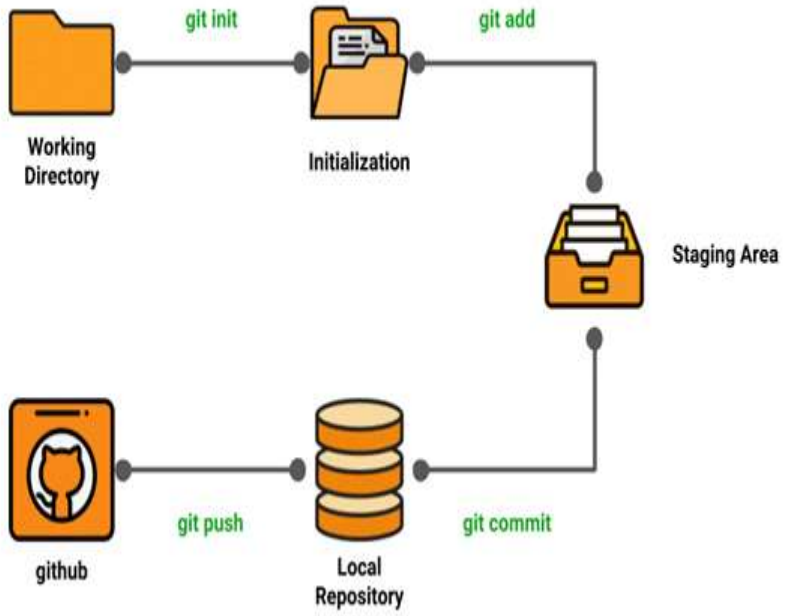




GIT Architecture



GIT Life Cycle



GIT VERSION

```
[root@user20-master plays]# git --version
git version 1.8.3.1
[root@user20-master plays]#
```

GIT HELP

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git --help
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

Setting Identity in GIT

```
[root@Techlanders ~]# git config --global  user.email
“Gagandeep.singh@Techlanders.com”
[root@Techlanders ~]# git config --global  user.name “Gagandeep Singh”
[root@Techlanders ~]# git config --global -l
user.name=Gagandeep Singh
user.email=Gagandeep.singh@Techlanders.com
[root@Techlanders ~]#
```

GIT Repository

- A **repository** is usually used to organize a single project.
- Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs.
- Repository is like a unique shared file system for a project.

Initializing a Repository

```
[root@master git]# mkdir /Repo1
[root@master git]# cd /Repo1/
[root@master Repo1]# git init
Initialized empty Git repository in /Repo1/.git/
[root@master Repo1]# ls -lrt /Repo1/.git/
total 12
drwxr-xr-x. 4 root root  31 Dec 19 19:32 refs
drwxr-xr-x. 2 root root  21 Dec 19 19:32 info
drwxr-xr-x. 2 root root 242 Dec 19 19:32 hooks
-rw-r--r--. 1 root root  73 Dec 19 19:32 description
drwxr-xr-x. 2 root root   6 Dec 19 19:32 branches
drwxr-xr-x. 4 root root  30 Dec 19 19:32 objects
-rw-r--r--. 1 root root  23 Dec 19 19:32 HEAD
-rw-r--r--. 1 root root  92 Dec 19 19:32 config
[root@master Repo1]#
```

Adding file to a Repository

```
[root@master Repo1]# git status
# On branch master
# Initial commit - nothing to commit (create/copy files
and use "git add" to track)
[root@master Repo1]# echo "File1 content" >> file1
[root@master Repo1]# ll
-rw-r--r--. 1 root root 14 Dec 19 19:35 file1
[root@master Repo1]# git add file1
[root@master Repo1]# git status
# On branch master
# Initial commit
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#       new file:   file1
#
[root@master Repo1]#
```

Checking Repository Status

```
[root@user20-master Repo1]# touch file2
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1
#
# Untracked files:
#   (use "git add <file>..." to include in what will be
committed)
#
#       file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@user20-master Repo1]# git add --all
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1
#       new file:   file2
#
[root@user20-master Repo1]# git commit -m "Commit one -
Added File1 & File2"
[master (root-commit) 9c301cb] Commit one - Added File1 &
File2
 2 files changed, 1 insertion(+)
 create mode 100644 file1
 create mode 100644 file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@master Repo1]#echo "File2 content added" > file2
[root@master Repo1]#git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be
committed)
#   (use "git checkout -- <file>..." to discard changes
in working directory)
#
#       modified:   file2
#
no changes added to commit (use "git add" and/or "git
commit -a")
[root@master Repo1]#git commit -am "second commit -
Changes done in File2"
[master 77de849] second commit - Changes done in File2
 1 file changed, 1 insertion(+)
[root@master Repo1]#
```

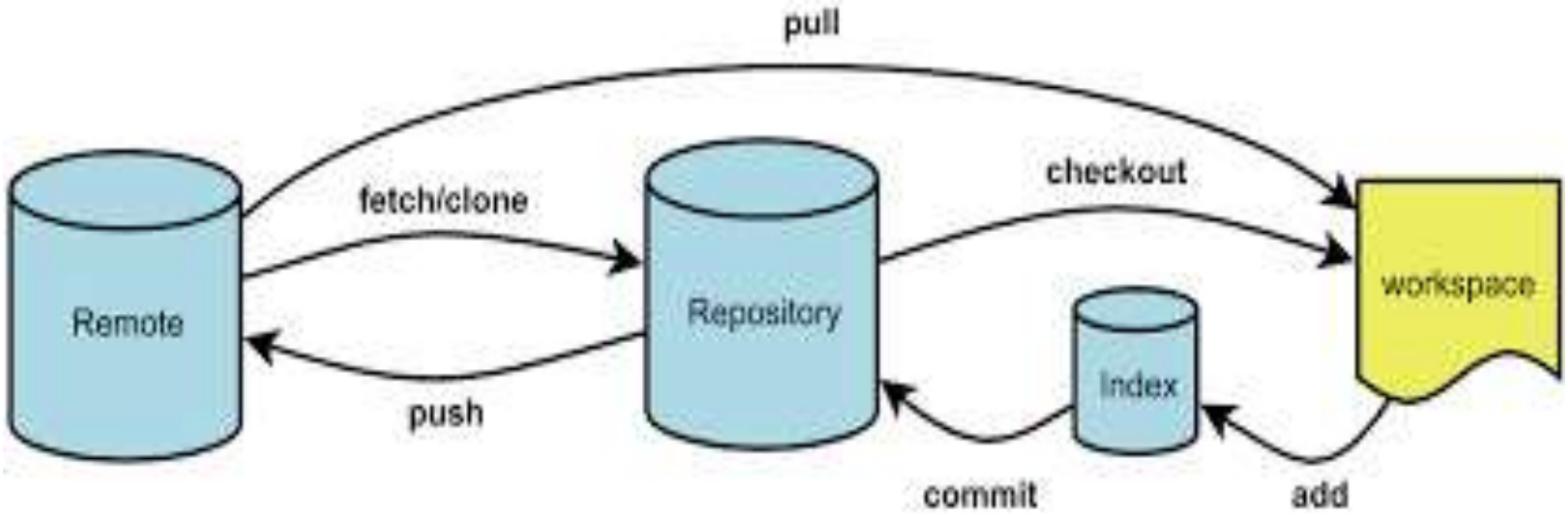
Git Log

```
[root@master Repo1]#git log
commit 77de8496c39c8d442d8e1212f9f3879a33253a1c
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>
Date:   Wed Dec 19 19:48:56 2018 +0000

    second commit - Changes done in File2

commit 9c301cb93733f666e959a87c7a3f61142d1d9f48
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>
Date:   Wed Dec 19 19:43:25 2018 +0000

    Commit one - Added File1 & File2
[root@master Repo1]#
```

Git Diff- Comparing two commits

```
[root@master Repo1]#git diff
9c301cb93733f666e959a87c7a3f61142d1d9f48
77de8496c39c8d442d8e1212f9f3879a33253a1c
diff --git a/file2 b/file2
index e69de29..de51b99 100644
--- a/file2
+++ b/file2
@@ -0,0 +1 @@
+File2 content added
[root@master Repo1]#
```

Git Flow



Initializing a Remote Repository

```
[root@master Repo1]#git remote add origin
https://github.com/admingagan/repo1.git
[root@master Repo1]#
[root@master Repo1]#git pull origin master
From https://github.com/admingagan/repo1
 * branch            master     -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 README.md |  1 +
 abc       |  0
 ntp.yaml  | 13 +++++++++++++
 3 files changed, 14 insertions(+)
 create mode 100644 README.md
 create mode 100644 abc
 create mode 100644 ntp.yaml
[root@master Repo1]#
```

Git Push

```
[root@master Repo1]#git push origin master
Username for 'https://github.com': admin.gagan@gmail.com
Password for 'https://admin.gagan@gmail.com@github.com':
Counting objects: 14, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.22 KiB | 0 bytes/s,
done.
Total 12 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1
local object.
To https://github.com/admingagan/ansibleplaybooks.git
   d9b5ae7..64bad4d  master -> master
[root@master Repo1]#
```

Git Pull

```
[root@master Repo1]#ls
abc  file1  file2  ntp.yaml  readme5 README.md

[root@master Repo1]#git pull origin dev
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-
reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/admingagan/ansibleplaybooks
 * branch            dev        -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 readme6 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 readme6
[root@master Repo1]#ls
abc  file1  file2  ntp.yaml  readme5  readme6  README.md
[root@master Repo1]#
```

Git Clone

```
[root@user20-master git]# git clone
https://github.com/admingagan/test.git
Cloning into 'test'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 23 (delta 5), reused 0 (delta 0), pack-
reused 0
Unpacking objects: 100% (23/23), done.
[root@user20-master git]# cd test
[root@user20-master test]# ll
total 16
-rw-r--r--. 1 root root 10 Dec 20 07:45 Readme
-rw-r--r--. 1 root root 24 Dec 20 07:45 Readme2
-rw-r--r--. 1 root root 57 Dec 20 07:45 readme3
-rw-r--r--. 1 root root  9 Dec 20 07:45 README4
[root@user20-master test]#
```

GIT BRANCH

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch training

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
  training
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch -d training
Deleted branch training (was 74e76df).

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

GIT CHECKOUT

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git checkout training
Switched to branch 'training'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training)
$ git checkout -b training_checkout
Switched to a new branch 'training_checkout'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git branch
  master
  training
* training_checkout
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git log
commit 76e137f900eb33b7eb4a4ac7c81f160af8124697 (HEAD -> training_checkout)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Fri Jun 9 20:38:59 2017 +0530

    commit in branch

commit 74e76dfcaf297ae9b63f950988907cdce8d275de (training, master)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:16:16 2017 +0530

    second commit

commit 3eaadebb9972b29b7463822e99b485b9d9b490eb
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:14:06 2017 +0530

    my initial commit
```

GIT MERGE

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git checkout master
Switched to branch 'master'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training_checkout
Updating 74e76df..76e137f
Fast-forward
 file.txt | 2 ++
 1 file changed, 2 insertions(+)
```

- Make some changes in file.txt available in master and training
- Run git checkout master
- Run git merge training

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ ls -ltr
total 2
-rw-r--r-- 1 kmayer 197121  30 Jun  9 20:51 file-in-training-branch.txt
-rw-r--r-- 1 kmayer 197121 238 Jun 12 20:47 file.txt
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   file.txt
```

GIT MERGE – RESOLVING CONFLICTS

```
1   Adding dummy data - Kulbhushan Mayer
2   Second Update
3
4   <<<<<<< HEAD
5   Making changes in branch for demo
6
7   Making one more change - 09/06/2017 08:44
8
9   Making change at 08:46 PM
10  =======
11  Making change to check merging - 12/06/2017
12  >>>>>>> training
13
```

- <<<<<<< depicts changes from the HEAD or BASE branch
- ======= divides your changes form the other branch
- >>>>>>> depicts the end of changes
- Remove <<<<<<<, =======, >>>>>>> from the file and make then necessary changes
- Now you have to commit the changes explictly

