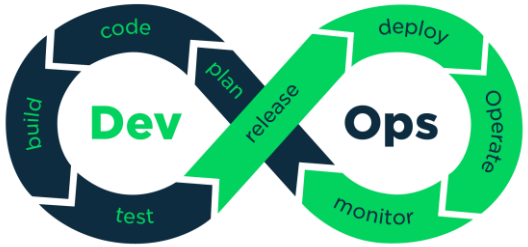
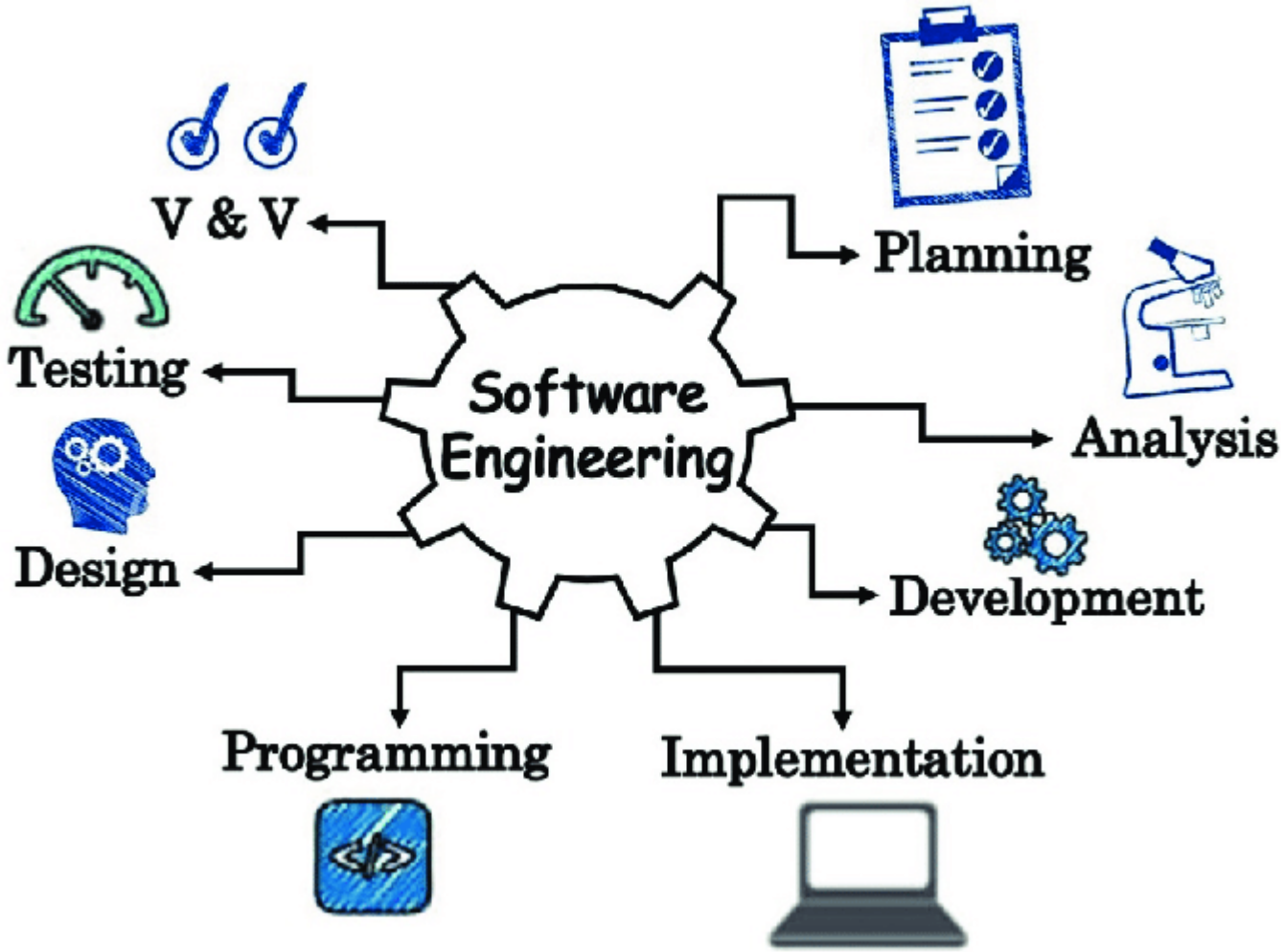
Devops– Let the Journey Begin



Sandeep Kumar

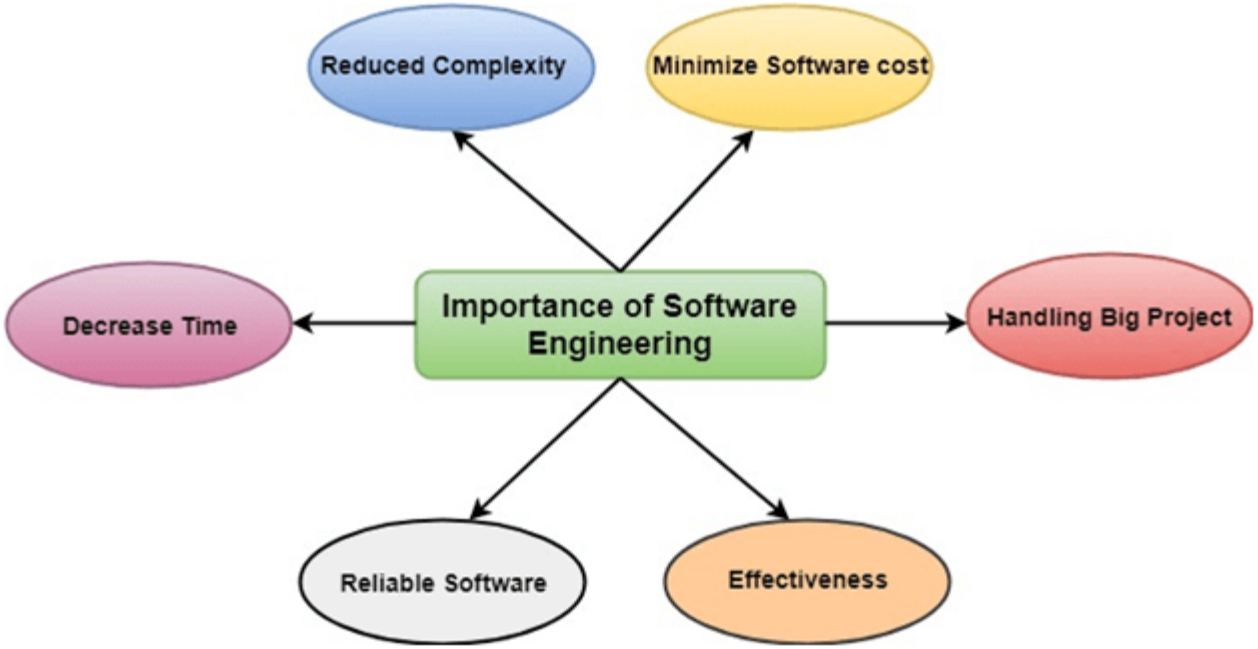
Software Engineering



Software Engineering

- The term **software engineering** is the product of two words, **software**, and **engineering**.
- Software Engineering is the process of designing, developing, testing, and maintaining software. It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.

Importance of Software Engineering