```
1   Adding dummy data - Kulbhushan Mayer
2   Second Update
3
4
5   Making changes in branch for demo
6
7   Making one more change - 09/06/2017 08:44
8
9   Making change at 08:46 PM
10
11  Making change to check merging - 12/06/2017
12
```

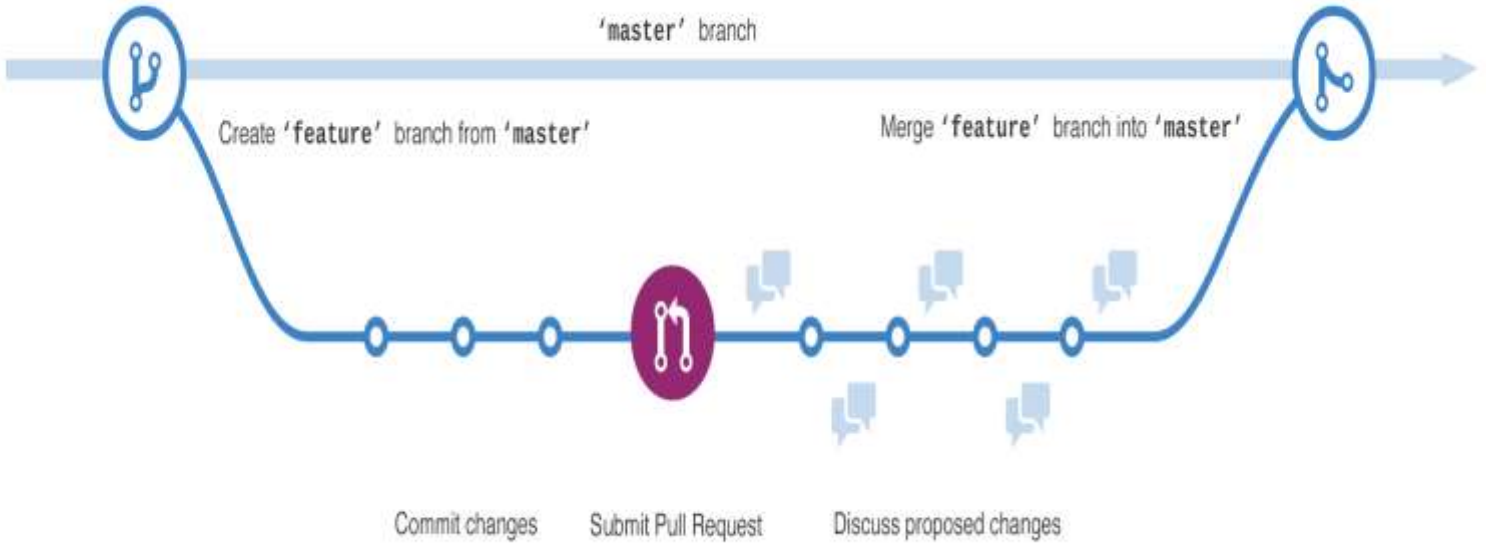
GIT Branch

- Branching is the way to work on different versions of a repository at one time.
- By default your repository has one branch named master which is considered to be the definitive branch.
- We use branches to experiment and make edits before committing them to master.
- When you create a branch off the master branch, you're making a copy, or snapshot, of master as it was at that point in time.
- If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.

GIT Branch

Here you have:

- The master branch
- A new branch called feature (because we'll be doing 'feature work' on this branch)
- The journey that feature takes before it's merged into master



Github/Bitbucket

What is Bitbucket /GitHub/Gitlab

- Bitbucket is a Git solution for professional teams. In simple layman language its a UI for Git, offered by Atlassian, similarly we have different available UI solutions from Github (most famous) and Gitlab.

- GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

- **Host in the cloud**: free for small teams (till 5 users) and paid for larger teams.

- **Host on Your server**: One-Time pay for most solutions.

- Visit "https://bitbucket.org/" and click "Get Started" to sign up for free account.
- Visit "https://github.com/" for Github details

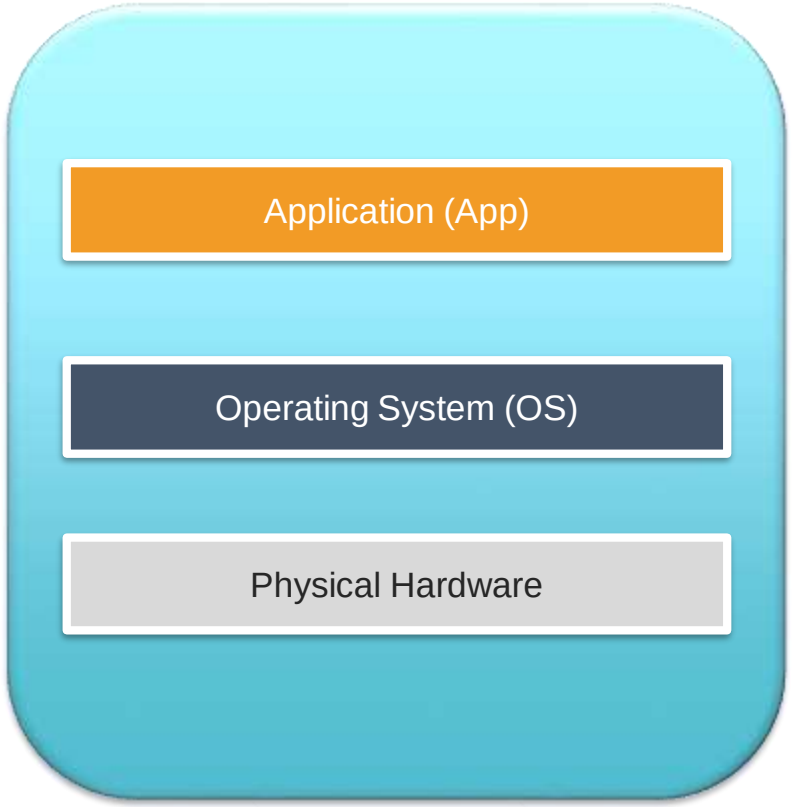
A History Lesson

- In the traditional ages of development and deployment of application

Unused Resources

Difficult Migrations

Vendor dependency



Slow Deployment time

Difficult to Scale

Huge cost in Infrastructure

Traditional Approach

Virtualization

OverHead

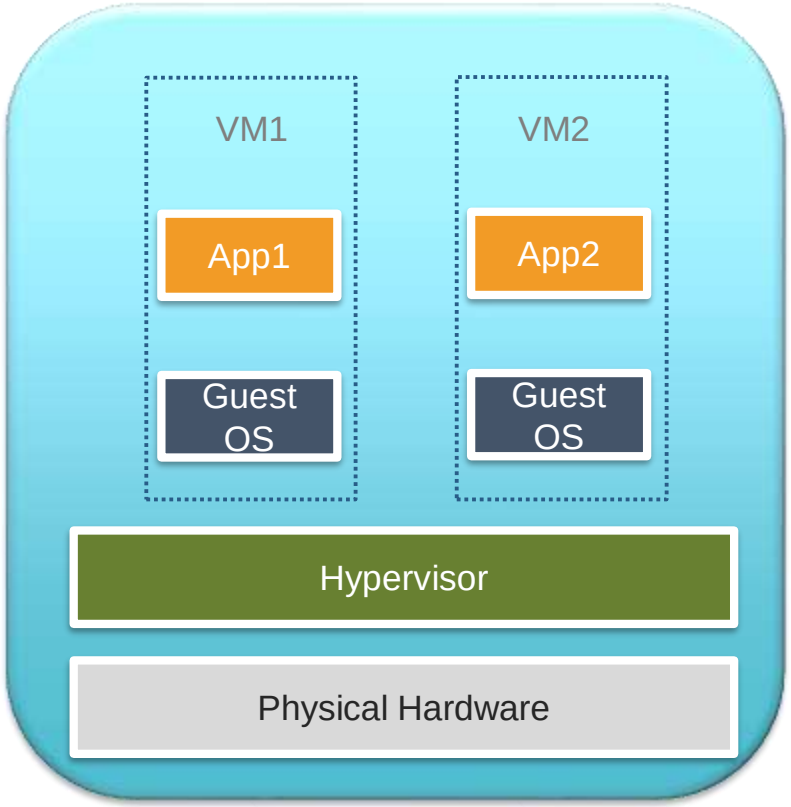
Portability issues

Boot-time still in Minutes

Scaling issue in Hybrid Env

Migrations still failing

Costly Solution



Server Virtualization

Better Resource Pooling

Easier to Scale

Flexibility & Easy Migration

Faster Deployments

Faster Boot time

Containerization

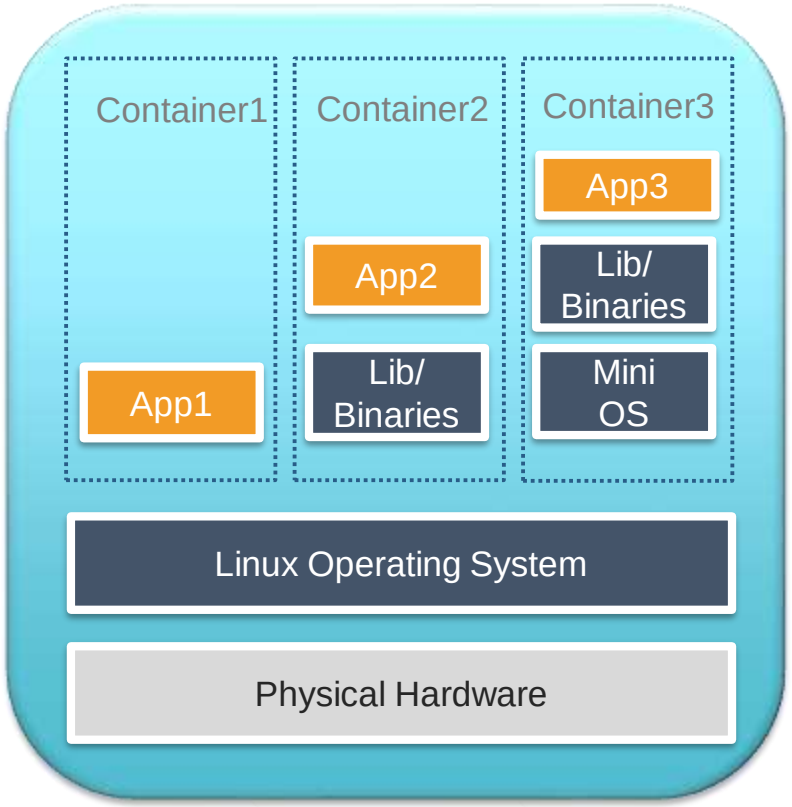
Less OverHead

Highly Portable

Scaling in Hybrid Env

High Migrations success ratio

Cost Effective Solution



Containerization

Much Better Resource Pooling

Extended Scaling

Flexibility & Easy Migration

Faster Deployments (Seconds)

Faster Boot time (Seconds)

Introducing Containers

- Container based virtualization uses the kernel on the host's operating system to run multiple guest instances

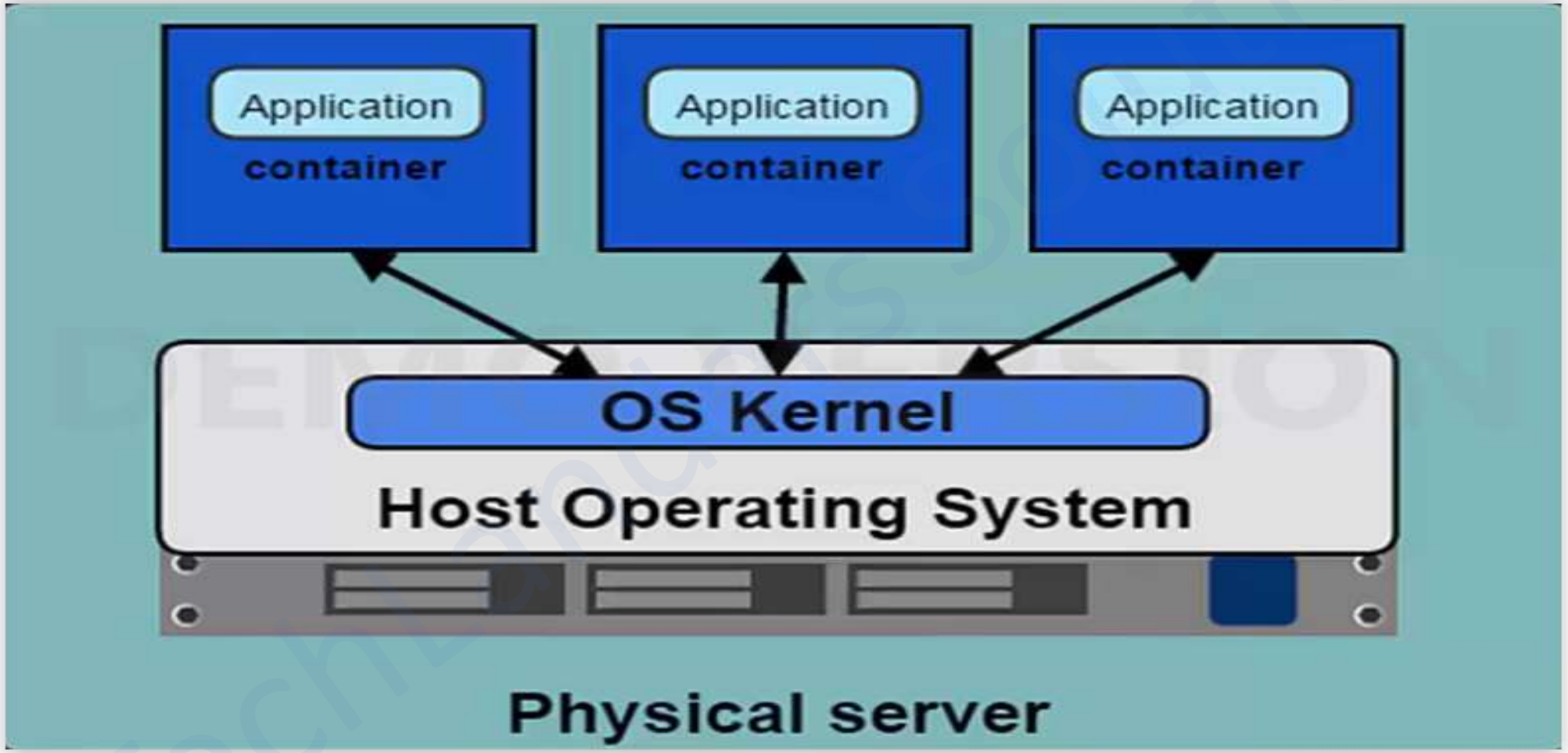
- Each guest instance is called a "Container"

- Each container has its own

- Root Filesystem
- Processes
- Memory
- Devices
- Network Ports

- From outside its looks like a VM but its not a VM

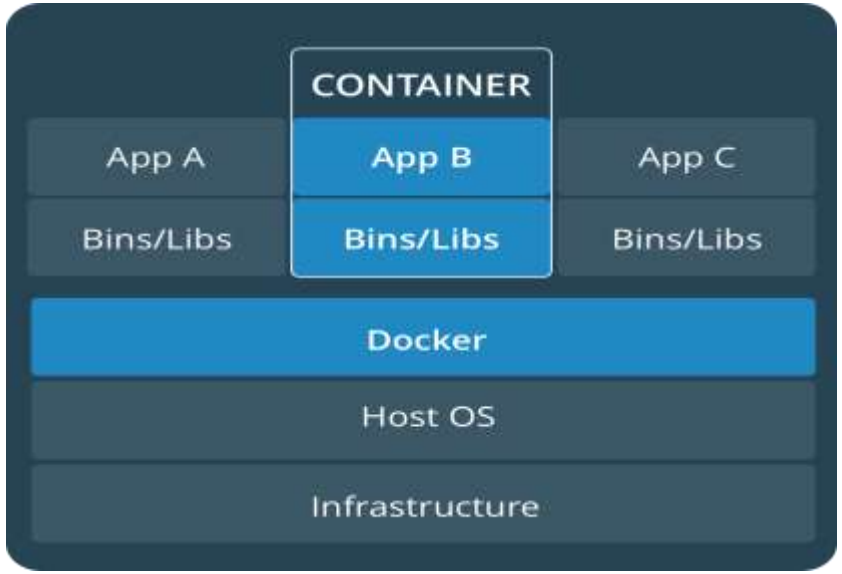
Overview of Containers



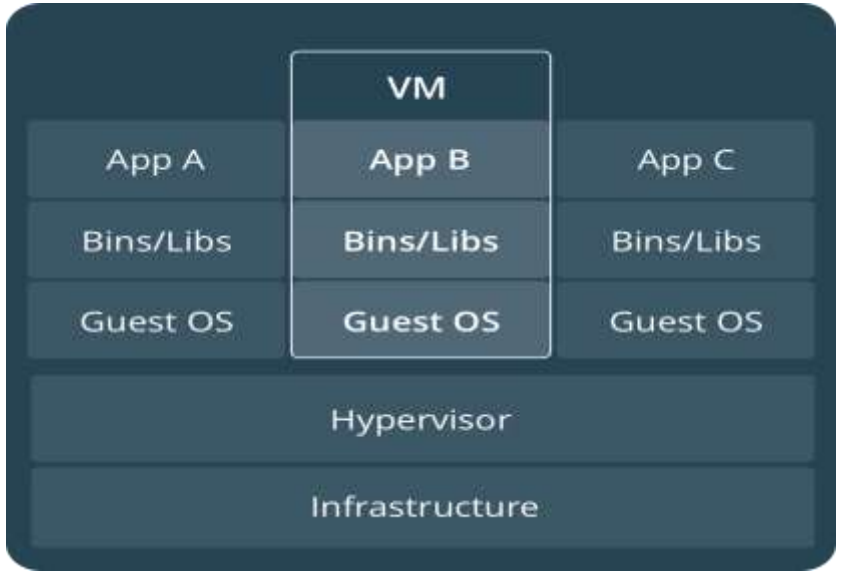
Containers VS VM’s

- Container are more light weight
- No need to install dedicated guest OS, no virtualization like VM is required
- Stop/Start time is very fast
- Less CPU, RAM, Storage Space required
- More containers per machine than VM’s
- Great Portability

Containers VS VM's

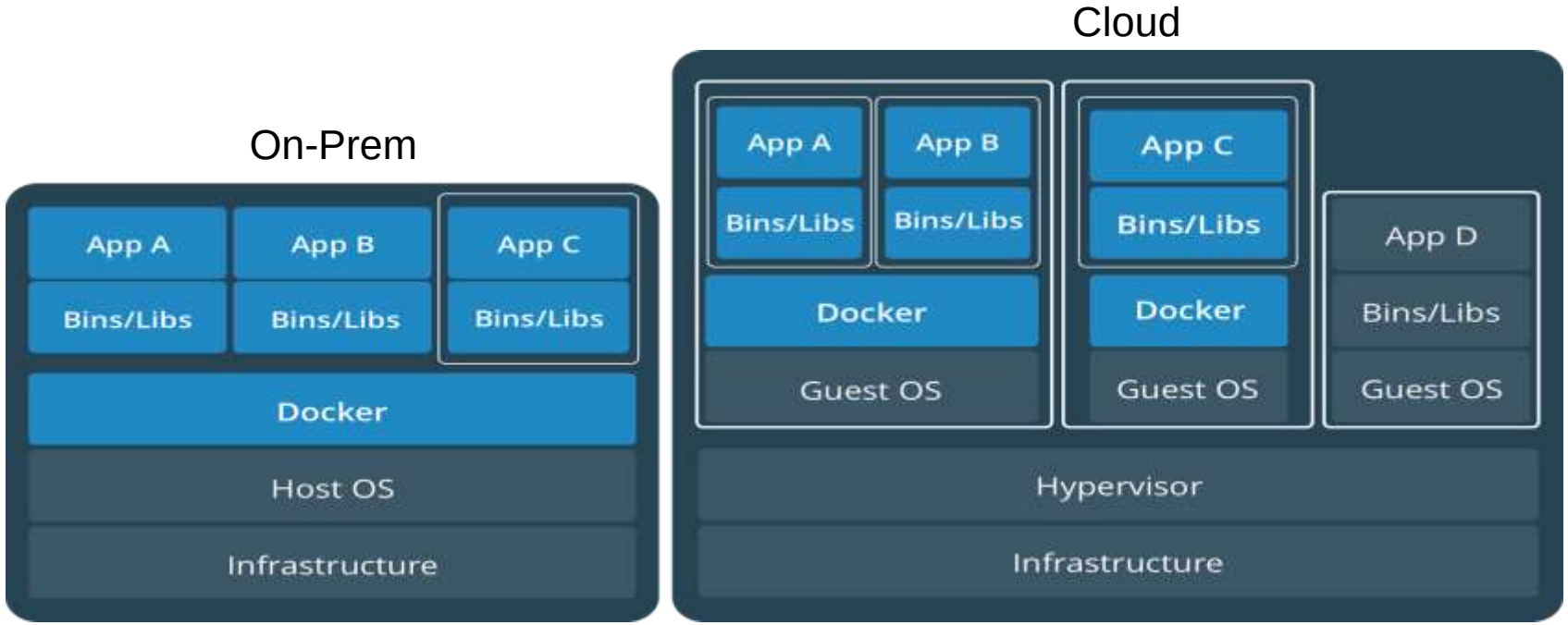


Containers are an app level construct



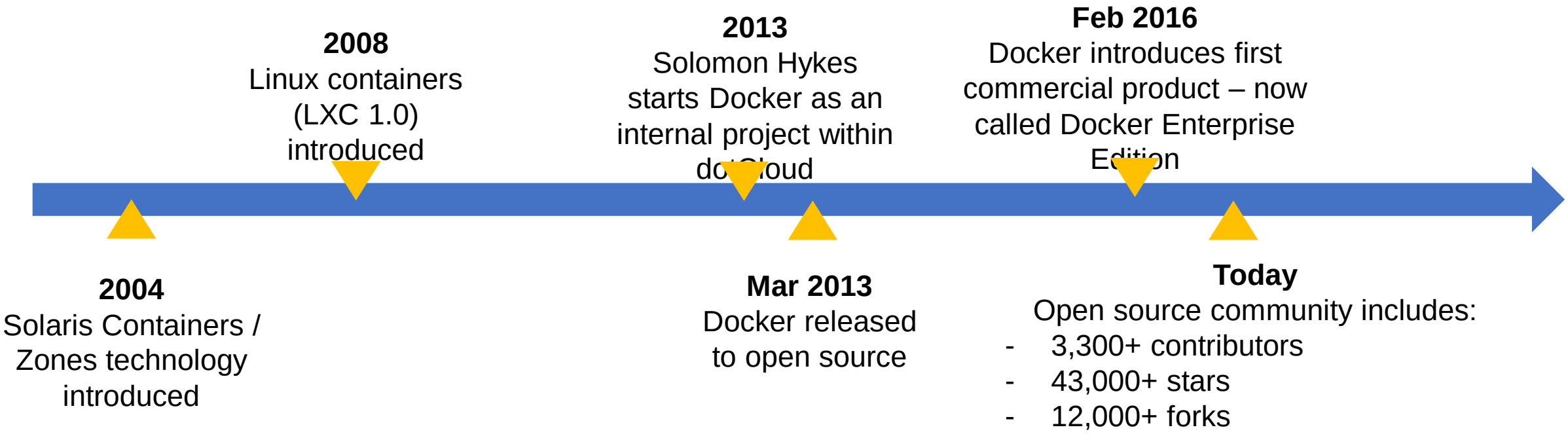
VMs are an infrastructure level construct to turn one machine into many servers

Containers and VM’s together



Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

Origins of Docker Project



Origins of Docker Project

- ‘dotCloud’ was operating a PaaS platform, using a custom container engine.
- This engine was based on ‘OpenVZ’ (and later, LXC) and AUFS.
- It started (circa 2008) as a single Python script.
- By 2012, the engine had multiple (~10) Python components. (and ~100 other micro-services!)
- End of 2012, ‘dotCloud’ refractors this container engine.
- The codename for this project is "Docker."

Docker becomes a platform

- The initial container engine is now known as "Docker Engine“.

- Other tools are added:

    - Docker Compose (formerly "Fig") (YAML file)
    - Docker Machine (SDK)
    - Docker Swarm (Cluster)
    - Docker Cloud (formerly "Tutum”)
    - Docker datacenter (Management Layer)

- Docker Inc. launches commercial offers.

About Docker Inc.

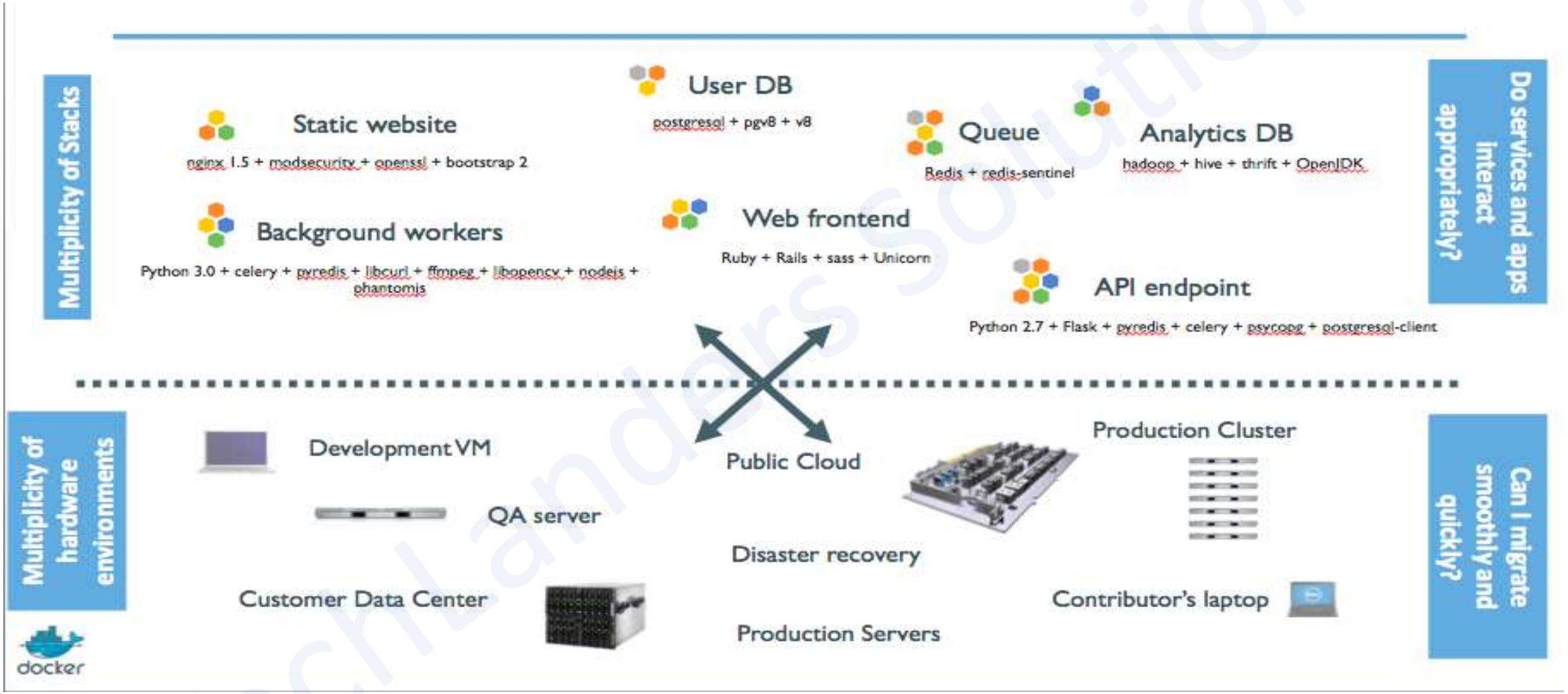
- Docker Inc. Formerly ‘dotCloud’ Inc, used to be a French company

- Docker Inc. is the primary sponsor and contributor to the Docker Project:
  - Hires maintainers and contributors.
  - Provides infrastructure for the project.
  - Runs the Docker Hub.

- HQ in San Francisco.

- Backed by more than 100M in venture capital.

Deployment Problem



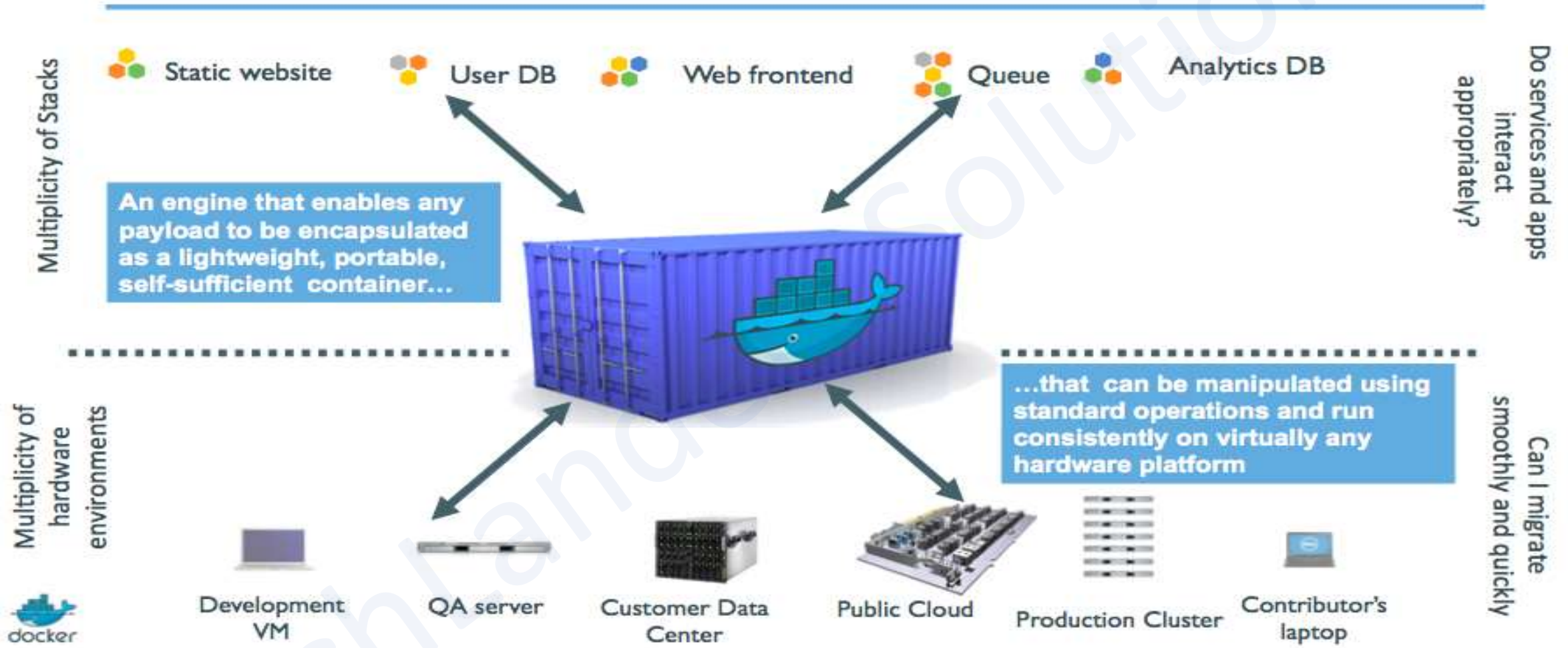
Matrix Checks

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

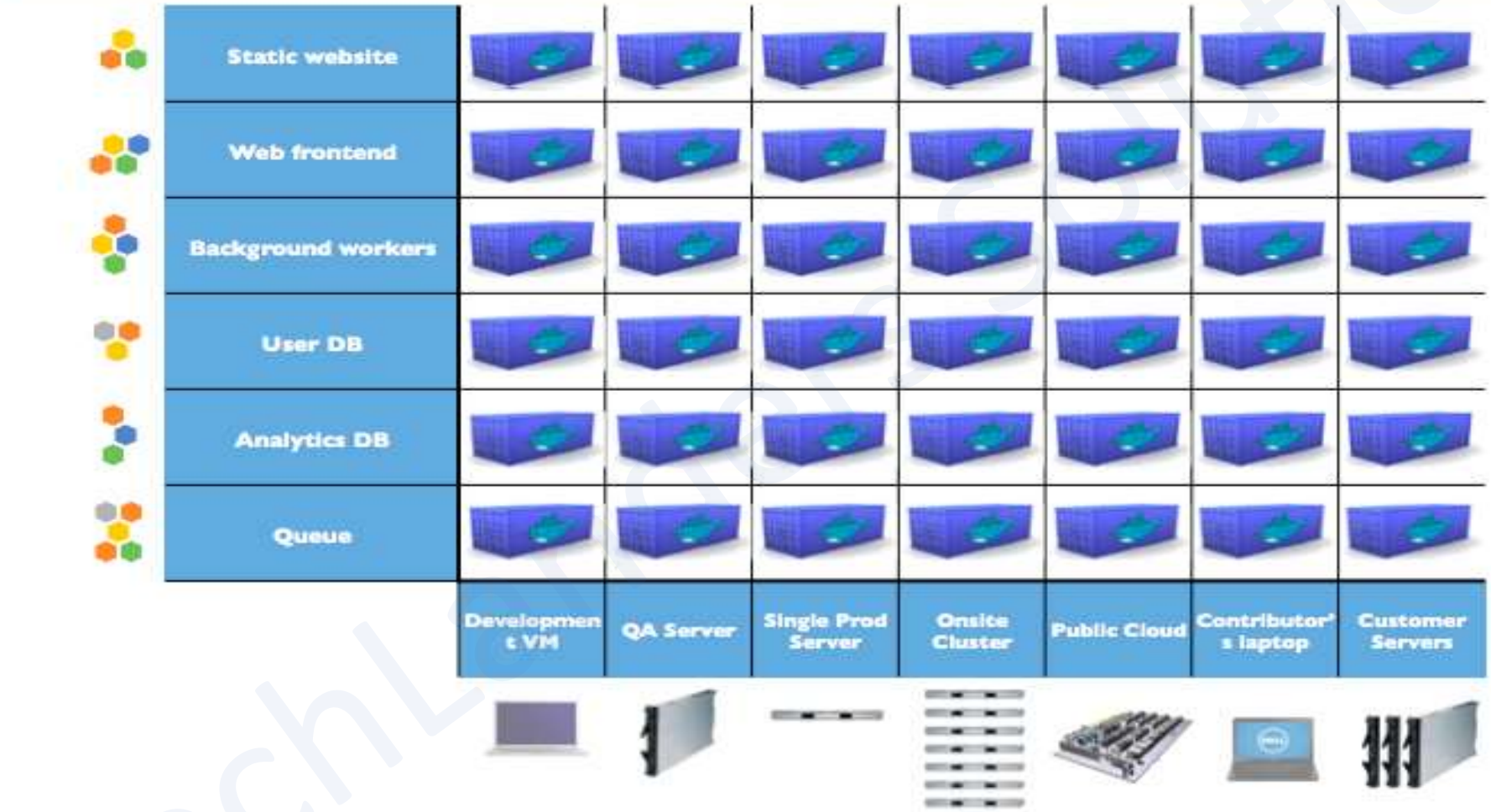
docker

Intermodal Shipping Containers

Shipping Container for Applications



Eliminate the Matrix



Results

Speed

- No OS to boot = applications online in seconds

Portability

- Less dependencies between process layers = ability to move between infrastructure

Efficiency

- Less OS overhead
- Improved VM density

Adoption in Just 4 years

Why Docker?


300% +
Faster Time to Market
1300% +
Developer Productivity
60% +
Deployment Velocity
40% -
IT Infrastructure Reduction
40% +
IT Operational Efficiency
72% +
Faster Issue Resolution

Session: 2

Docker Components

Docker Overview

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- By using Docker’s methodologies for shipping, testing, and deploying, you can reduce time of customer delivery.

Docker Platform

- Docker provides the ability to package and run an application in a loosely isolated environment called a container. The **isolation** and **security** allow you to run many containers simultaneously on a given host.

- Because of the **lightweight** nature of containers, which run without the extra load of a hypervisor, you can run more containers on a given hardware combination than if you were using virtual machines.

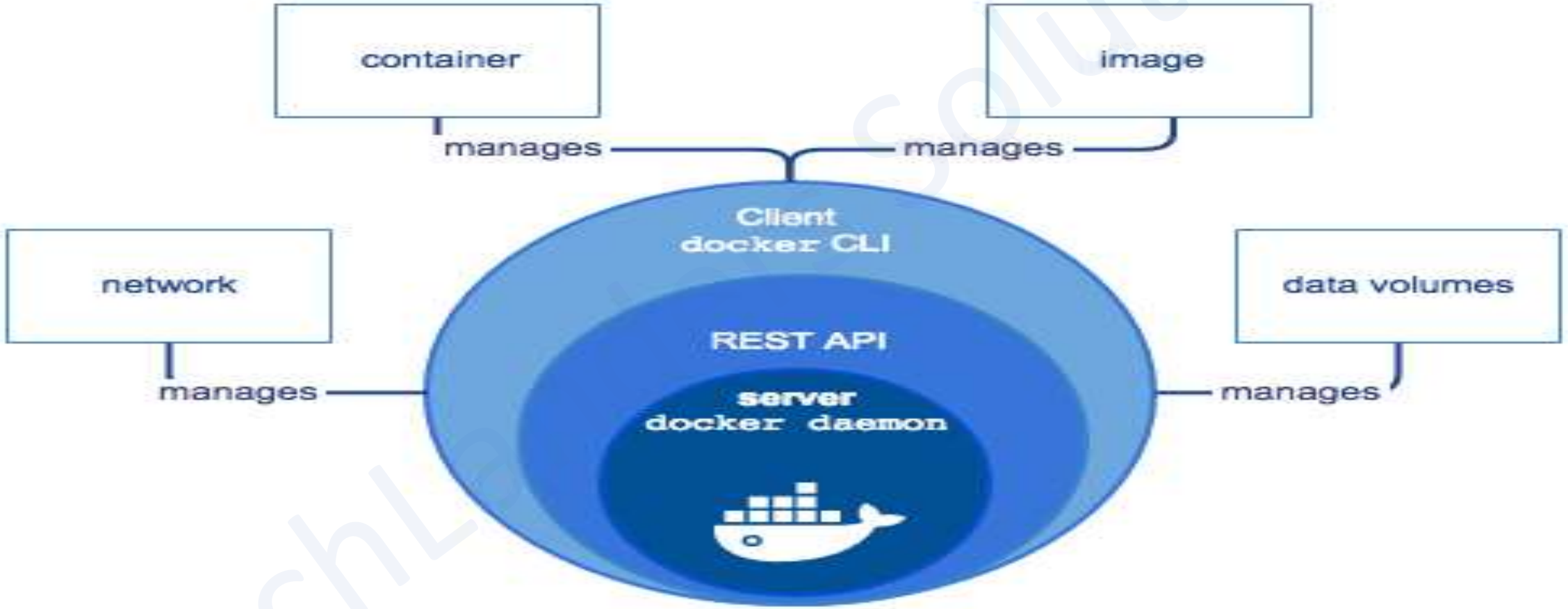
Docker Test

- docker --version
- Test the docker functioning: docker run hello-world

Docker Platform

- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Encapsulate your applications (and supporting components) into Docker containers
  - Distribute and ship those containers to your teams for further development and testing
  - Deploy those applications to your production environment, whether it is in a local data center or the Cloud

Docker Engine



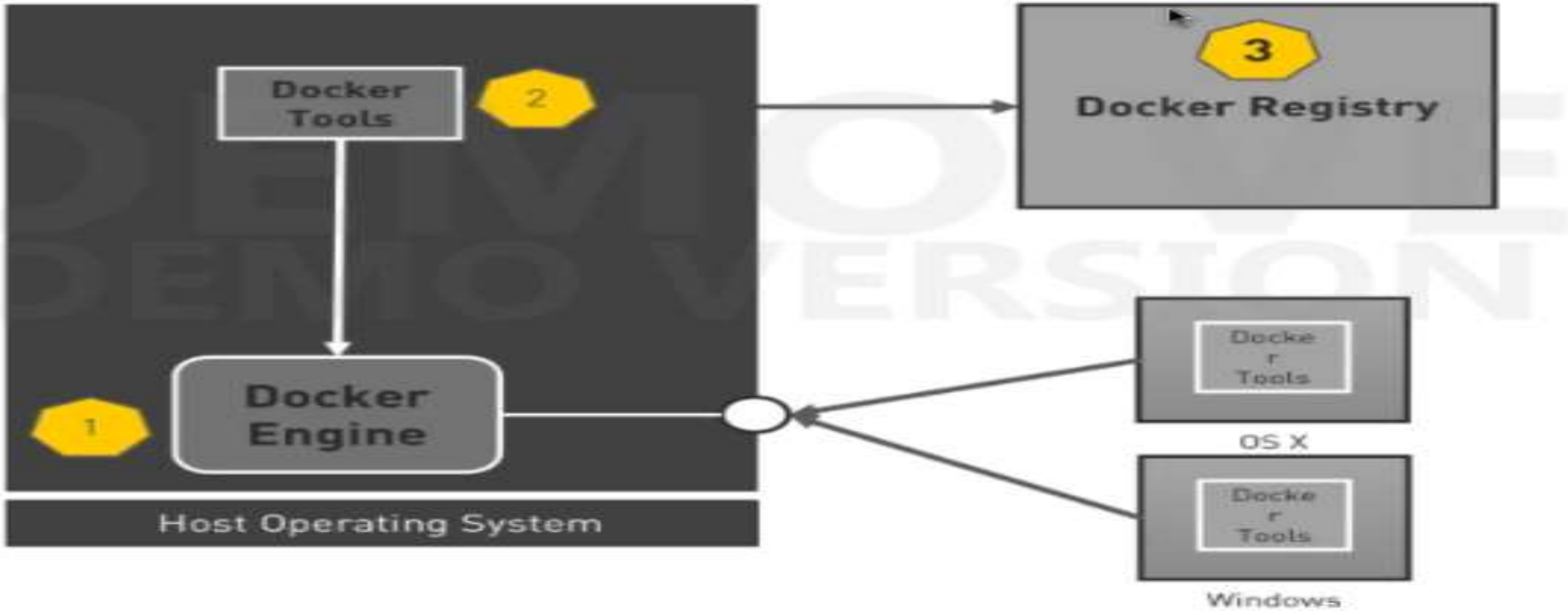
Docker Engine

- Docker Engine is a client-server application with these major components:
  - A server which is a type of long-running program called a daemon process.
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  - A command line interface (CLI) client.

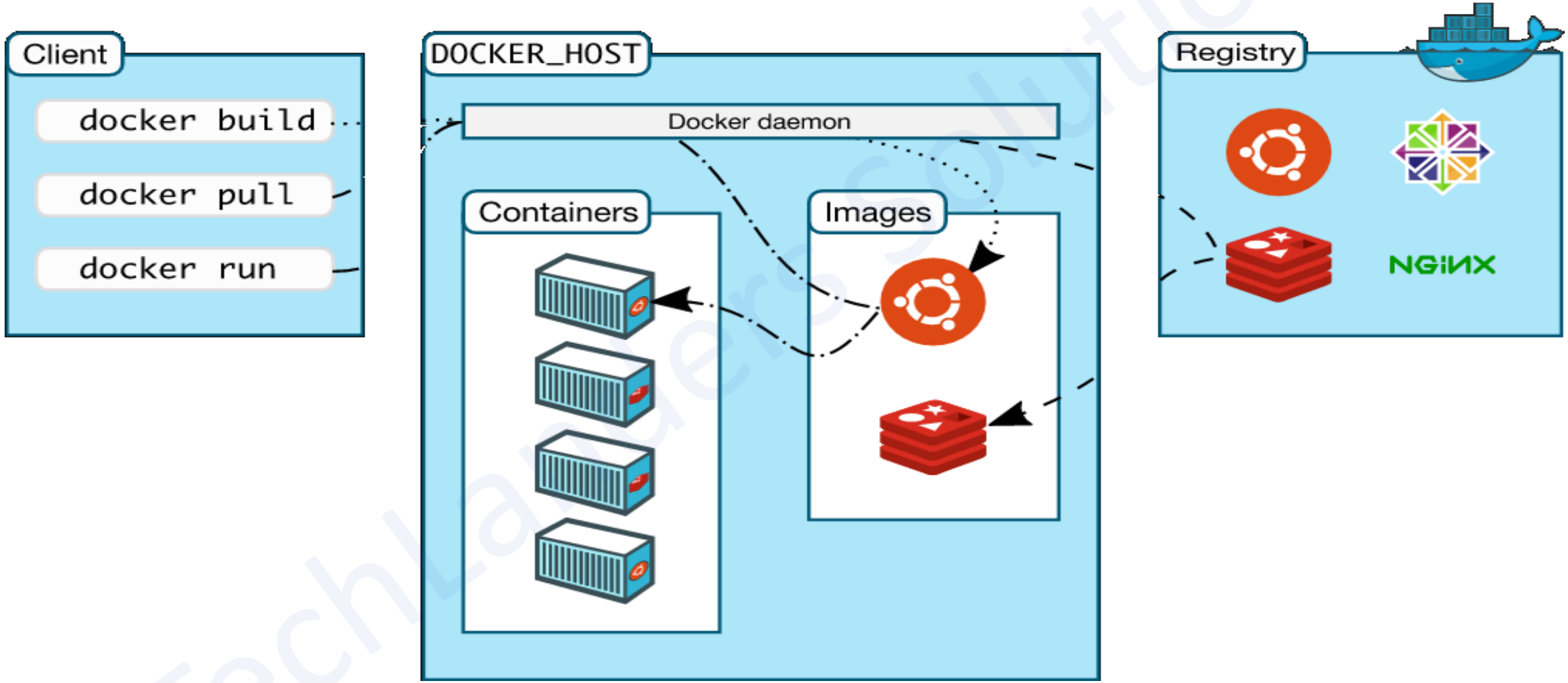
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands

- The daemon creates and manages Docker objects, such as images, containers, networks, and data volumes

Docker Architecture



Docker Architecture



Docker Architecture

- Docker uses a client-server architecture.
- The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.
- The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

Docker Architecture

- The Docker Daemon
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- The Docker Client
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker.
  - It accepts commands and configuration flags from the user and communicates with a Docker daemon.

Docker Architecture

Image

The basis of a Docker container. The content at rest.

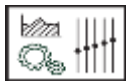
Container

The image when it is ‘running.’ The standard unit for app service

Engine

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

Registry



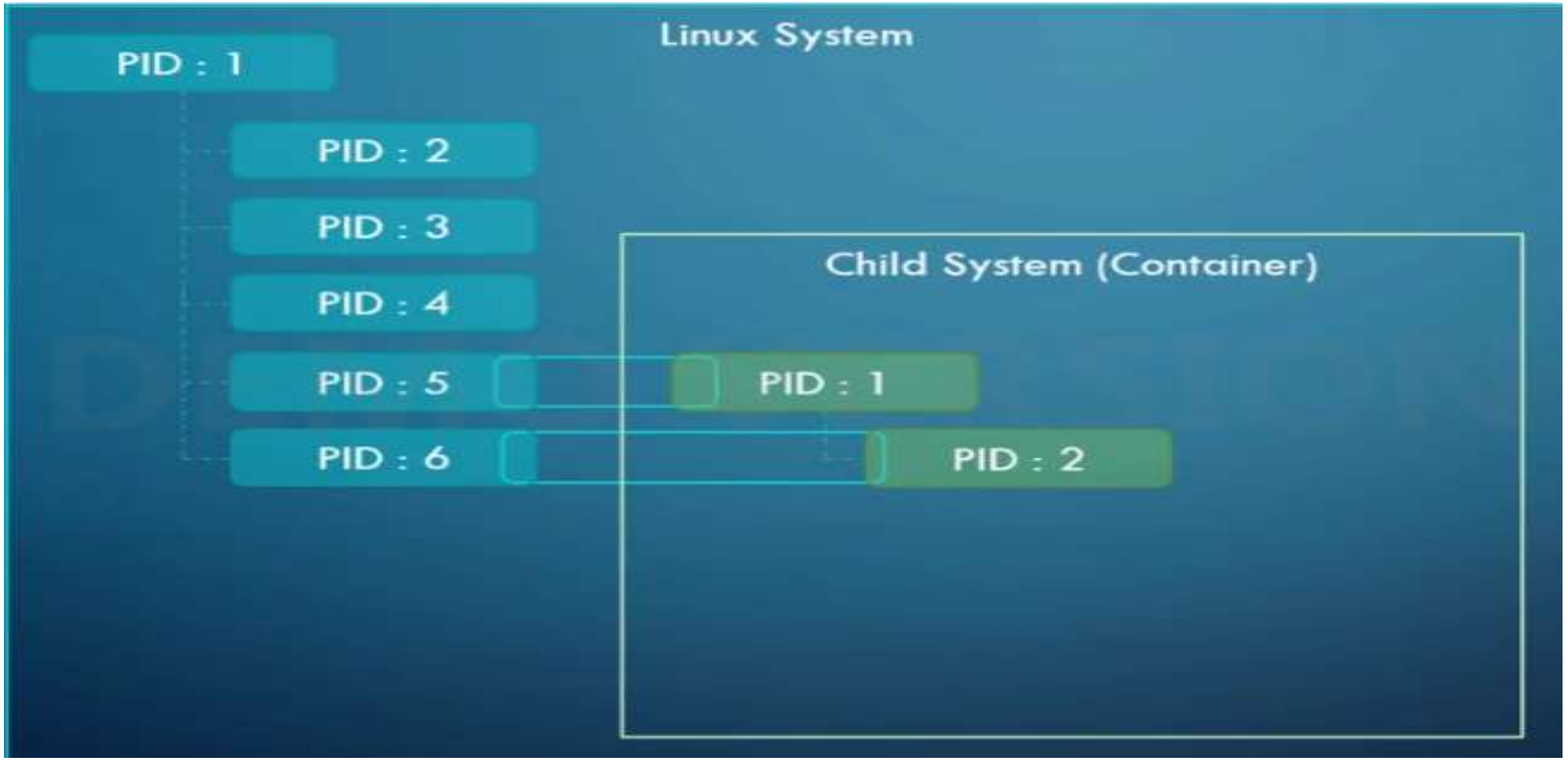
Stores, distributes and manages Docker images

Docker Architecture

- Lets understand how does applications works in isolation under the hood.
- Docker uses namespace which is a nothing but an isolated environment like VM, but on top of VM called container.

Docker Architecture

- The demonstration will be given once we understand the container operations.



Docker Images

- A Docker image is a read-only template with instructions for creating a Docker container.
- For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others.
- A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.
- Docker images are the build component of Docker.

Docker Containers

- A Docker container is a running instance of a Docker image.
- You can run, start, stop, move, or delete a container using Docker API or CLI commands.
- When you run a container, you can provide configuration metadata such as networking information or environment variables.
- Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases.
- Docker containers are the run component of Docker.

Docker Registries

- A docker registry is a library of images.
- A registry can be public or private, and can be on the same server as the Docker daemon or Docker client, or on a totally separate server.
- Docker registries are the distribution component of Docker.
- “Docker Hub” is known as global registry.

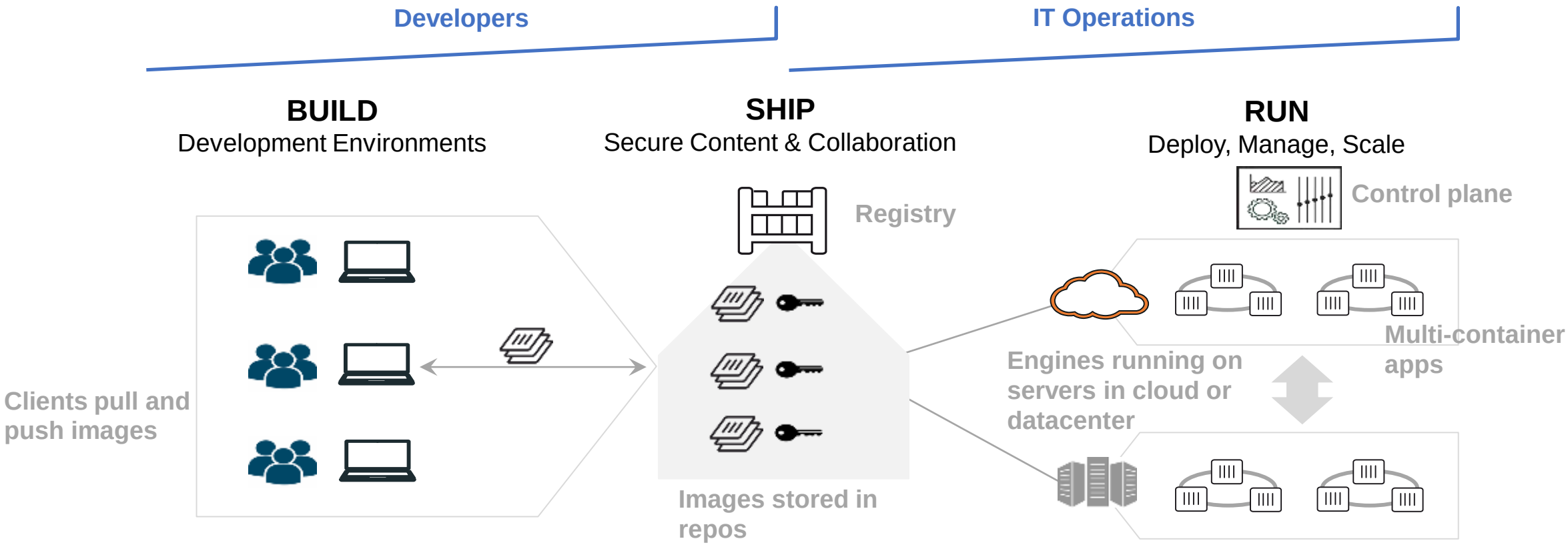
Docker Features

- Lightweight
  - Containers running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.

- Open
  - Docker containers are based on open standards allowing containers to run on all major Linux distributions and Microsoft operating systems with support for every infrastructure.

- Secure
  - Containers isolate applications from each other and the underlying infrastructure while providing an added layer of protection for the application.

Container as a Service



Session: 3

Classroom Environment

Docker Engine Install Demo

- Docker Engine/Client would be installed on Training Environment as demo LAB.
- https://docs.docker.com/engine/installation/linux/centos/

Docker Installation Key Points

- docker.service Systemd File:

[root@TechLanders lib]# more /usr/lib/systemd/system/docker.service

- Docker Socket file:

[root@TechLanders lib]# file /var/run/docker.sock
/var/run/docker.sock: socket

- Docker PID file and Docker Container PID file (This is Docker Daemon which will have pid1)

root@TechLanders run]# cat /var/run/docker.pid
3715
[root@TechLanders libcontainerd]# cat /var/run/docker/containerd/docker-containerd.pid
4251

- Docker Detailed Information:

[root@TechLanders libnetwork]# docker info

Manage Docker as a non-root user

- The docker daemon binds to a Unix socket instead of a TCP port.
- By default that Unix socket is owned by the user ”root” and other users can only access it using sudo.
- The docker daemon always runs as the root user.
- If you don’t want to use sudo when you use the docker command, add users to Unix group called ”docker” example: usermod -aG docker <user-name>
- When the docker daemon starts, it makes the ownership of the Unix socket read/writable by the docker group.

Session: 4

Containers

How Container works

- A container uses the host machine's Linux kernel, and consists of any extra files you add when the image is created, along with metadata associated with the container at creation or when the container is started.
- Each container is built from an image.
- The image defines the container's contents, which process to run when the container is launched, and a variety of other configuration details.
- The Docker image is read-only. When Docker runs a container from an image, it adds a read-write layer on top of the image (using a UnionFS) in which your application runs.

How Container works

- When you use the ”docker run” CLI command, the Docker Engine client instructs the Docker daemon to run a container.

  docker run ubuntu ps ax

- This example tells the Docker daemon to run a container using the centos Docker image, to remain in the foreground in interactive mode (-i), provide a tty terminal (-t) and to run the /bin/bash command.

  docker run -i -t centos /bin/bash

  docker run -i -t centos (bash will be the default shell)

  Because if you exit the current running process /bin/bash (pid 1), container will stop/exit. Use “Ctrl pq” to safe exit without stopping the container.

How Container works

```
[root@TechLanders libcontainerd]# docker run -i -t centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
d9aaf4d82f24: Pull complete
Digest: sha256:eba772bac22c86d7d6e72421b4700c3f894ab6e35475a34014ff8de74c10872e
Status: Downloaded newer image for centos:latest
[root@9a06b1a61fc5 /]#
```

```
[root@TechLanders overlay]# ls -lrt /var/lib/docker/image/overlay2/repositories.json
-rw-------. 1 root root 545 Sep 18 03:17 /var/lib/docker/image/overlay2/repositories.json
```

How Container works

[root@TechLanders overlay]# cat repositories.json

{"Repositories":{"centos":{"centos:latest":"**sha256:196e0ce0c9fbb31da595b893dd39bc9fd4aa78a474bbdc21459a3ebe855b7768**","centos@sha256:**eba772bac22c86d7d6e72421b4700c3f894ab6e35475a34014ff8de74c10872e**":"sha256:196e0ce0c9fbb31da595b893dd39bc9fd4aa78a474bbdc21459a3ebe855b7768"},"hello-world":{"hello-world:latest":"sha256:05a3bd381fc2470695a35f230afefd7bf978b566253199c4ae5cc96fafa29b37","hello-world@sha256:1f19634d26995c320618d94e6f29c09c6589d5df3c063287a00e6de8458f8242":"sha256:05a3bd381fc2470695a35f230afefd7bf978b566253199c4ae5cc96fafa29b37"}}}

[root@TechLanders overlay]# docker image list

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
centos	latest	**196e0ce0c9fb**	3 days ago	197MB
hello-world	latest	05a3bd381fc2	5 days ago	1.84kB

[root@TechLanders overlay]# docker image inspect centos

More Useful - Container

- Do something in our container:

  Lets suppose we try to use “talk” for communication.

- Let’s check how many packages are installed:

  rpm –qa | wc –l

A non interactive - Container

- In your Docker environment, just run the following command:

  docker run jpetazzo/clock

- This container just displays the time every second.
- This container will run forever.
- To stop it, press ^C.
- Docker has automatically downloaded the image jpetazzo/clock.

Run in background - Container

- Containers can be started in the background, with the -d flag (daemon mode):

  docker run -d jpetazzo/clock

- We don't see the output of the container.
- But don't worry: Docker collects that output and logs it!
- docker ps -a
- docker logs <conatiner-id>
- Docker gives us the ID of the container.

List Running Containers

- With docker ps, just like the UNIX ps command, lists running processes.

  docker ps
  docker ps –l
  docker ps –a
  docker ps -q

- The (truncated) ID of our container.
- The image used to start the container.
- That our container has been running (Up) for a couple of minutes.
- Now, start multiple containers and use “docker ps” to list them.

Stop our Container

- There are two ways we can terminate our detached container.
    - Killing it using the docker “kill” command.
    - Stopping it using the docker “stop” command.
- The first one stops the container immediately, by using the KILL signal.
- The second one is more graceful. It sends a TERM signal, and after 10 seconds, if the container has not stopped, it sends KILL.

Removing Container

- Let’s remove our container:

```
docker rm <yourContainerID>
```

Attaching to a Container

- You can attach to a container:

  docker attach <containerID>

- The container must be running.

- There *can be multiple clients attached to the same container.*

Restarting a Container

- When a container has exited, it is in stopped state.
- It can then be restarted with the “start” command.

  docker start <containerID>

- The container must be running.
- You can also use restart command

  docker restart <container-id>

LAB1

- Create one Ubuntu Container in interactive and terminal mode
- Exit out of the container
- Check the status of container
- Restart the container
- Get attached to the container
- Get out of container without exiting ,, stop the container
- Remove the container
- Create the container now in detached mode and follow the same steps ..

LAB2

- Create a new nginx container using (-d) option.
- Docker run –d nginx
- Now check the difference using docker ps –a command
- Try to access this container
- Docker attach <container-ID>

  cat /etc/os-release (THIS WILL NOT WORK) (BECAUSE WE ARE NOT USING INETRACTIVE IN THE CREATION)
- Use docker exec -it <CONATINER-ID> /bin/bash
- Remove everything
- Docker stop
- Docker rm
- Docker run –dt nginx (YOU NEED TO PROVIDE A TERMINAL HERE) → THIS IS TRUE WITH BASE IMAGES
- Docker attach <container-ID>

LAB3

- *On your host machine install httpd package*

  #For centos

  - Yum install httpd –y

  #For ubuntu :

  -- apt-get install apache2

- *Verify installation:*

  rpm –qa | grep –i httpd

  #For ubuntu

  find / -name apache2 or dpkg –list | grep –i apache2

- *Create a container :*
- #For ubuntu :

docker run –it –name c1 ubuntu

docker run –it centos

Learning - Containers have isolated independent root filesystem

LAB3 cont..

Check the package inside it:

dpkg --list | grep -i apache

*Install the package if not there :

apt-get update –y

apt-get install apache2

dpkg --list | grep apache2

#For centos :

rpm –qa | grep –i httpd

- If facing an error while installing httpd in centos container run below :

sudo sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*

sudo sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*

sudo yum update –y

Session: 5

Docker – Containers Advanced

Checking Info

Checking System usage:

$ docker system df // use –v for verbose run

Checking Docker Information:

$ docker system info

Checking Docker events:

$ docker system events //Checking runtime events

Checking Logs:

$ docker logs {containerid}

Cleaning up Unused resources:

$ docker system prune //use –a for deleting all unused images

docker volume prune //to remove all unused volumes

Container

- *Set a Name to container*

  docker run --name my-redis -d redis

- *Setting a hostname from outside:*

  docker run -dit --name container1 --hostname c1 centos

- *Finding stats for container:*

  docker top container-name

  docker stats

- *Inspecting Docker containers*

  docker inspect container-name

- *Homedirectory*

  /var/lib/docker

Resource binding to a Container

Limiting memory(Quota):

$ docker run -dit -m 300M --name c1 redis

Verify :

Docker stats c1

Limiting CPUs:

docker run -dit --cpus 0.02 --name c2 redis

Check the nanocpus

Docker inspect c2 | grep –I cpu

Session: 5

Ports and Volumes

Working with Volumes

- Containers have ephemeral storage
- Data shared within the container is not accessible after it is terminated
- Volumes provide persistence to containers
- Each volume is implicitly or explicitly mapped to a host directory

Working with Volumes

- Question is who is responsible for performing all these operations in Docker.
- And the answer is “Storage Drivers”, some of the common drivers are like “aufs, zfs, overlay, overlay2, device mapper etc.)
- The choice of device driver depends upon the underlying OS.
- “aufs” is the default Storage Driver for Ubuntu, however this is not present in Centos, in such cases “device mapper” is a better option to go with.
- The key part is “docker will pick the best storage driver for you automatically”.
- Execute docker info | more to know about the drivers used by the docket for your host.

Volume mapping

- cd /home/ubuntu/raman/
- touch hostfile

- #On the command-line, with the -v flag for “docker run”.

  docker run -it --name c1 -v /home/ubuntu/raman:/appdata centos

Edit the file hostfile in appdata folder in container

#Exit out the container safely

See the changes getting reflected in local as well.

Docker Ports and Volumes

- Two most important things in docker are Ports and Volumes.
- docker images
- docker run -d nginx:latest
- docker ps
- docker inspect <container-id>| grep -i ip
- docker ps
- docker stop <container-id>
- docker rm `docker ps -a -q`

Docker Ports and Volumes

- Lets expose the containers to a random port by docker itself
- docker run -d --name=nginxserver -P nginx:latest
- docker ps (you will get a port mapped by the docker for you)
- docker port <conatainer-name> $CONTAINERPORT
- docker stop nginxserver

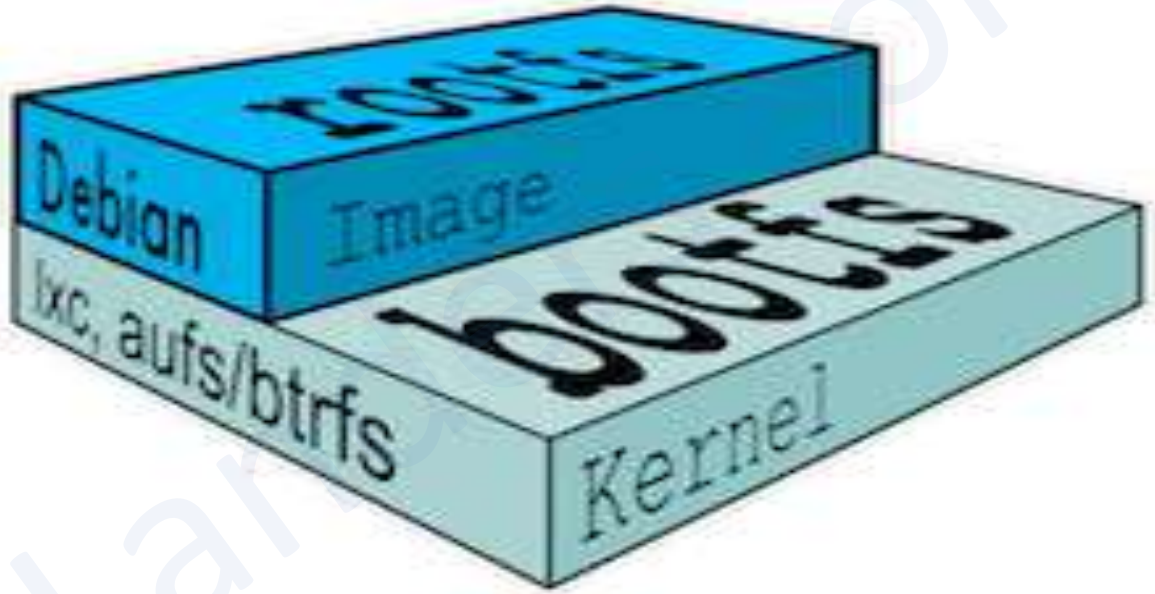
Docker Ports and Volumes

- But how to bound the container with a particulat port:
- docker run -d -p 8080:80 nginx
- Even you can bound multiple ports:
- docker run -d -p 8080:80,8081:443 nginx
- Once done stop and remove the container

Session: 6

Docker - Images

Docker Images



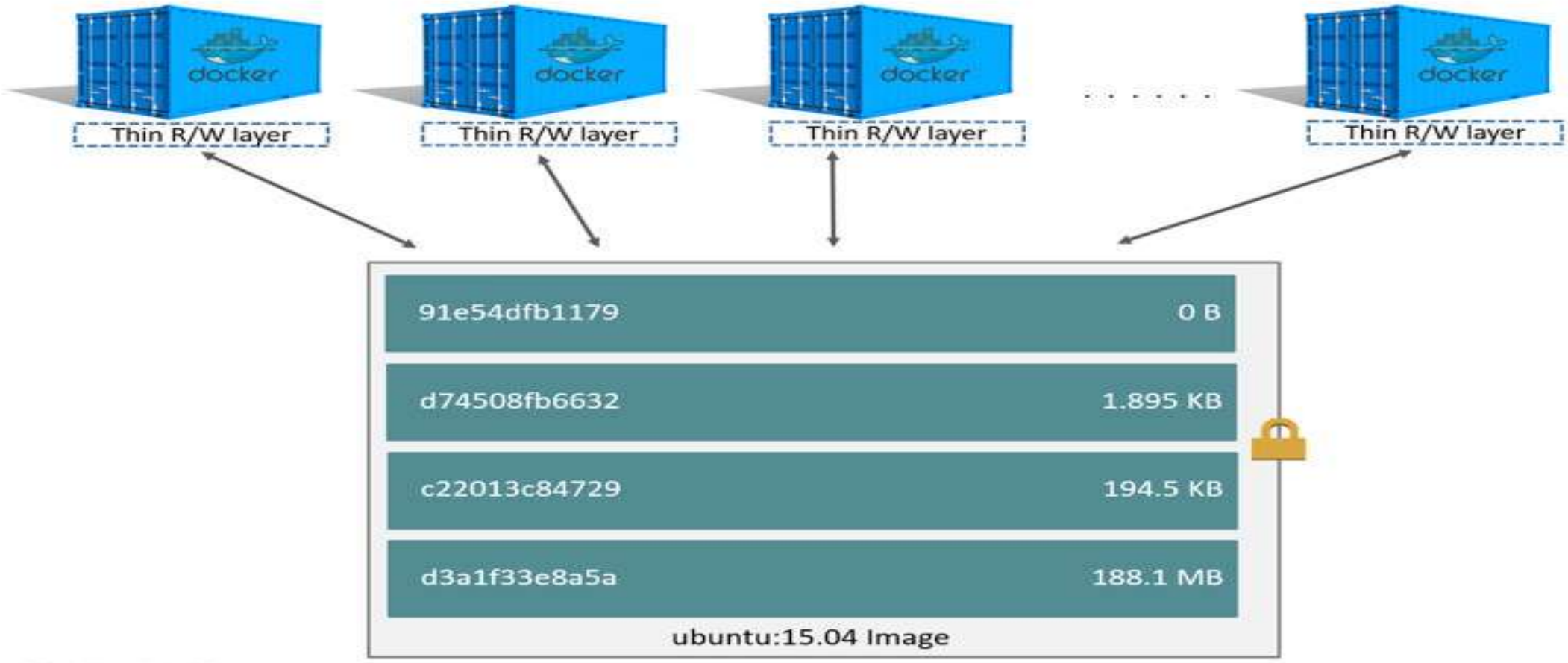
What is an image?

- An image is a collection of files + some meta data.
- Images are made of *layers, conceptually stacked on top of each other.*
- Each layer can add, change, and remove files.
- Images can share layers to optimize disk usage, transfer times, and memory use.

Container vs Image

- An image is a read-only filesystem.
- A container is an encapsulated set of processes running in a read-write copy of that filesystem.
- docker run starts a container from a given image.

Container vs Image



Store & manage images

- Images can be stored:
    - On your Docker host.
    - In a Docker registry.
- You can use the Docker client to download (pull) or upload (push) images.
- To be more accurate: you can use the Docker client to tell a Docker server to push and pull images to and from a registry.
- Lets explore docker public registry called “docker hub”

Showing current images

- Let's look at what images are on our host now.

  docker images

  docker image list

Searching for images

- We cannot list all images on a remote registry, but we can search for a specific keyword:

  docker search zookeeper

- "Stars" indicate the popularity of the image.

Downloading images

- There are two ways to download images.
  - Explicitly, with “docker pull”.
  - Implicitly, when executing “docker run” and the image is not found locally.
- Pulling an image.

  docker pull debian:jessie

- Images can have tags.
- Tags define image versions or variants.
- “docker pull ubuntu” will refer to “ubuntu:latest”.
- The :latest tag is generally updated often.

Docker Image LifeCycle

- To list docker images: docker images
- To remove docker images locally: docker rmi <image-name>
- Special Cases:
- Lets try to remove docker images used by current container: docker rmi <image-name>
- This time you will get an error, but the image can be removed forcefully using “-f” option.
- Docker rmi -f <image-name> (Note: It wont work with Image ID)

Docker Image Information

- Docker Image details:
- docker image list

- Detailed Infromation about docker image:
- ls –lrt /var/lib/docker/image/overlay2/imagedb/content/sha256

- High level certificate sign information (SHA):
- cat /var/lib/docker/image/overlay2/repositories.json

- All details with command line:
- docker image inspect <image-name>

- Even you can check all of the information about the docker image:
- docker history <image-id>

Docker Image and Container LifeCycle

- To list docker: docker ps -a
- To list docker images: docker images
- To remove container: docker rm <container-id> or <Name>
- To remove multiple container: docker rm `docker ps –a –q`
- To remove docker images locally: docker rmi <image-name>

Docker Image and Conatiner LifeCycle

- Special Cases:
- Lets try to remove docker images used by current container: docker rmi <image-name>
- This time you will get an error, but the image can be removed forcefully using “-f” option.
- Docker rmi -f <image-name> (Note: It wont work with Image ID)
- The best part is though you have removed the image forcefully, but this will not impact the current container as the container preserves the metadata in containers folder.
- You containers are safe!.

Session: 7

Building Images

Building Images Interactively

- Let’s have a Use Case:
  - We will build an image that has httpd.
  - First, we will do it manually with docker commit.
  - Then, we will use a Dockerfile and “docker build”.

Create a new container

- Let’s start from base image “centos”:
  docker run -dit --name c1 centos:7

  docker attach c1
  yum update –y

  yum install –y httpd
  exit

- Inspect the changes:
  docker diff <yourContainerId>

- Commit the changes:

docker commit -m "added httpd and updated" -a "raman khanna" c1 ramancentosimage:v1

Run & Tag the image

- Remove the previous centos container and image as well

- Let’s run the new image:

  docker run -it <newImageId> : docker run -it --name c1 ramancentosimage:v1

  rpm –qa | grep –i httpd

- Tagging images:

  docker tag <newImageId> newhttpd : docker tag ramancentosimage:v1 newhttpd

- Run it using Tag:

  docker run -it newhttpd

Dockerfile overview

- A Dockerfile is a build recipe for a Docker image.
- It contains a series of instructions telling Docker how an image is constructed.
- The “docker build” command builds an image from a Dockerfile.

First Dockerfile

- Create a directory to hold our Dockerfile.
  mkdir myimage

- Create a Dockerfile inside this directory.
  cd myimage
  vi Dockerfile

- Write below in our Dockerfile

```
FROM centos:7
RUN yum update -y
RUN yum -y install httpd
```

First Dockerfile

- “FROM” indicates the base image for our build.
- Each “RUN” line will be executed by Docker during the build.
- No input can be provided to Docker during the build.

First Dockerfile

- Build the Dockerfile:

  docker build -t httpd . Or docker build -t httpd /home/ubuntu/raman/myimage/

- -t indicates the tag to apply to the image.

- . indicates the location of the Directory of Dockerfile.

Run & Tag the image

- Let’s run the new images:

  docker run -it <newImageId>

  rpm –qa | grep –i httpd

Using Image & viewing history

- The history command lists all the layers composing an image.
- For each layer, it shows its creation time, size, and creation command.
- When an image was built with a Dockerfile, each layer corresponds to a line of the Dockerfile.

  docker history httpd

Dockerfile

- Dockerfile:

```
FROM centos:7
MAINTAINER  Raman Khanna raman.khanna@TechLanders.com
RUN mkdir /data
RUN yum update -y
RUN yum -y install httpd   php
RUN echo " TechLanders Solutions Deals in DevOps and Cloud" > /var/www/html/index.html
EXPOSE 80
VOLUME  /data
RUN echo "httpd" >> /root/.bashrc
CMD ["/bin/bash"]
```

- Build the image:

```
docker build -t webapp:v1 .
```

```
docker run -dit --name c1 -p 8080:80 webapp:v1
```

```
curl 172.31.84.13:8080
```

Browse in browser as well

COPY Instruction

- For Use Case, let’s build a container that copy file from localhost

  echo " TechLanders Solutions Deals in DevOps and Cloud " > /home/ubuntu/image/index.html

Dockerfile content

```
FROM centos:7
RUN yum update -y
RUN yum install -y httpd
COPY ./index.html /var/www/html/index.html
EXPOSE 80
WORKDIR /var/www/html
CMD ["httpd","-D","FOREGROUND"]
```

Docker Restart Policy

$ docker run -dit --restart unless-stopped centos

Flag	Description
no	**Do not automatically restart the container. (the default)**
on-failure	**Restart the container if it exits due to an error, which manifests as a non-zero exit code.**
always	**Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted. (See the second bullet listed in restart policy details)**
unless-stopped	**Similar to always, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.**

Docker Registries

Local Registry (Local to Host)

Remote Registry (Private)

Global Registry (Public)

Docker Trusted Registry

- Enterprise-grade image storage solution from Docker
- Highly Secure
- Image and job management with CICD
- HA Availability
- Efficiency with Near to user storage and bandwidth sharing
- Security Scanning
- Similar tools in market Sonatype Nexus (open source), AWS ECR (PaaS), azure Container Registry, GCP container Registry etc

Docker Limitations

- Hardware Issues? High Availability?
- How IP address will be managed for failover?
- Scaling?
- Auto Healing?
- Autoscaling?
- No Application Management - Only Containerization
- Updation of application/management

Why Kubernetes

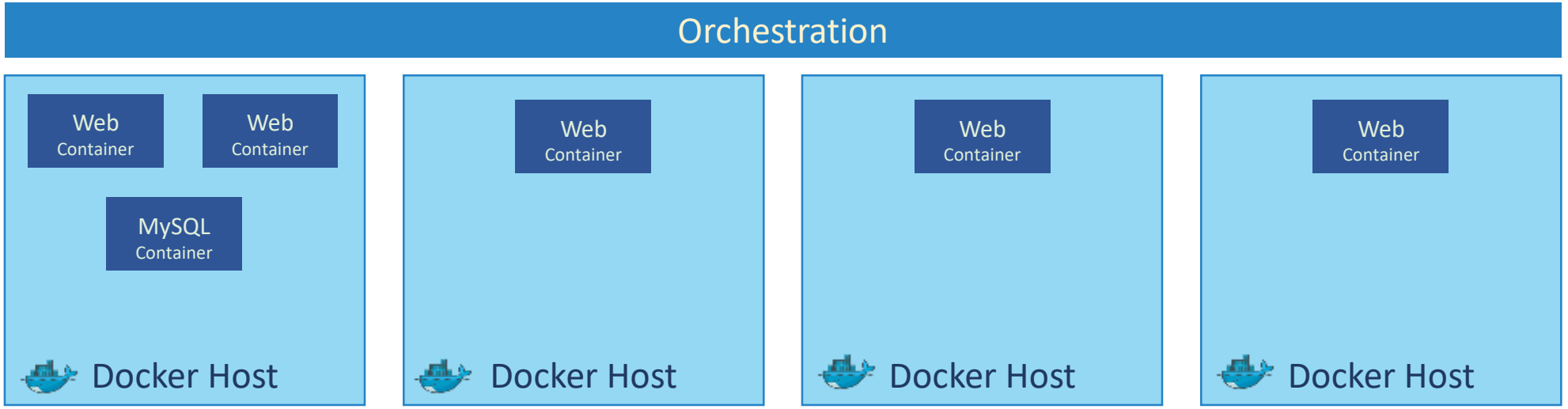
- Kubernetes can schedule and run application containers on clusters of physical or virtual machines.
- **host-centric** infrastructure to a **container-centric** infrastructure.
- Orchestrator
- Load balancing
- Auto Scaling
- Application Health checks
- Rolling updates

Container Orchestration

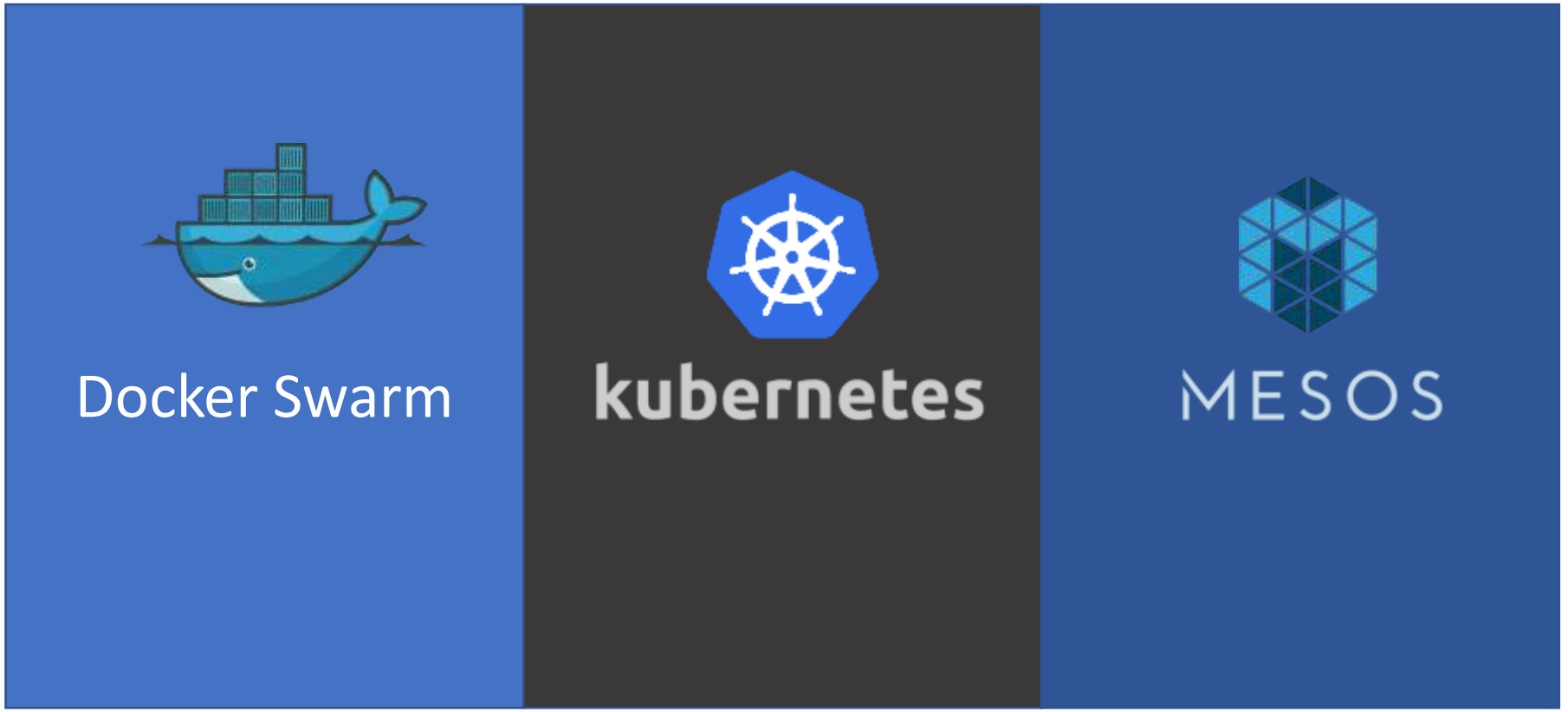
Containers Limitation?

High Availability?
Overlay Network?
Application Centric or Infra Centric?
Versioning of Application – Rollout, Rollback?
Scaling?
Autoscaling?
Monitoring?
Dependency between containers?

Container orchestration



Orchestration Technologies



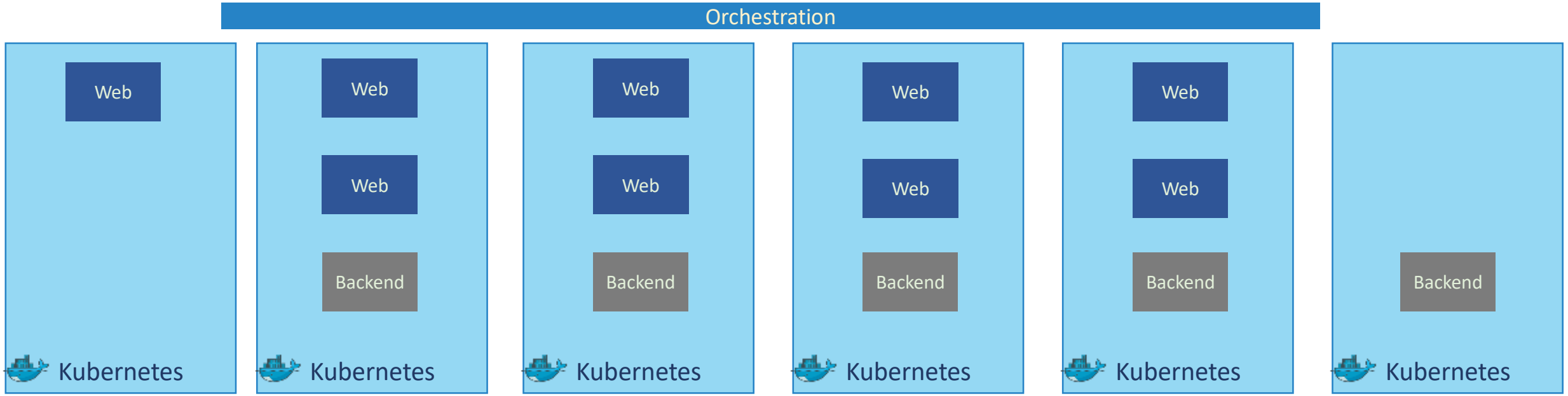
What is Kubernetes?

- The Kubernetes project was started by Google in 2014.
- Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale.
- Kubernetes can run on a range of platforms, from your laptop, to VMs on a cloud provider, to rack of bare metal servers.
- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.
- **portable**: with all public, private, hybrid, community cloud
- **self-healing**: auto-placement, auto-restart, auto-replication, auto-scaling

Why Kubernetes

- Kubernetes can schedule and run application containers on clusters of physical or virtual machines.
- **host-centric** infrastructure to a **container-centric** infrastructure.
- Orchestrator
- Load balancing
- Auto Scaling
- Application Health checks
- Rolling updates

Kubernetes Advantage



And that is kubernetes..

Setup

Minikube

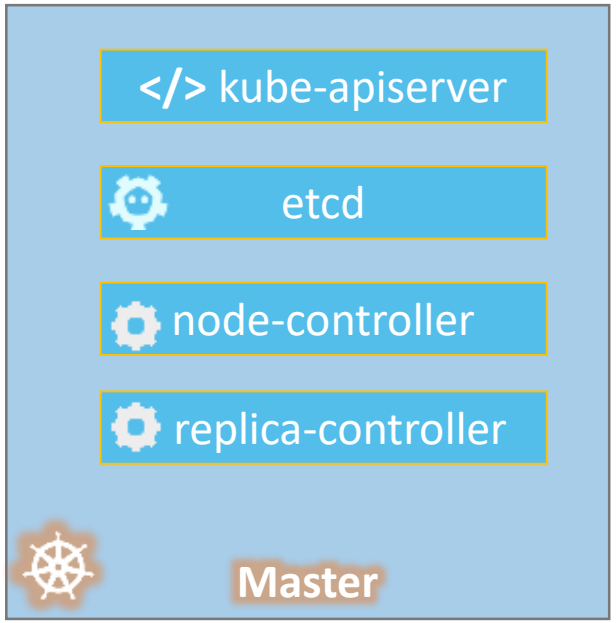
Kubeadm

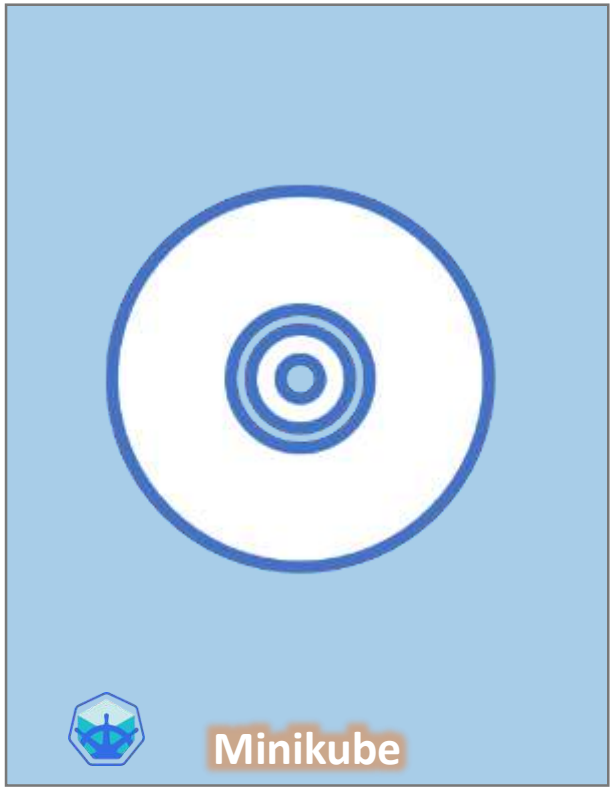
Google Cloud Platform

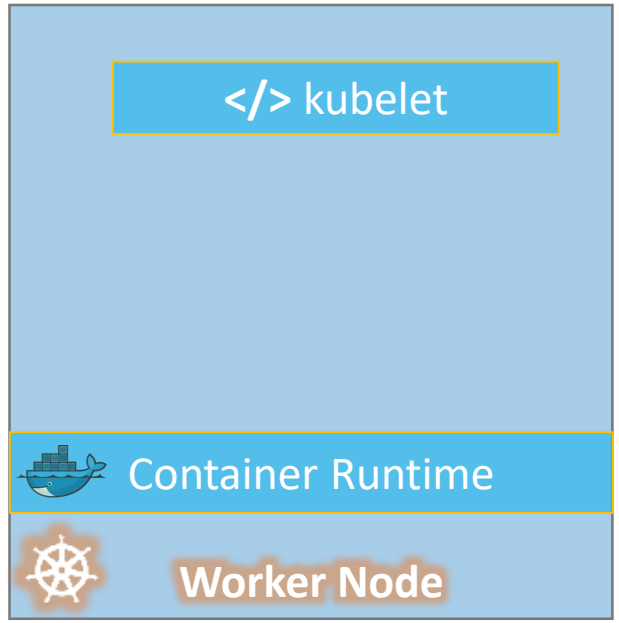
Amazon Web Services

play-with-k8s.com

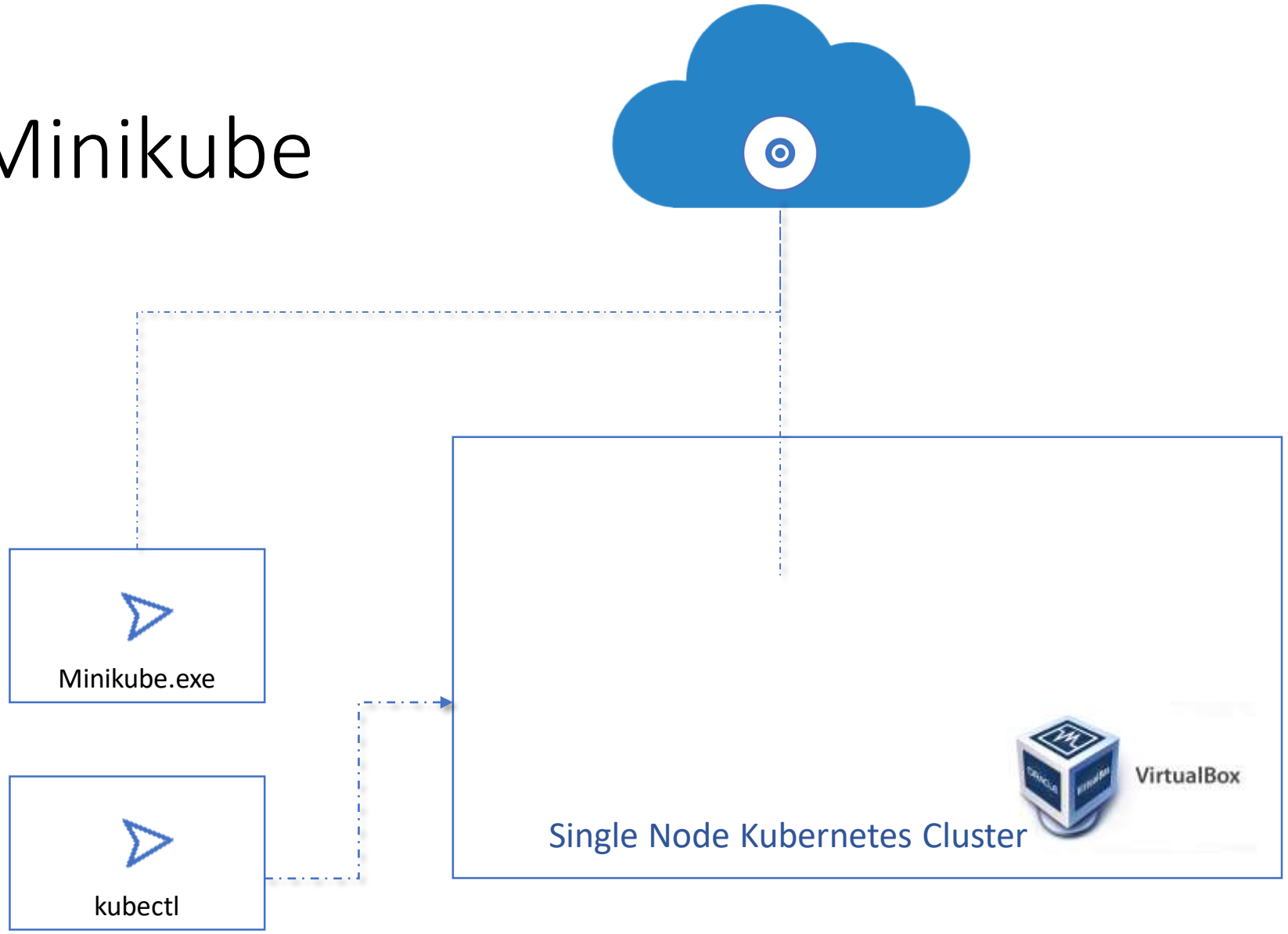
Minikube


</> kube-apiserver
etcd
node-controller
replica-controller
Master


Minikube


</> kubelet
Container Runtime
Worker Node

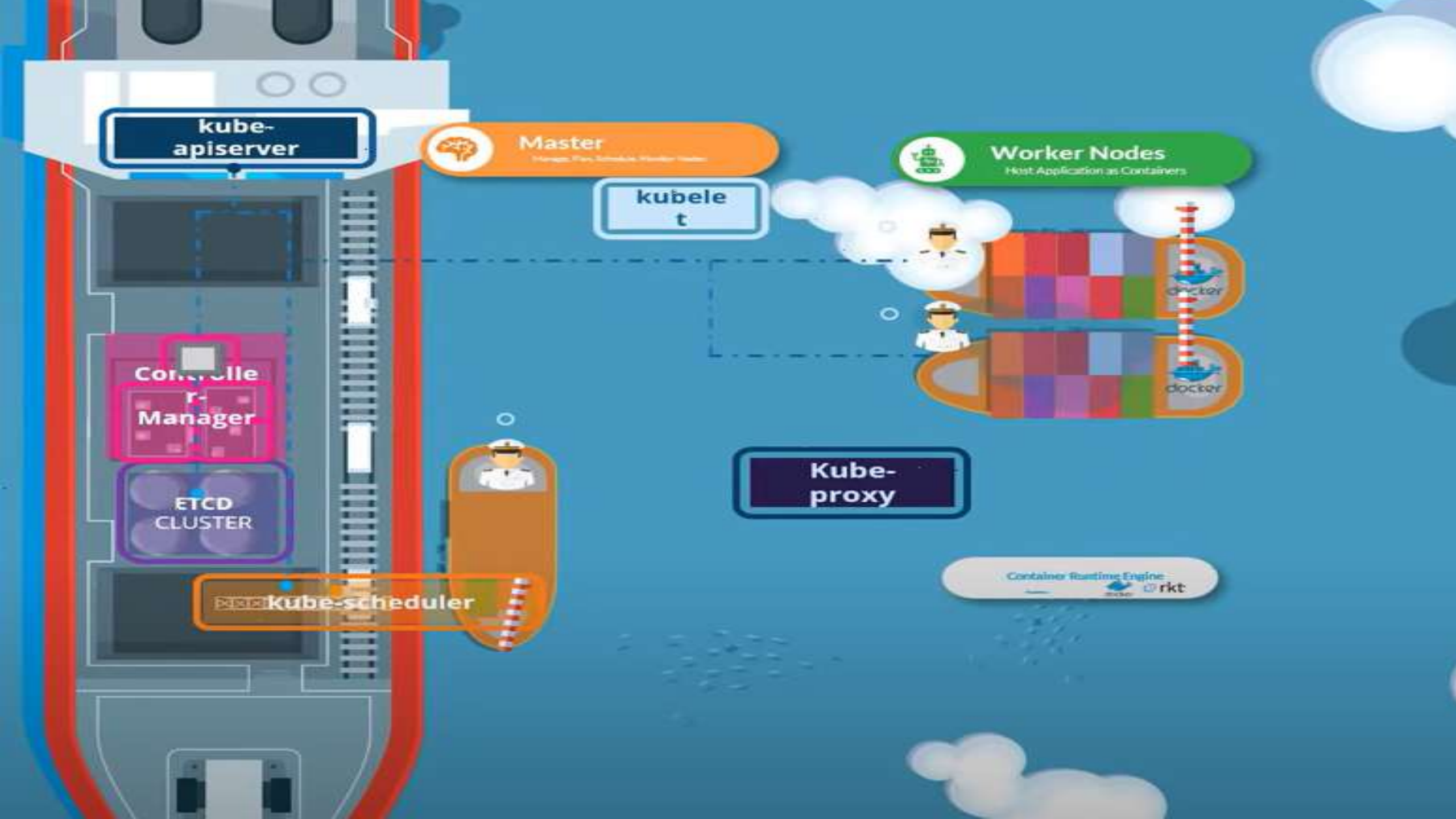
Minikube


Minikube.exe
kubectl
VirtualBox
Single Node Kubernetes Cluster

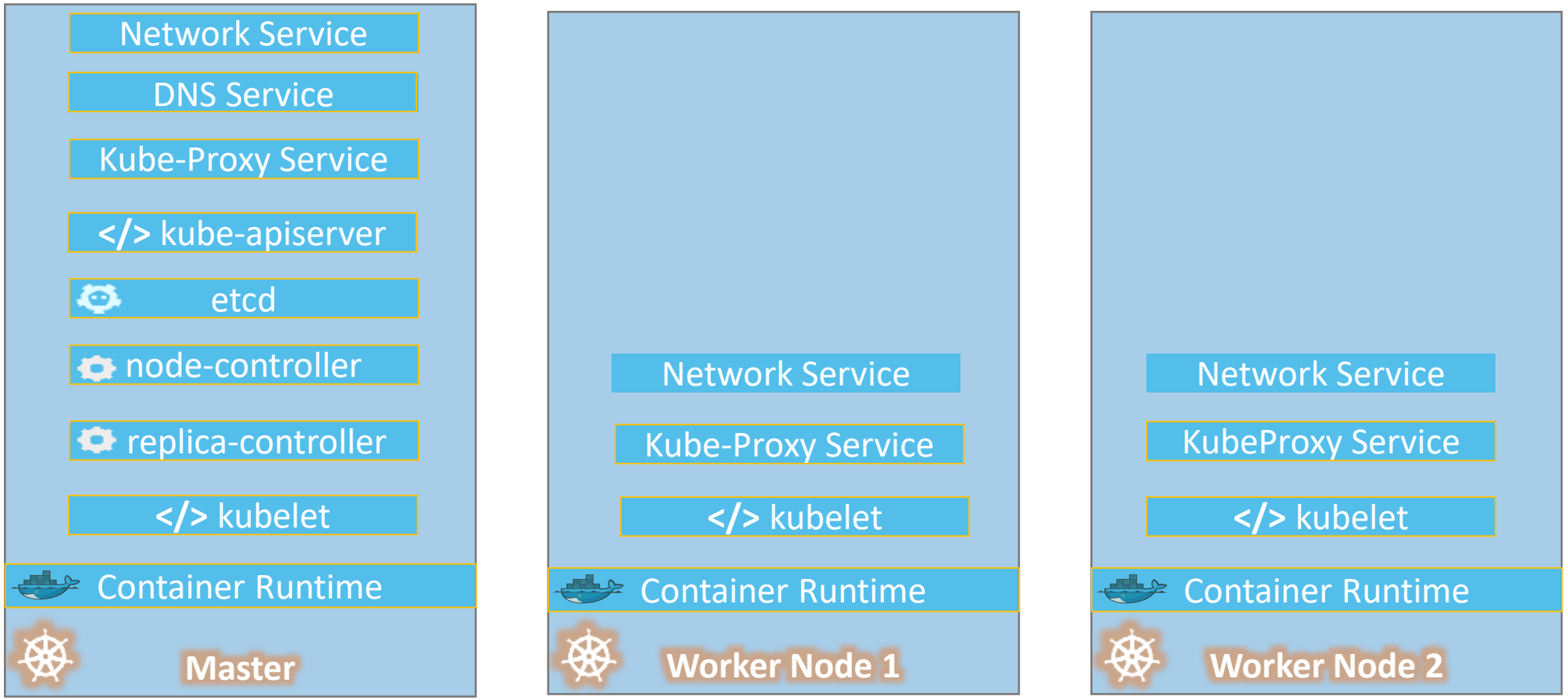
Setup - kubeadm

Kubernetes Cluster

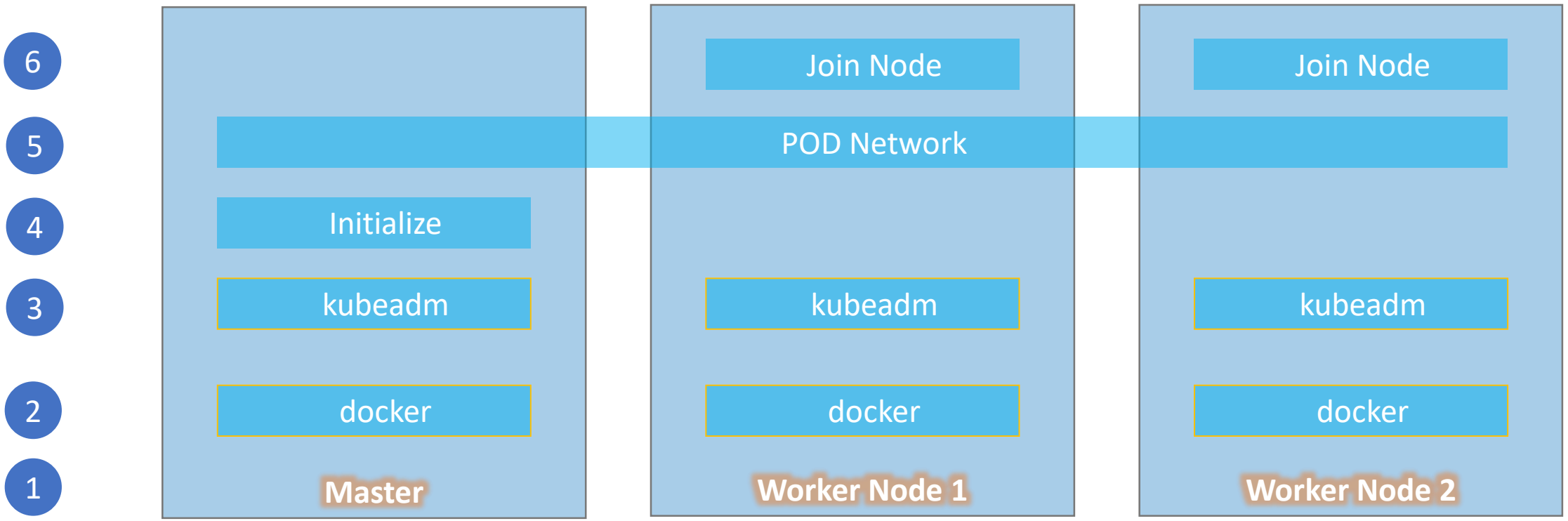
- A Kubernetes cluster consists of two types of resources:
- **Master**: Which coordinates with the cluster
- The Master is responsible for managing the cluster. The master coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.
- **Nodes**: Are the workers that run application
- A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.
- Masters manage the cluster and the nodes are used to host the running applications.
- **The nodes communicate with the master using the Kubernetes API**, which the master exposes.


kube-
apiserver
Master
kubele
t
Worker Nodes
Host Application as Containers
docker
docker
Controlle
r-
Manager
ETCD
CLUSTER
kube-scheduler
Kube-
proxy
Container Runtime Engine
rkt

kubeadm


Network Service
DNS Service
Kube-Proxy Service
</> kube-apiserver
etcd
node-controller
replica-controller
</> kubelet
Container Runtime
Master
Network Service
Kube-Proxy Service
</> kubelet
Container Runtime
Worker Node 1
Network Service
KubeProxy Service
</> kubelet
Container Runtime
Worker Node 2

Steps


6
5
4
3
2
1
Join Node
Join Node
POD Network
Initialize
kubeadm
kubeadm
kubeadm
docker
docker
docker
Master
Worker Node 1
Worker Node 2

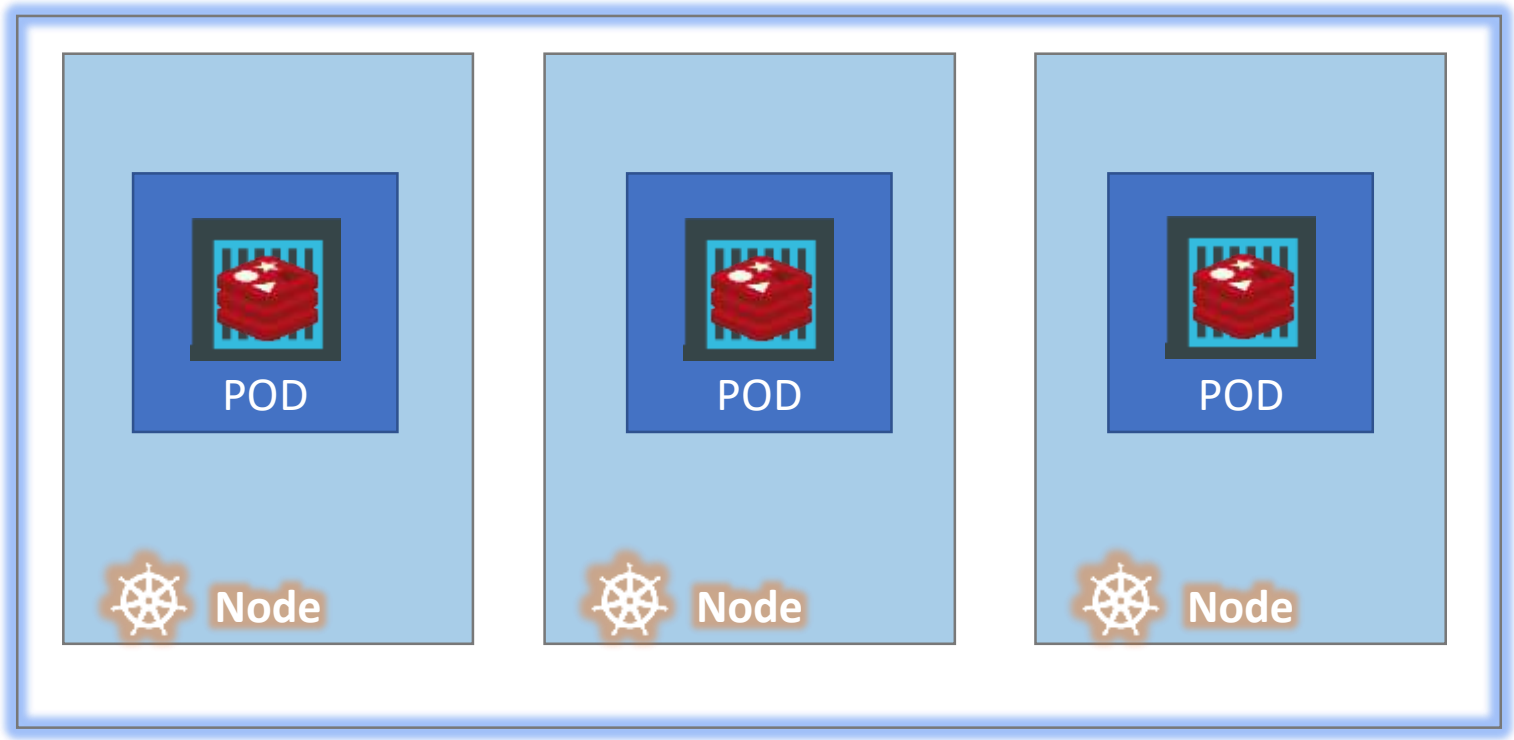
POD

Assumptions

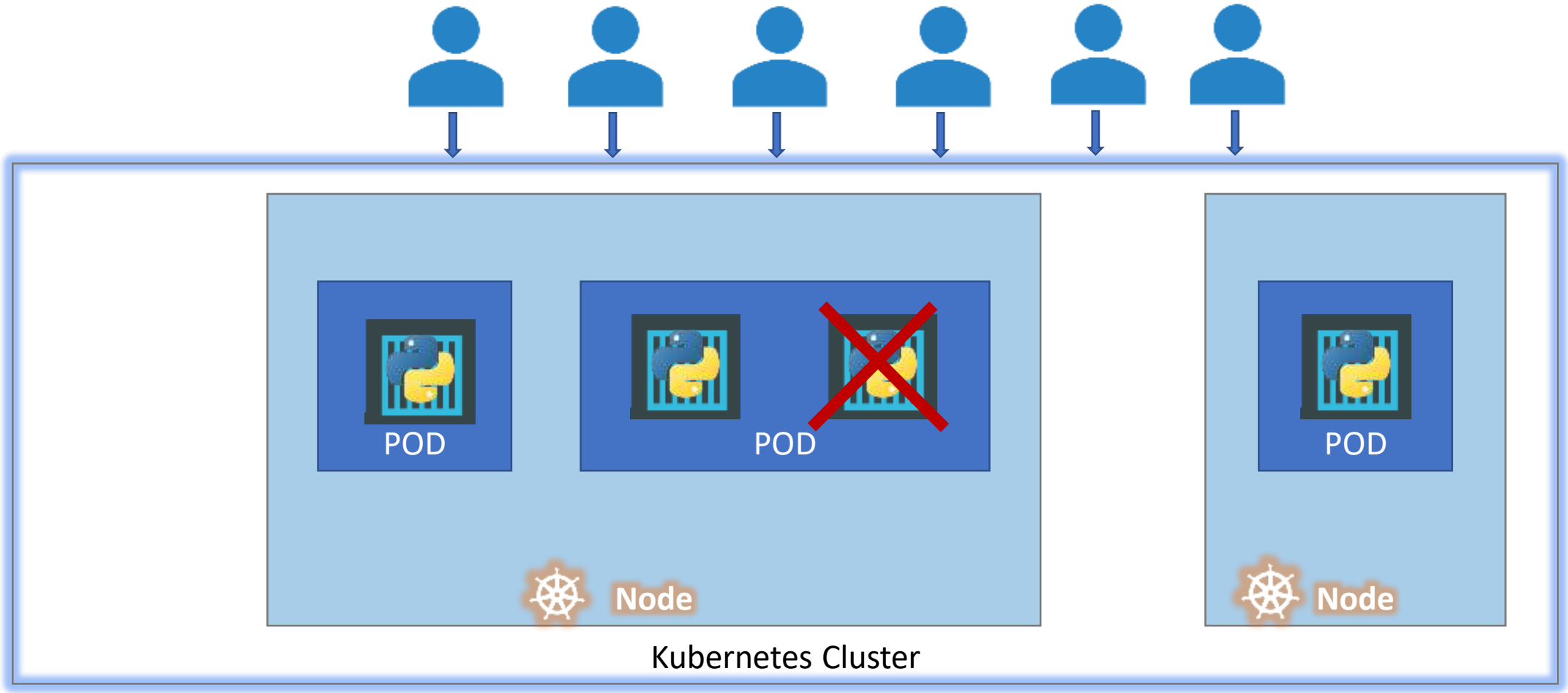
Docker Image

Kubernetes Cluster

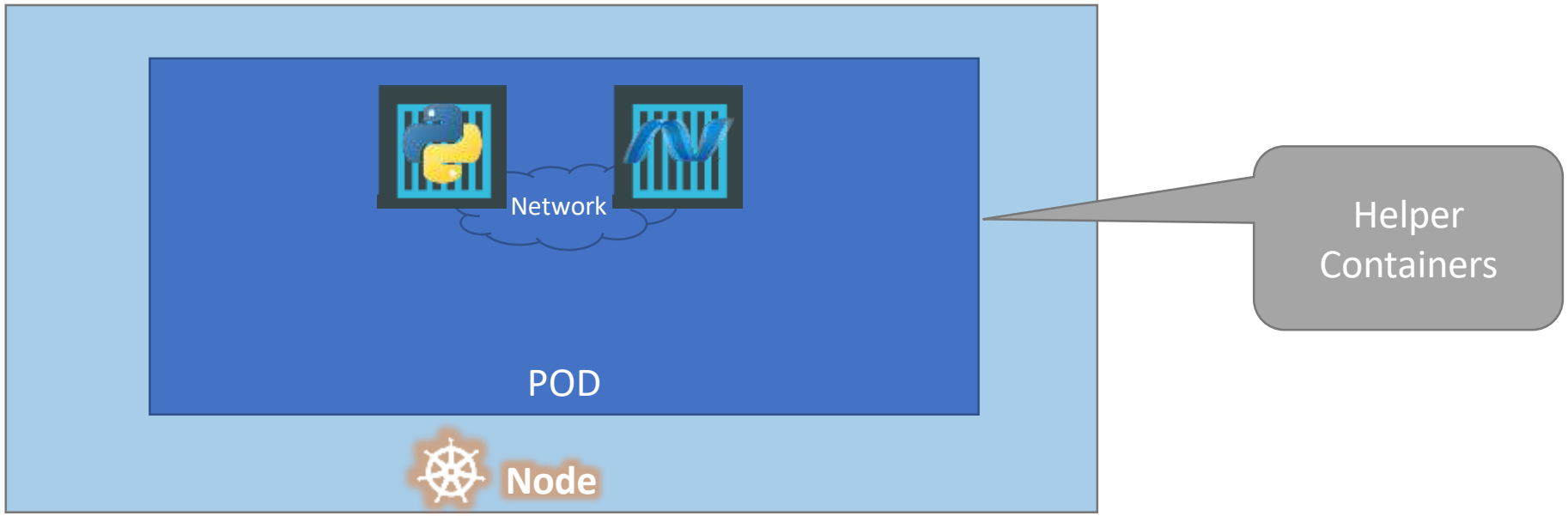
POD



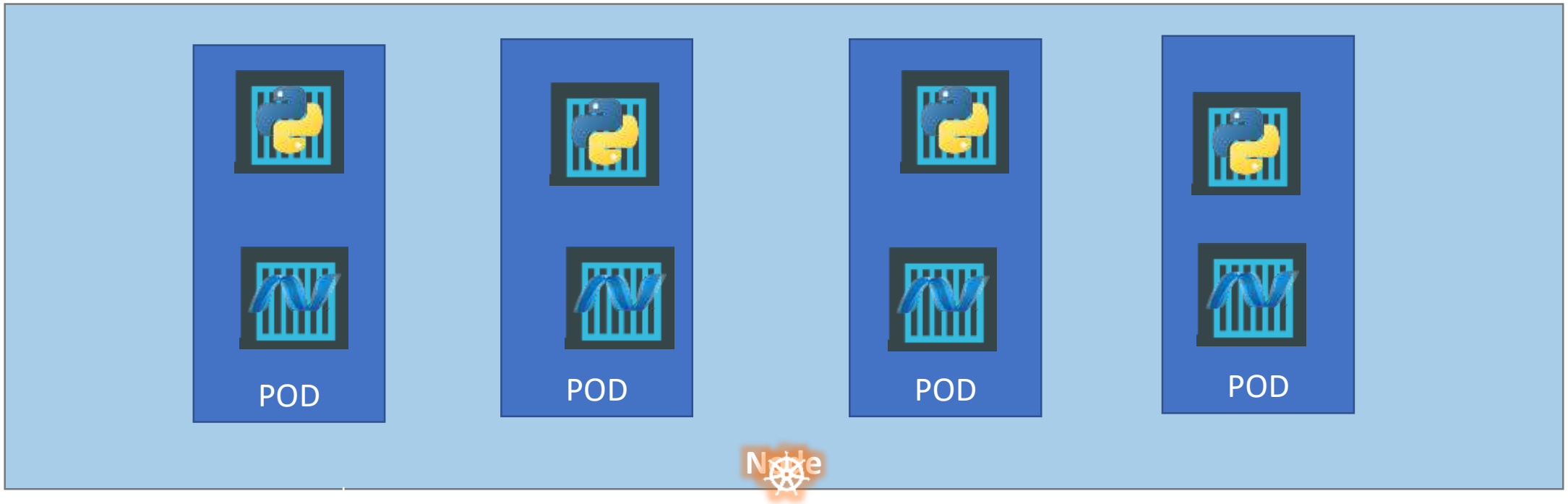
POD



Multi-Container PODs



PODs Again!



Note: I am avoiding networking and load balancing details to keep explanation simple.

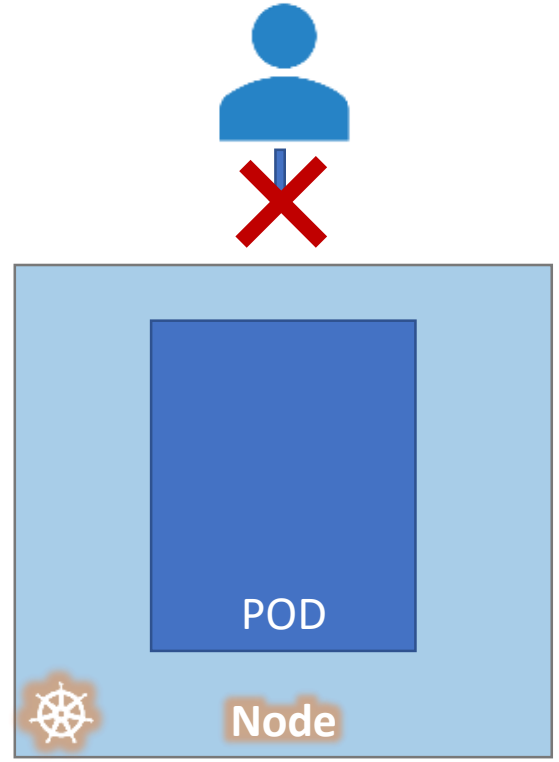
kubectl

- `kubectl run nginx --image nginx`

`kubectl get pods`

```
C:\Kubernetes>kubectl get pods
NAME                     READY     STATUS              RESTARTS   AGE
nginx-8586cf59-whssr     0/1       ContainerCreating   0          3s
```

```
C:\Kubernetes>kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-8586cf59-whssr     1/1       Running   0          8s
```



YAML Introduction

POD

With YAML

YAML in Kubernetes

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
    - name: nginx-container
      image: nginx
```

String

String

List/Array

1st Item in List

- kubectl create -f pod-definition.yml

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

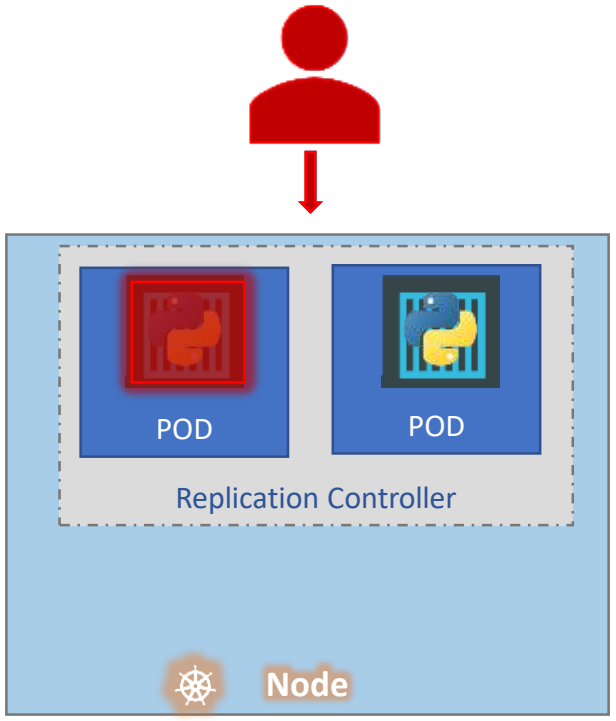
Commands

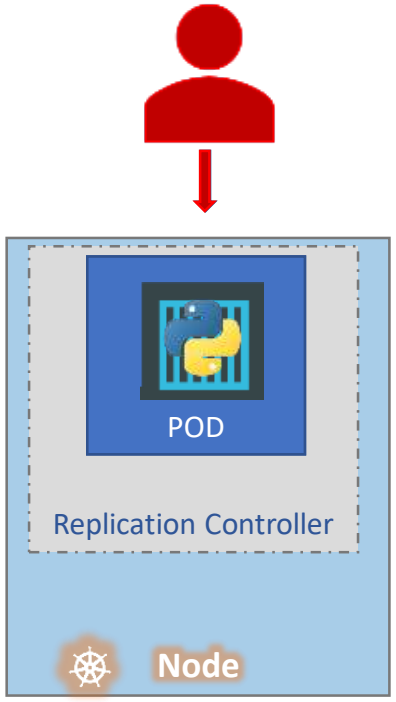
```
> kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
myapp-pod       1/1       Running   0          20s
```

```
> kubectl describe pod myapp-pod
Name:         myapp-pod
Namespace:    default
Node:         minikube/192.168.99.100
Start Time:   Sat, 03 Mar 2018 14:26:14 +0800
Labels:       app=myapp
              name=myapp-pod
Annotations:  <none>
Status:       Running
IP:           10.244.0.24
Containers:
  nginx:
    Container ID:   docker://830bb56c8c42a86b4bb70e9c1488fae1bc38663e4918b6c2f5a783e7688b8c9d
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:4771d09578c7c6a65299e110b3ee1c0a2592f5ea2618d23e4ffe7a4cab1ce5de
    Port:           <none>
    State:          Running
      Started:      Sat, 03 Mar 2018 14:26:21 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-x95w7 (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Events:
  Type    Reason                 Age   From               Message
  ----    ------                 ----  ----               -------
  Normal  Scheduled              34s   default-scheduler  Successfully assigned myapp-pod to minikube
  Normal  SuccessfulMountVolume  33s   kubelet, minikube  MountVolume.SetUp succeeded for volume "default-token-x95w7"
  Normal  Pulling                33s   kubelet, minikube  pulling image "nginx"
  Normal  Pulled                 27s   kubelet, minikube  Successfully pulled image "nginx"
  Normal  Created                27s   kubelet, minikube  Created container
  Normal  Started                27s   kubelet, minikube  Started container
```

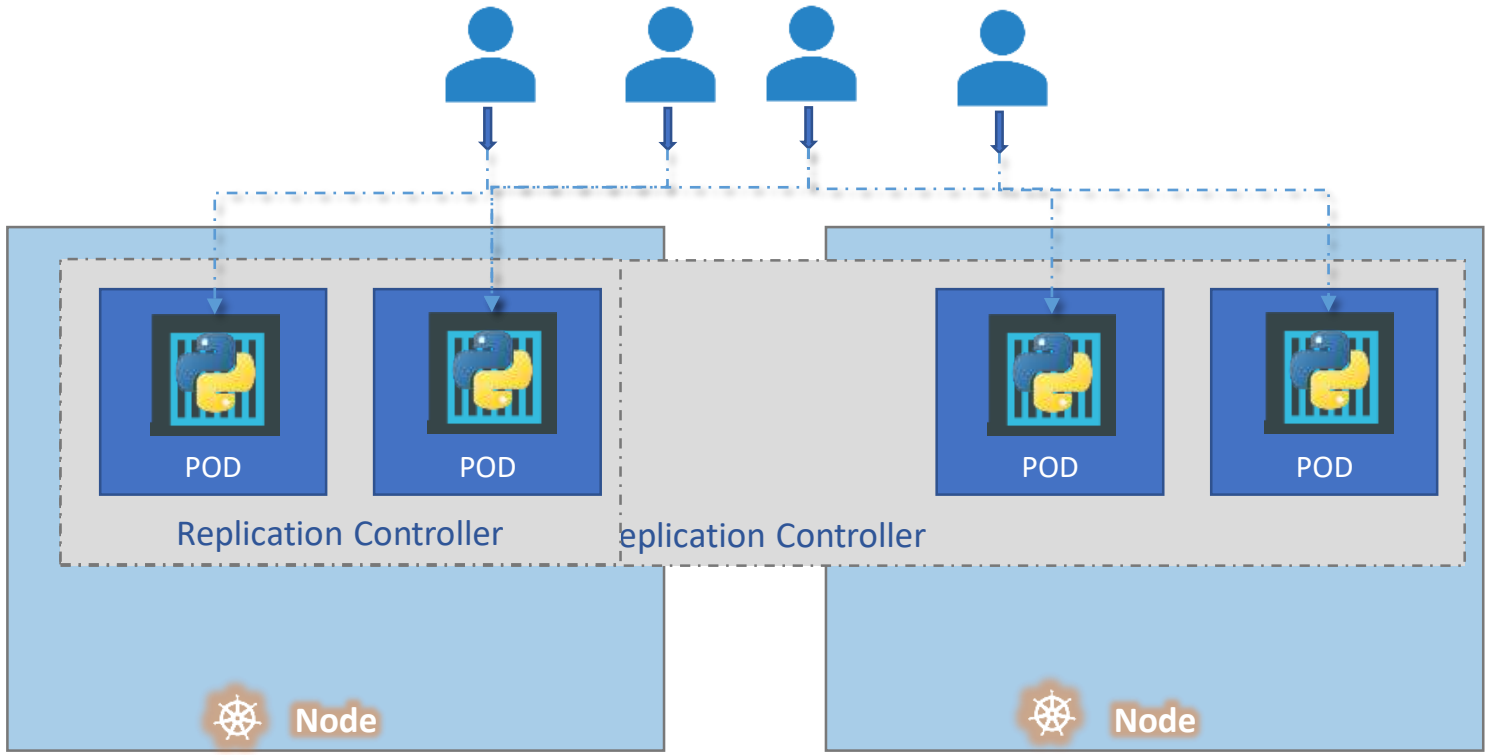
Replication Controller

High Availability





Load Balancing & Scaling



- Replication Controller

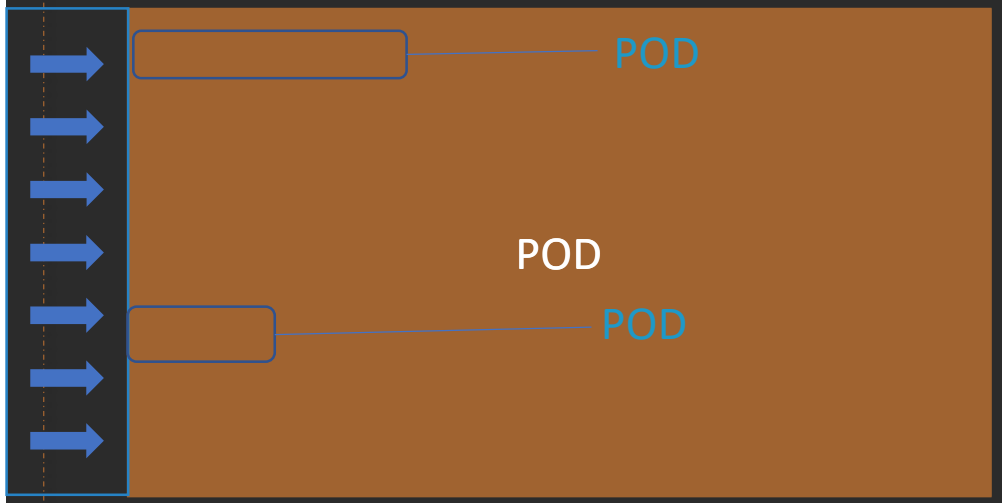
Replica Set

rc-definition.yml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
      app: myapp
      type: front-end
spec:
  template:
    replicas: 3
```

Replication Controller

Replication Controller



pod-definition.yml

```yaml
apiVersion: v1
kind: Pod

metadata:
 name: myapp-pod
 labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: nginx-container
    image: nginx
```

- > kubectl create -f rc-definition.yml

```
replicationcontroller "myapp-rc" created
```

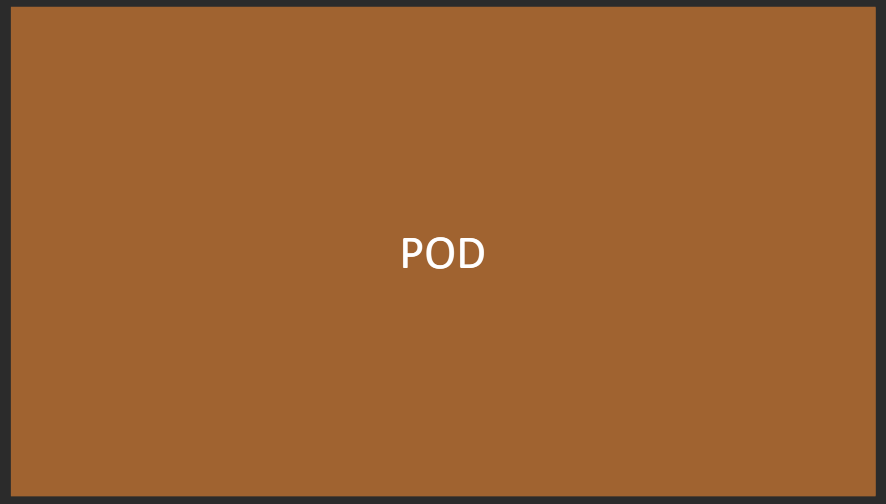
> kubectl get replicationcontroller

```
NAME       DESIRED   CURRENT   READY     AGE
myapp-rc   3         3         3         19s
```

> kubectl get pods

```
NAME             READY     STATUS    RESTARTS   AGE
myapp-rc-4lvk9   1/1       Running   0          20s
myapp-rc-mc2mf   1/1       Running   0          20s
myapp-rc-px9pz   1/1       Running   0          20s
```

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-repl
  labels:
      app: myapp
      type: front-end
spec:
  template:
      
      
  replicas: 3
  selector:
      matchLabels:
          type: front-end
```

```
error: unable to recognize "replicaset-
definition.yml": no matches for /, Kind=ReplicaSet
```

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
  labels:
      app: myapp
      type: front-end
spec:
  containers:
  - name: nginx-container
    image: nginx
```

- > kubectl create –f replicaset-definition.yml

```
replicaset "myapp-replicaset" created
```

> kubectl get replicaset

```
NAME            DESIRED   CURRENT   READY     AGE
myapp-replicaset   3         3         3         19s
```

> kubectl get pods

```
NAME                    READY     STATUS    RESTARTS   AGE
myapp-replicaset-9ddl9   1/1       Running   0          45s
myapp-replicaset-9jtpx   1/1       Running   0          45s
myapp-replicaset-hq84m   1/1       Running   0          45s
```

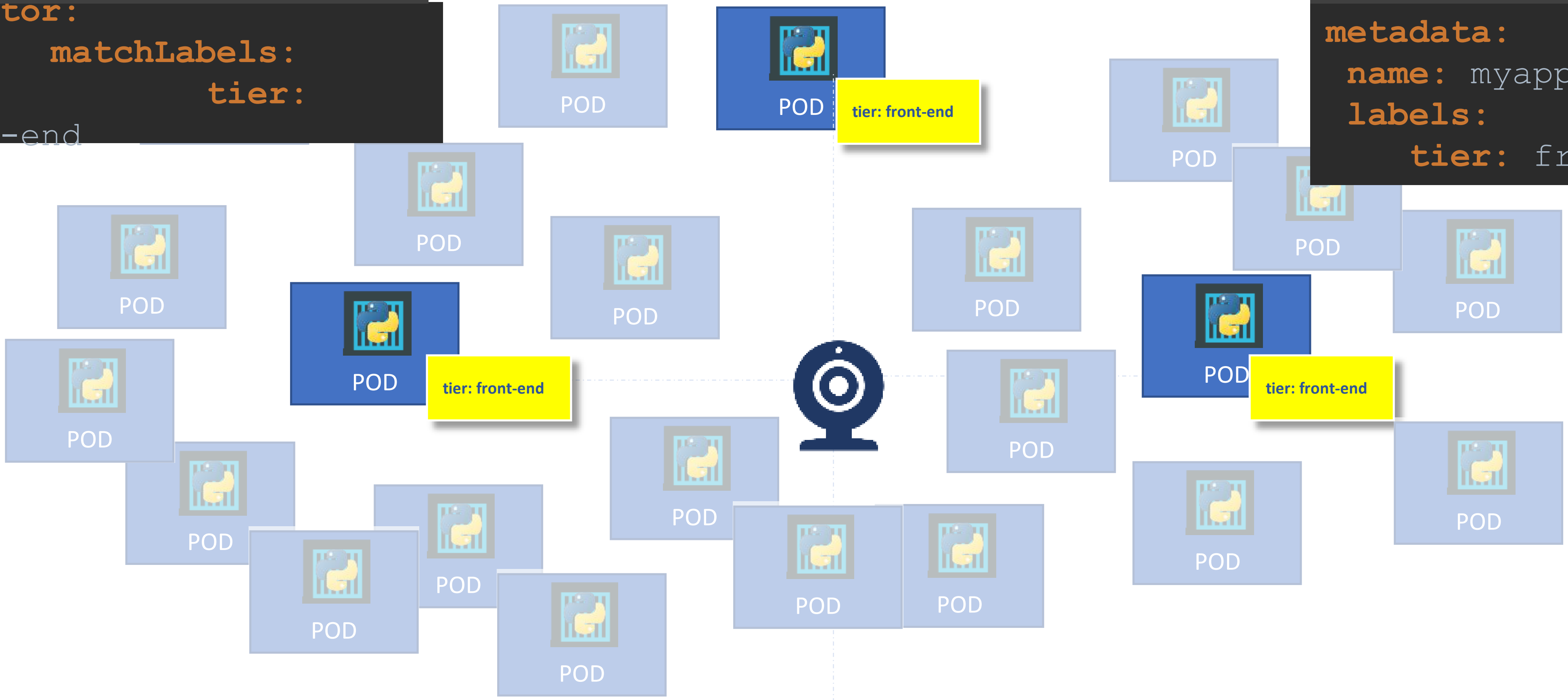
Labels and Selectors

replicaset-definition.yml

```yaml
selector:
    matchLabels:
        tier: front-end
```

pod-definition.yml

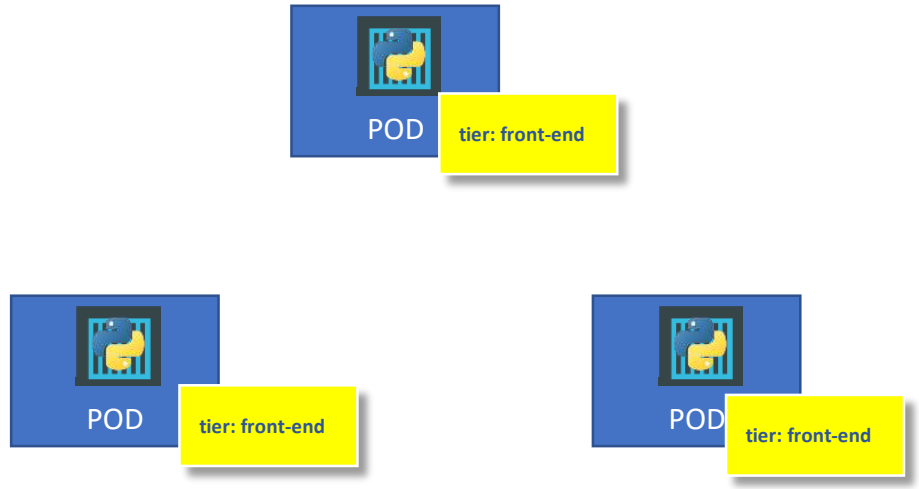
```yaml
metadata:
  name: myapp-pod
  labels:
     tier: front-end
```



replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
          app: myapp
          type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 3
  selector:
    matchLabels:
        type: front-end
```

Template



Scale

```
> kubectl replace -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 replicaset myapp-replicaset
```

TYPE NAME

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 6
  selector:
    matchLabels:
      type: front-end
```

commands

```
> kubectl create –f replicaset-definition.yml
```

```
> kubectl get replicaset
```

```
> kubectl delete replicaset myapp-replicaset
```

*Also deletes all underlying PODs

```
> kubectl replace -f replicaset-definition.yml
```

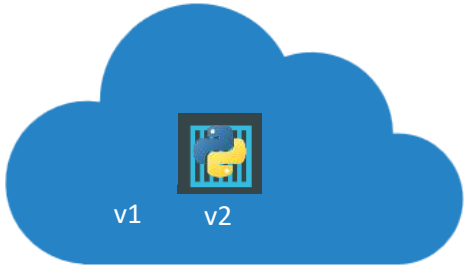
```
> kubectl scale –replicas=6 -f replicaset-definition.yml
```

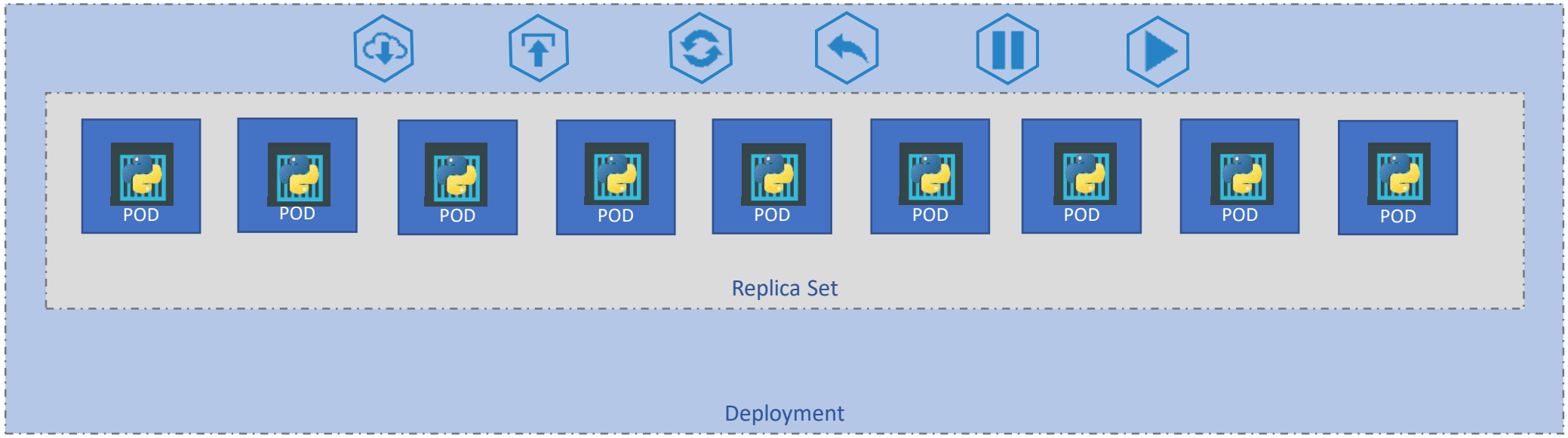
Demo

ReplicaSet

Deployment

Deployment


v1
v2


POD
POD
POD
POD
POD
POD
POD
POD
POD
Replica Set
Deployment

Definition

```
> kubectl create -f deployment-definition.yml
deployment "myapp-deployment" created
```

```
> kubectl get deployments
NAME              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE

myapp-deployment  3         3         3            3           21s
```

```
> kubectl get replicaset
NAME                        DESIRED   CURRENT   READY     AGE

myapp-deployment-6795844b58 3         3         3         2m
```

```
> kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
myapp-deployment-6795844b58-5rbjl 1/1       Running   0          2m
myapp-deployment-6795844b58-h4w55 1/1       Running   0          2m
myapp-deployment-6795844b58-lfjhv 1/1       Running   0          2m
```

deployment-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-deployment
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
         app: myapp
         type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
       type: front-end
```

commands

```
> kubectl get all
NAME                     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/myapp-deployment   3         3         3            3           9h

NAME                               DESIRED   CURRENT   READY     AGE
rs/myapp-deployment-6795844b58   3         3         3         9h

NAME                                    READY     STATUS    RESTARTS   AGE
po/myapp-deployment-6795844b58-5rbjl   1/1       Running   0          9h
po/myapp-deployment-6795844b58-h4w55   1/1       Running   0          9h
po/myapp-deployment-6795844b58-lfjhv   1/1       Running   0          9h
```

Demo

Deployment

Demo

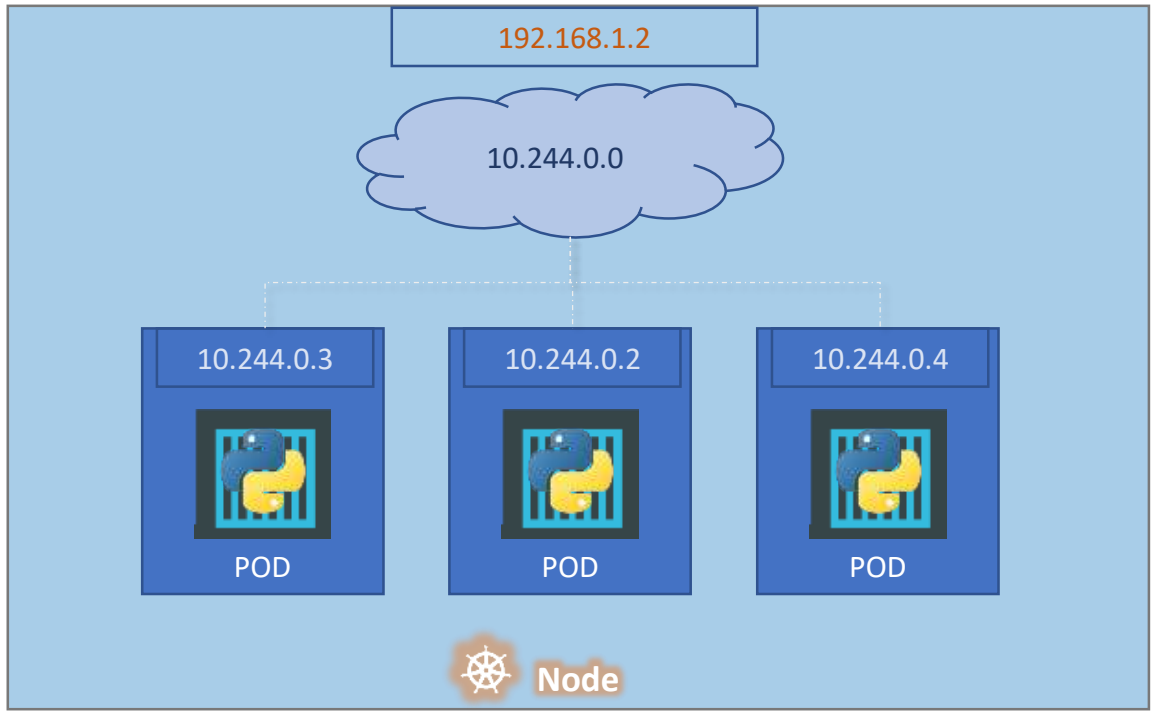
Deployment

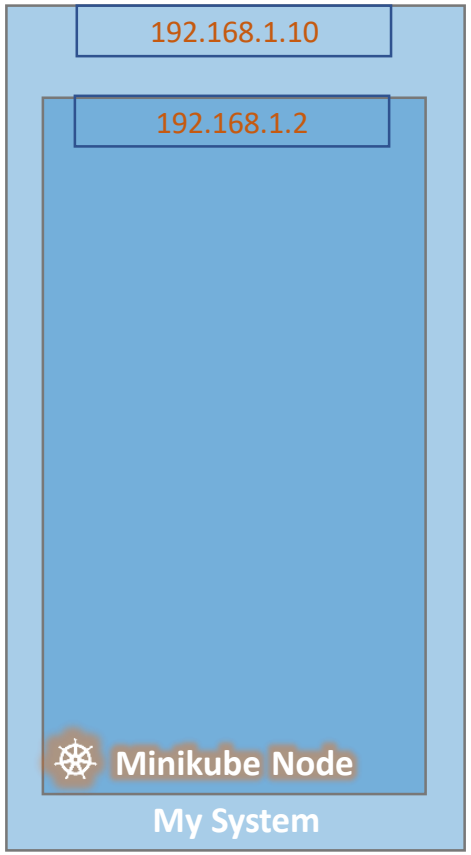
Networking 101

Raman Khanna

Kubernetes Networking - 101

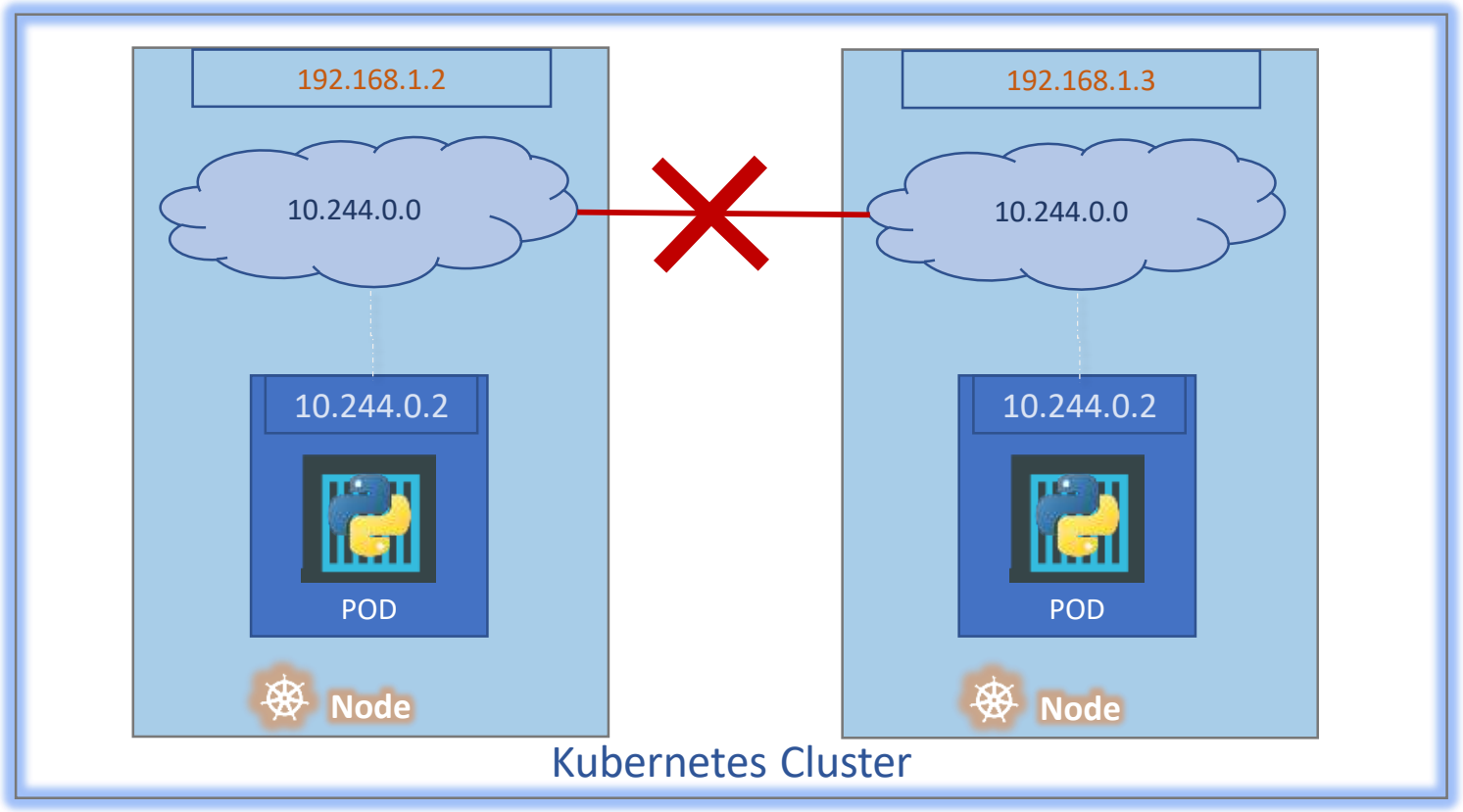
- IP Address is assigned to a POD





Cluster Networking

- All containers/PODs can communicate to one another without NAT
- All nodes can communicate with all containers and vice-versa without NAT

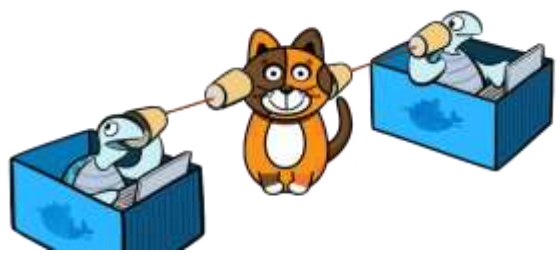



CISCO

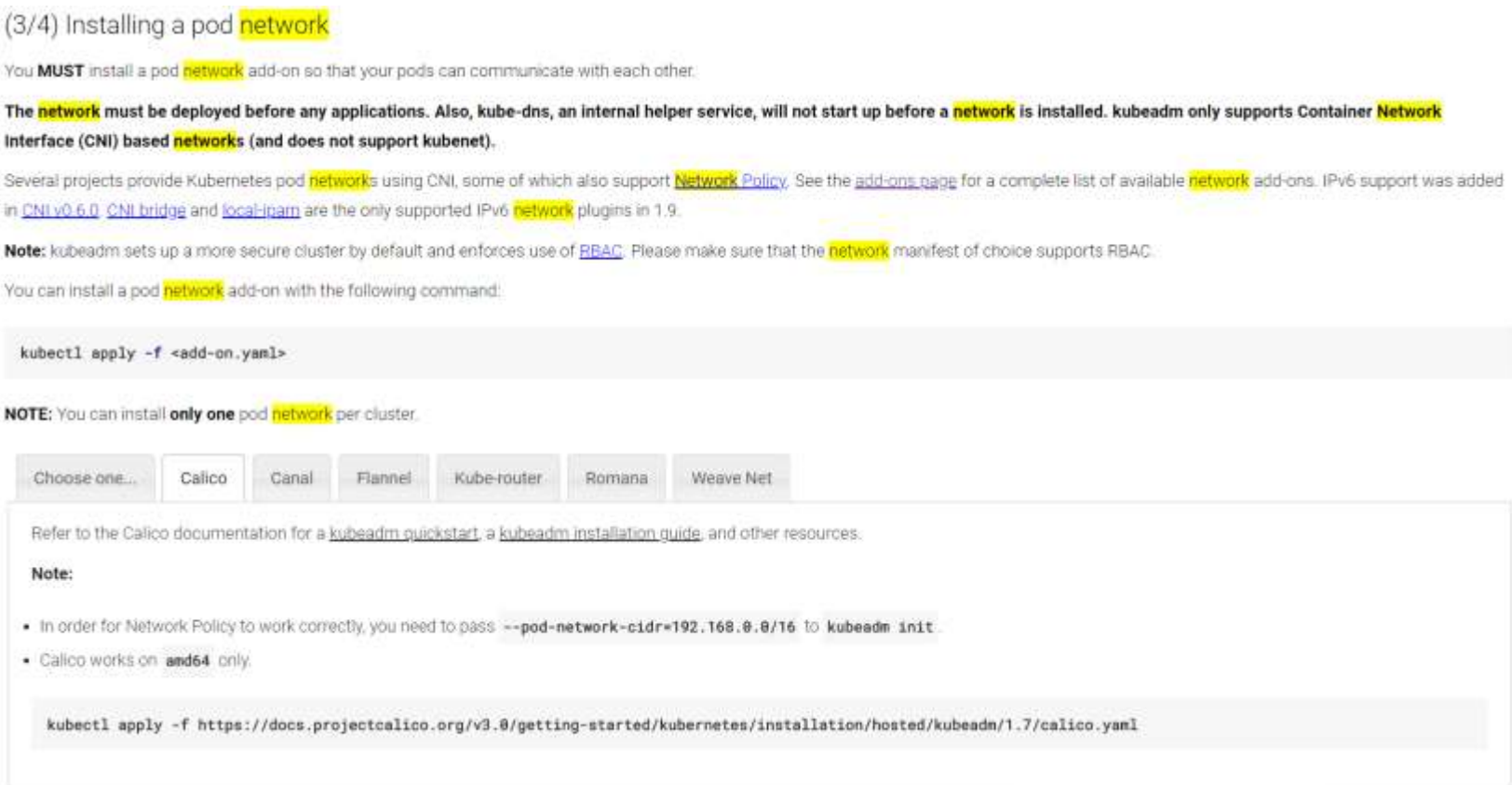

cilium


flannel


vmware
NSX



Cluster Networking Setup



(3/4) Installing a pod network

You **MUST** install a pod network add-on so that your pods can communicate with each other.

The network must be deployed before any applications. Also, kube-dns, an internal helper service, will not start up before a network is installed. kubeadm only supports Container Network Interface (CNI) based networks (and does not support kubenet).

Several projects provide Kubernetes pod networks using CNI, some of which also support Network Policy. See the add-ons page for a complete list of available network add-ons. IPv6 support was added in CNI v0.6.0. CNI bridge and local-ipam are the only supported IPv6 network plugins in 1.9.

Note: kubeadm sets up a more secure cluster by default and enforces use of RBAC. Please make sure that the network manifest of choice supports RBAC.

You can install a pod network add-on with the following command:

```
kubectl apply -f <add-on.yaml>
```

NOTE: You can install **only one** pod network per cluster.

Choose one... | Calico | Canal | Flannel | Kube-router | Romana | Weave Net

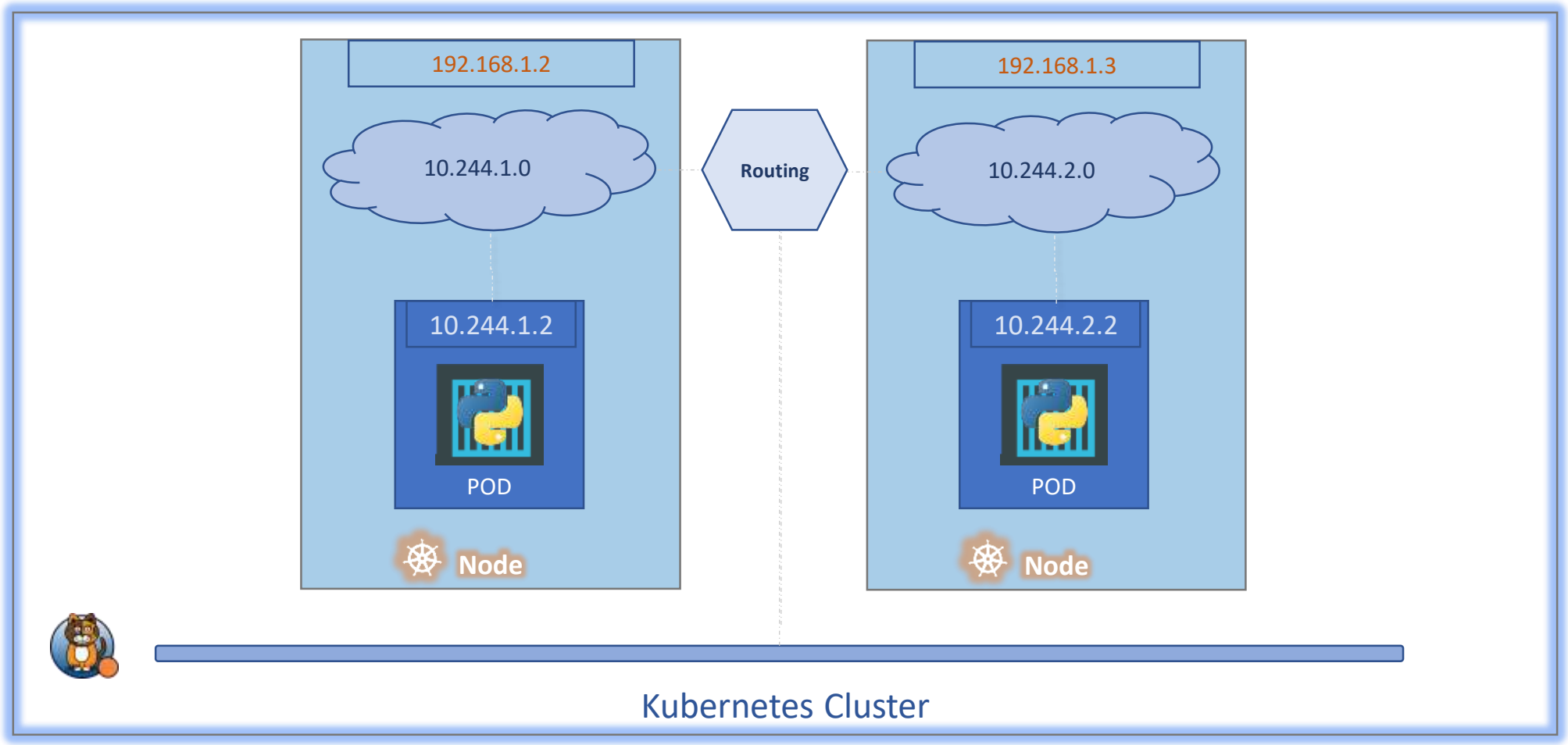
Refer to the Calico documentation for a kubeadm quickstart, a kubeadm installation guide, and other resources.

Note:

- In order for Network Policy to work correctly, you need to pass `--pod-network-cidr=192.168.0.0/16` to `kubeadm init`.
- Calico works on `amd64` only.

```
kubectl apply -f https://docs.projectcalico.org/v3.0/getting-started/kubernetes/installation/hosted/kubeadm/1.7/calico.yaml
```

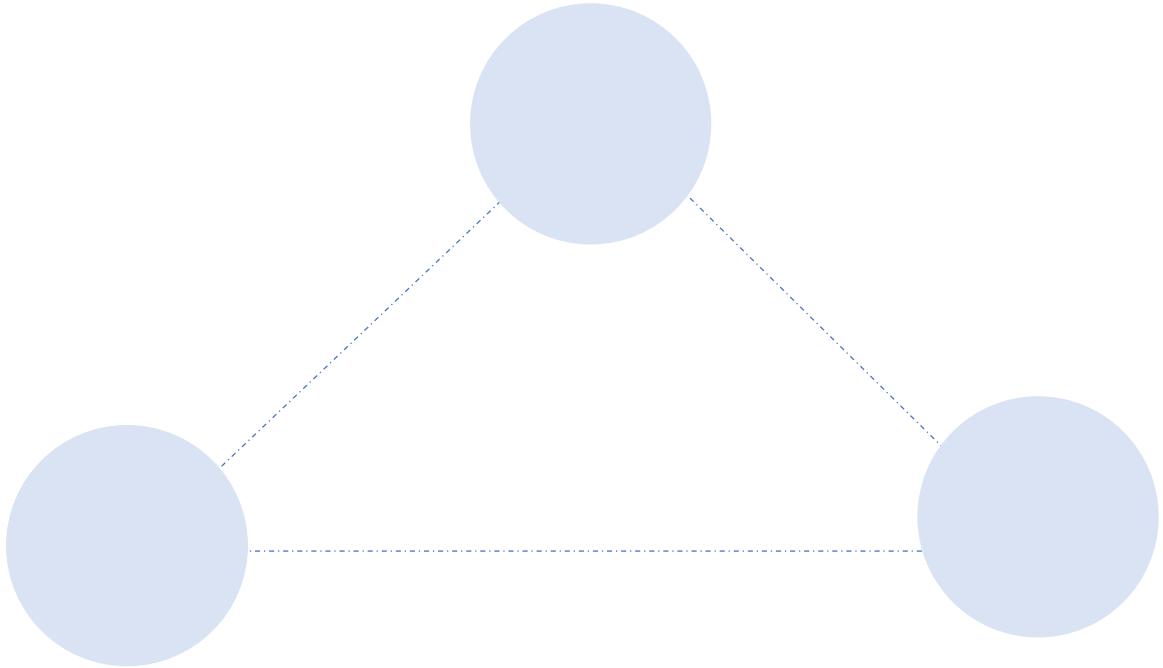
Cluster Networking



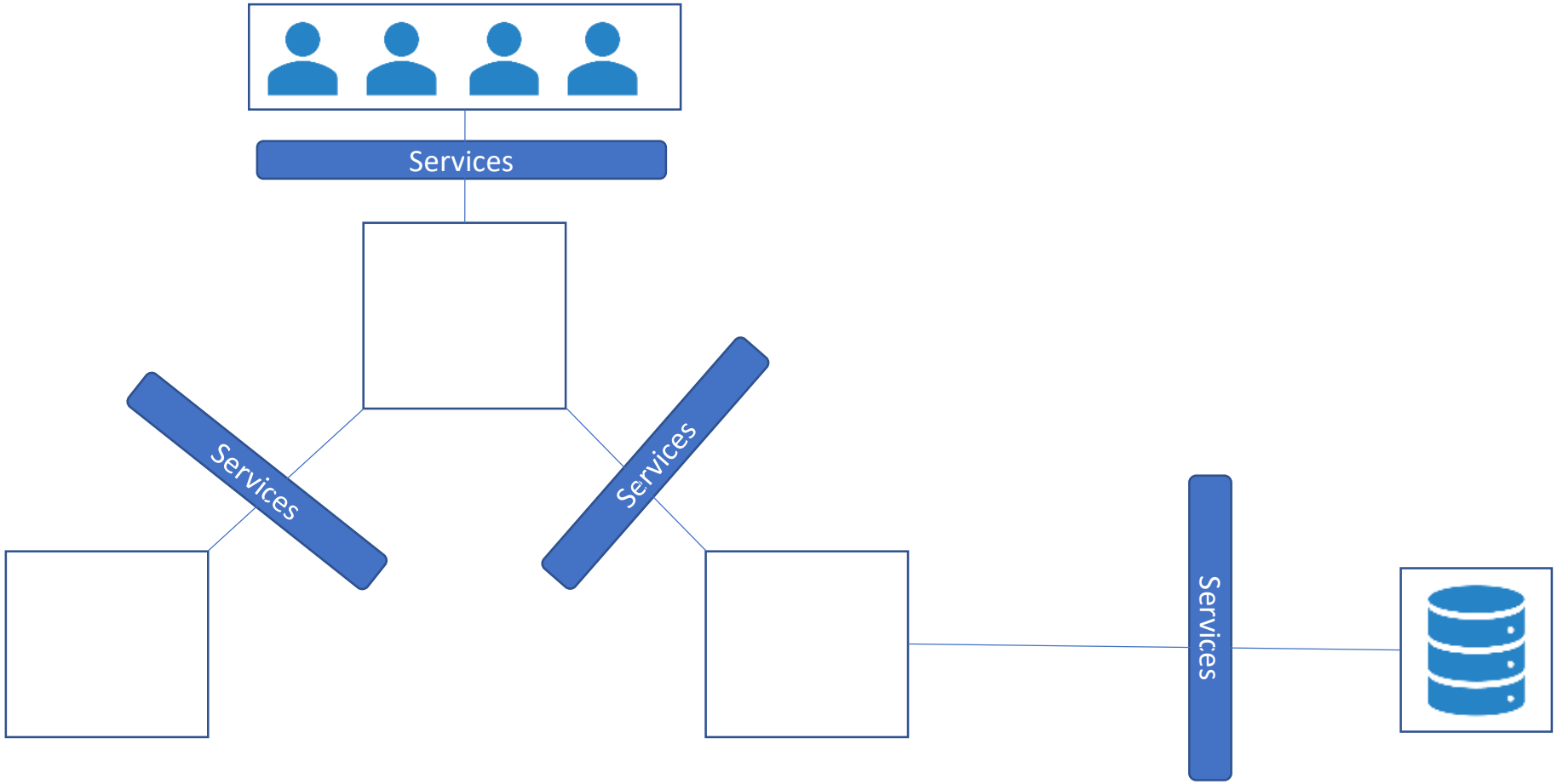
Demo

Networking

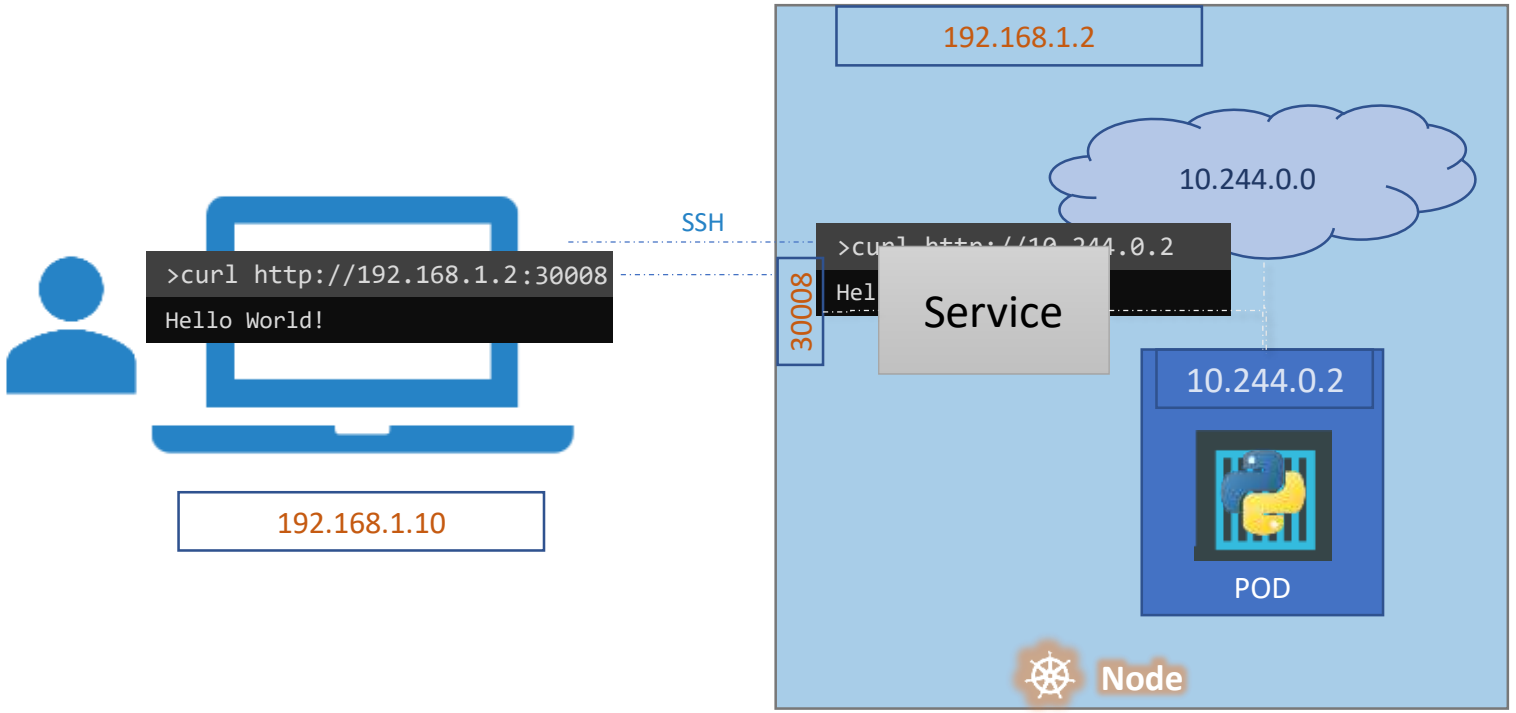
Services



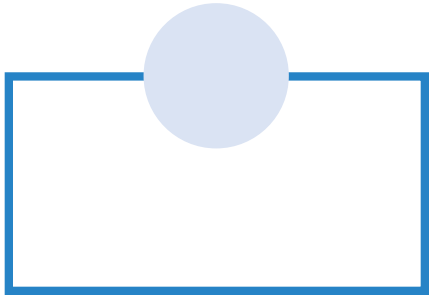
Services



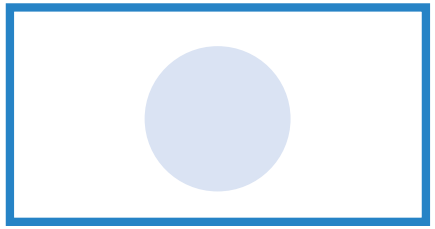
Service


>curl http://192.168.1.2:30008
Hello World!
192.168.1.10
SSH
192.168.1.2
10.244.0.0
30008
Service
10.244.0.2
POD
Node

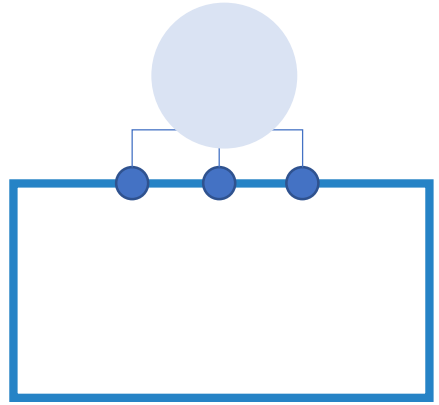
Services Types



NodePort

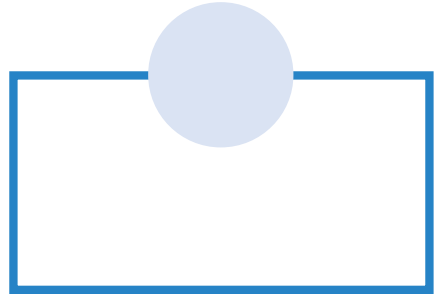


ClusterIP

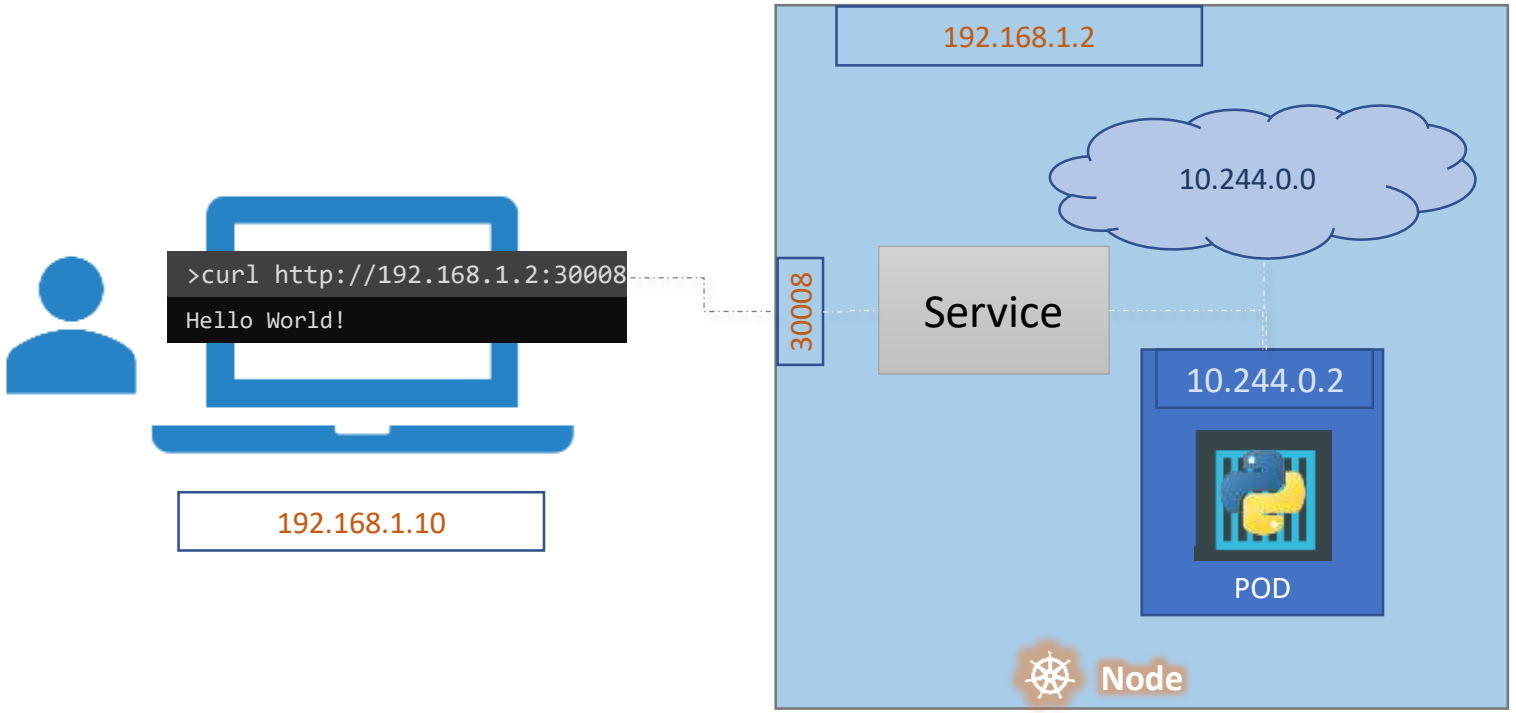


LoadBalancer

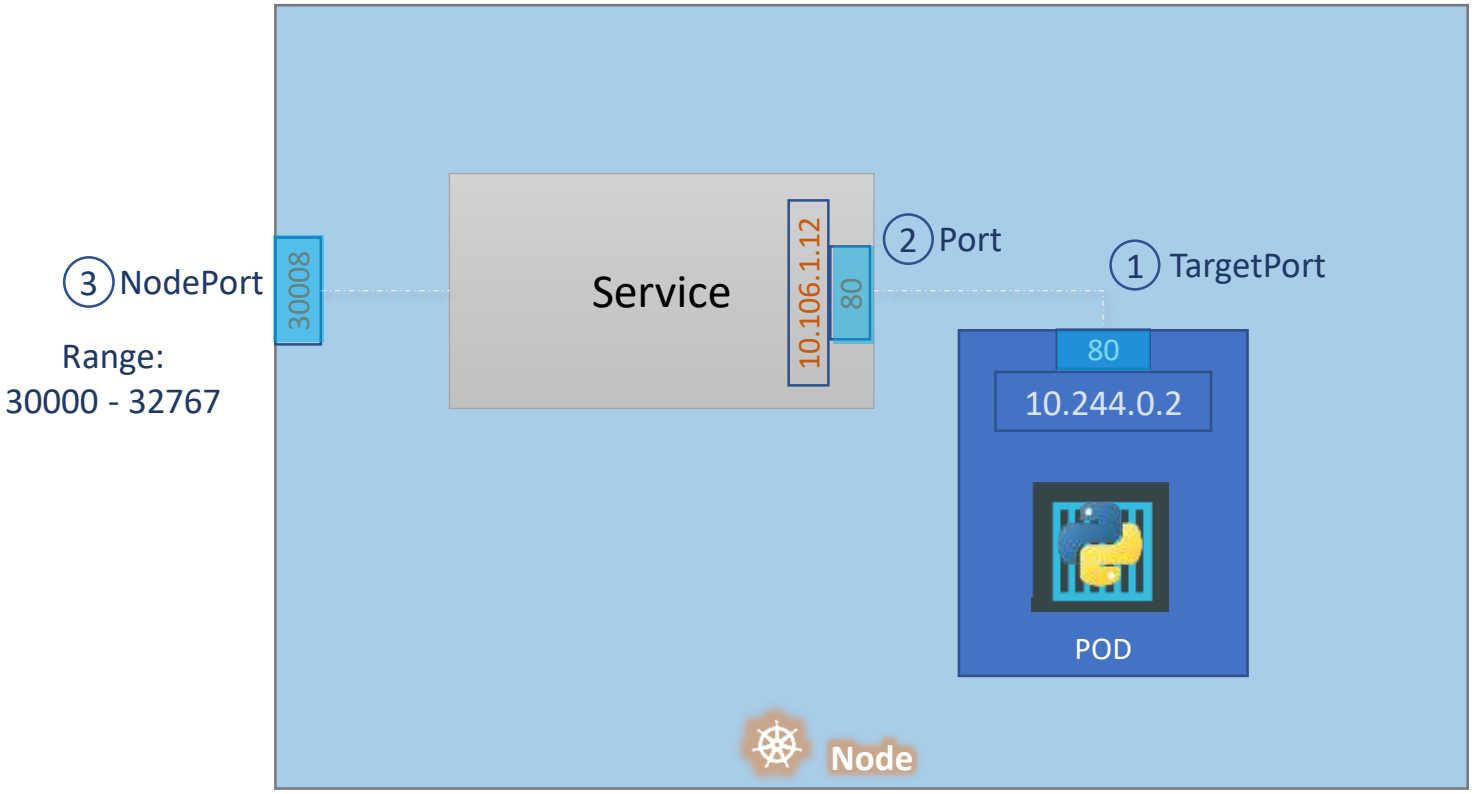
NodePort



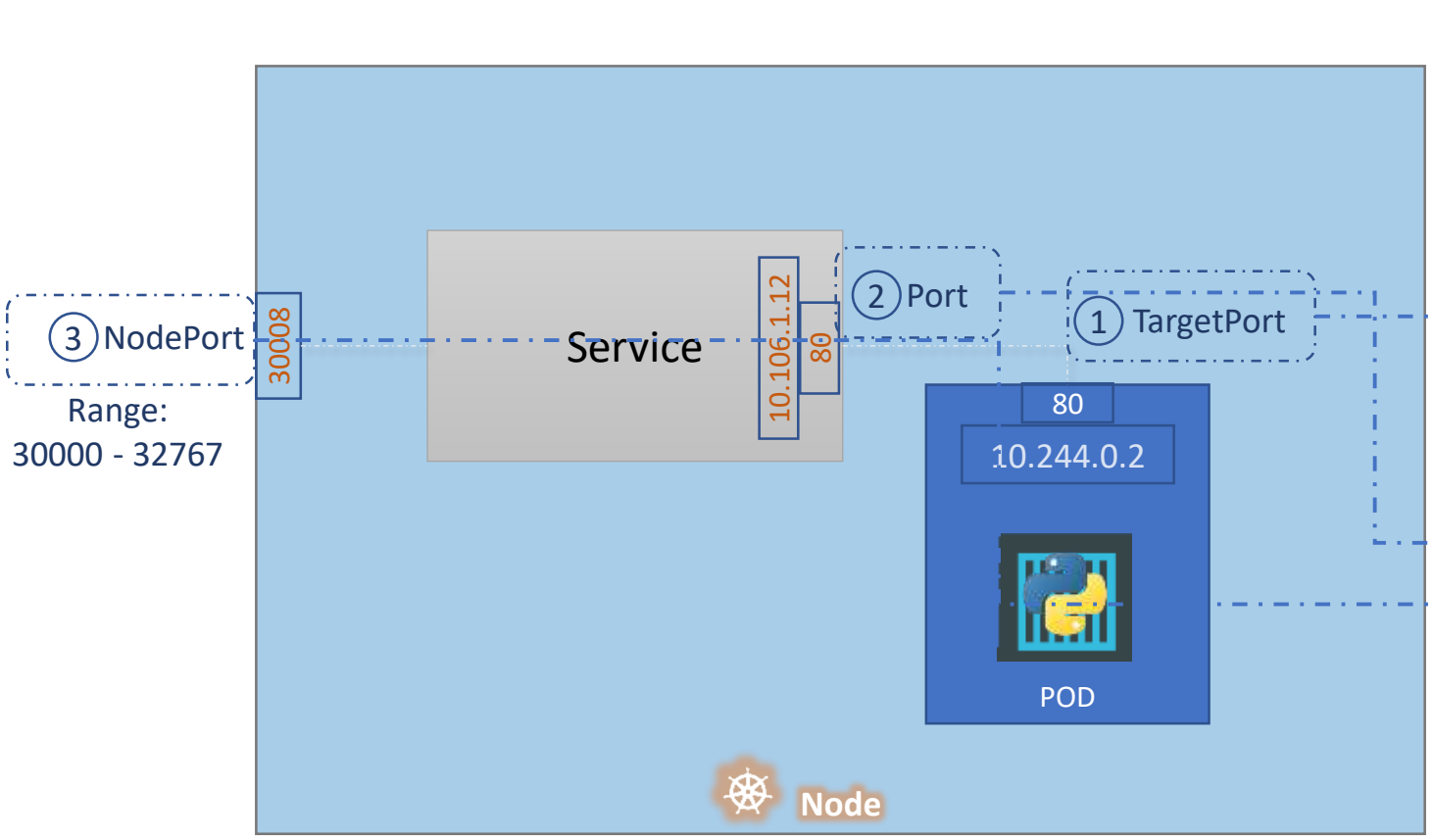
Service - NodePort



Service - NodePort



Service - NodePort



service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
      - targetPort: 80
        port: 80
        nodePort: 30008
```

Service - NodePort

service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
     - targetPort: 80
       port: 80
       nodePort: 30008
    selector:
```

pod-definition.yml

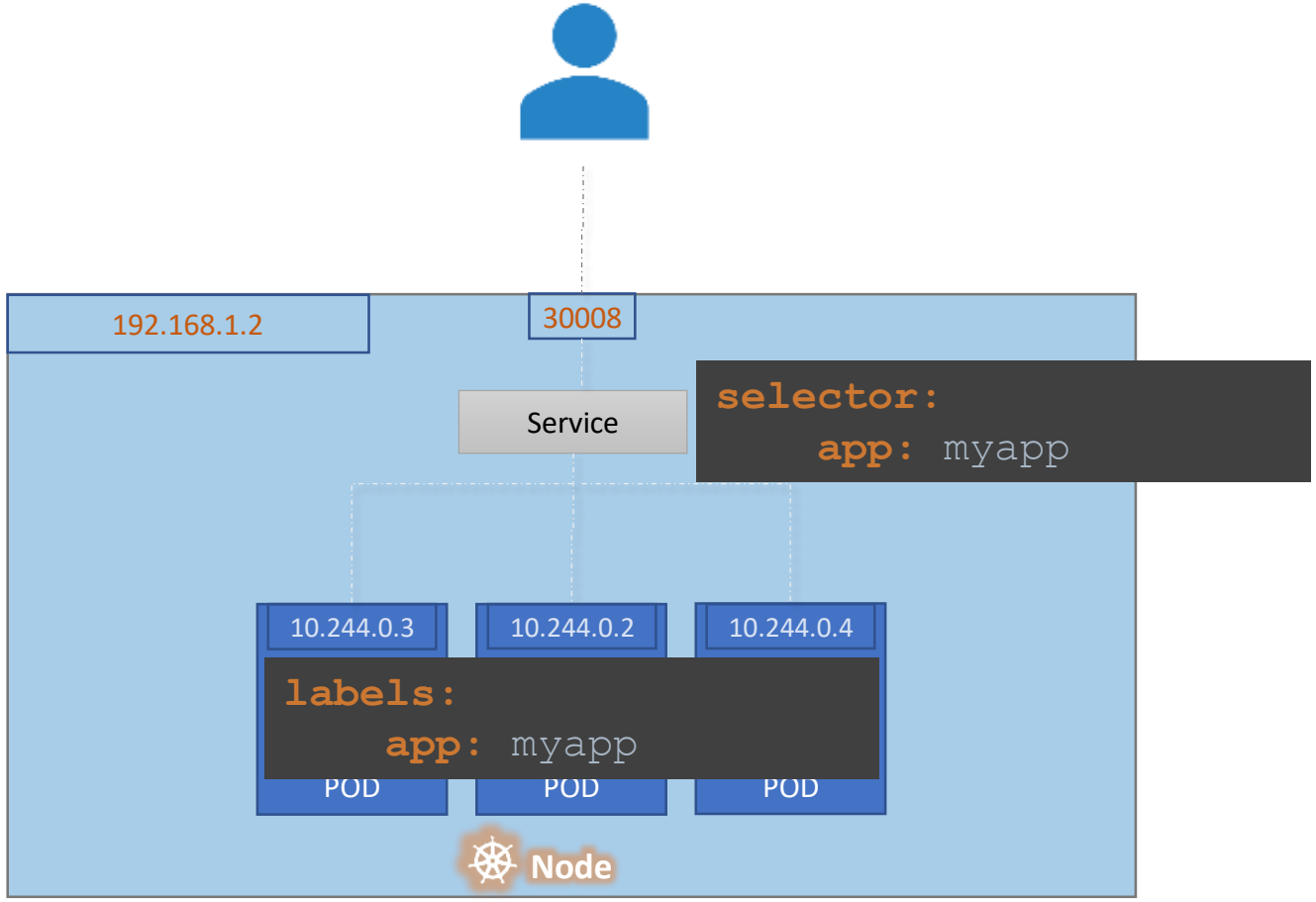
```
> kubectl create -f service-definition.yml
service "myapp-service" created

> kubectl get services
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        16d
myapp-service   NodePort    10.106.127.123   <none>        80:30008/TCP   5m
```

```yaml
      app: myapp
```

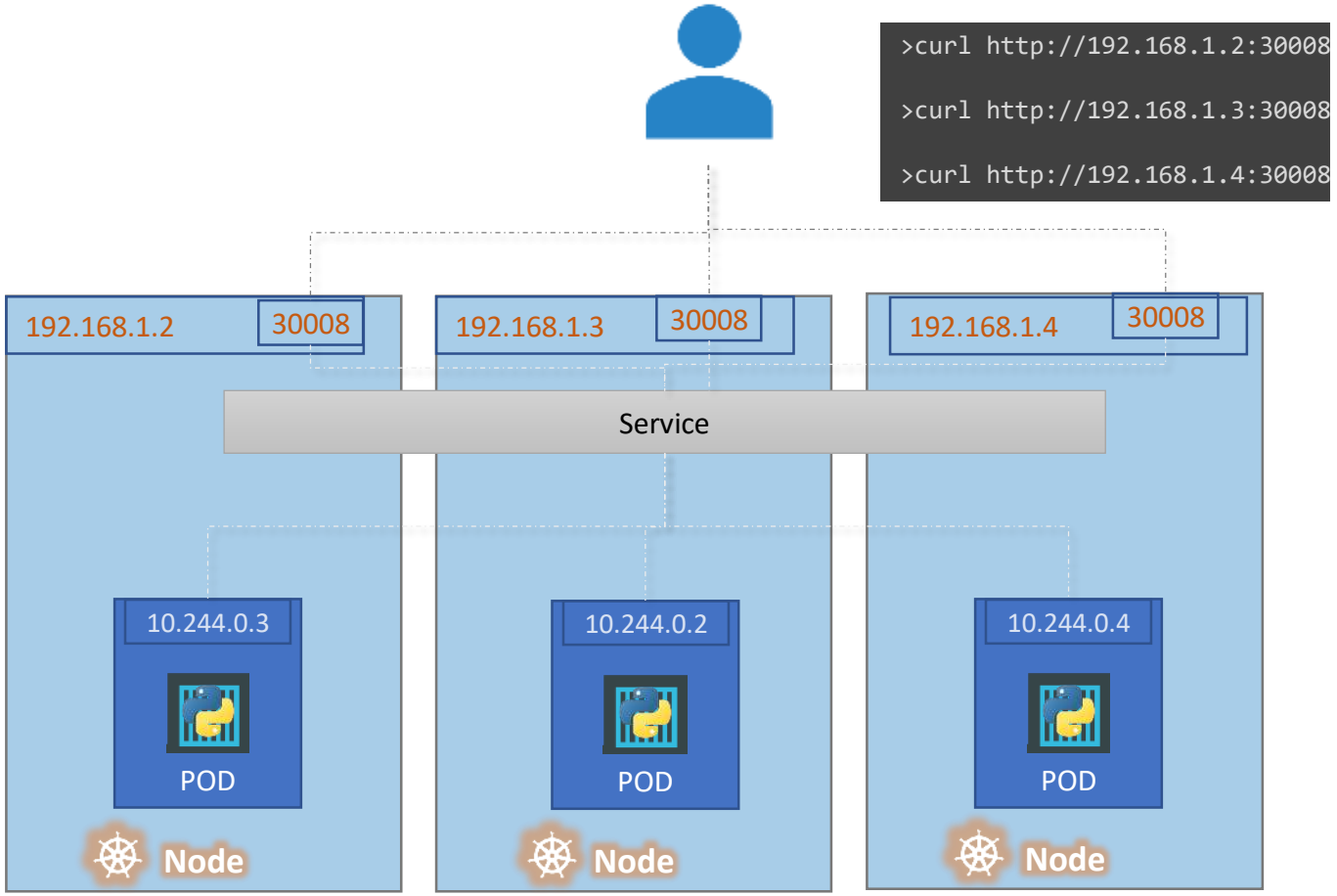
```
> curl http://192.168.1.2:30008
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
```

Service - NodePort

Algorithm: Random
SessionAffinity: Yes



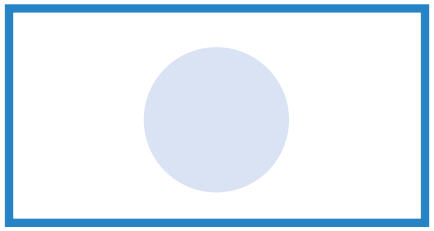
Service - NodePort



Demo

Service - NodePort

ClusterIP



ClusterIP

front-end

back-end

redis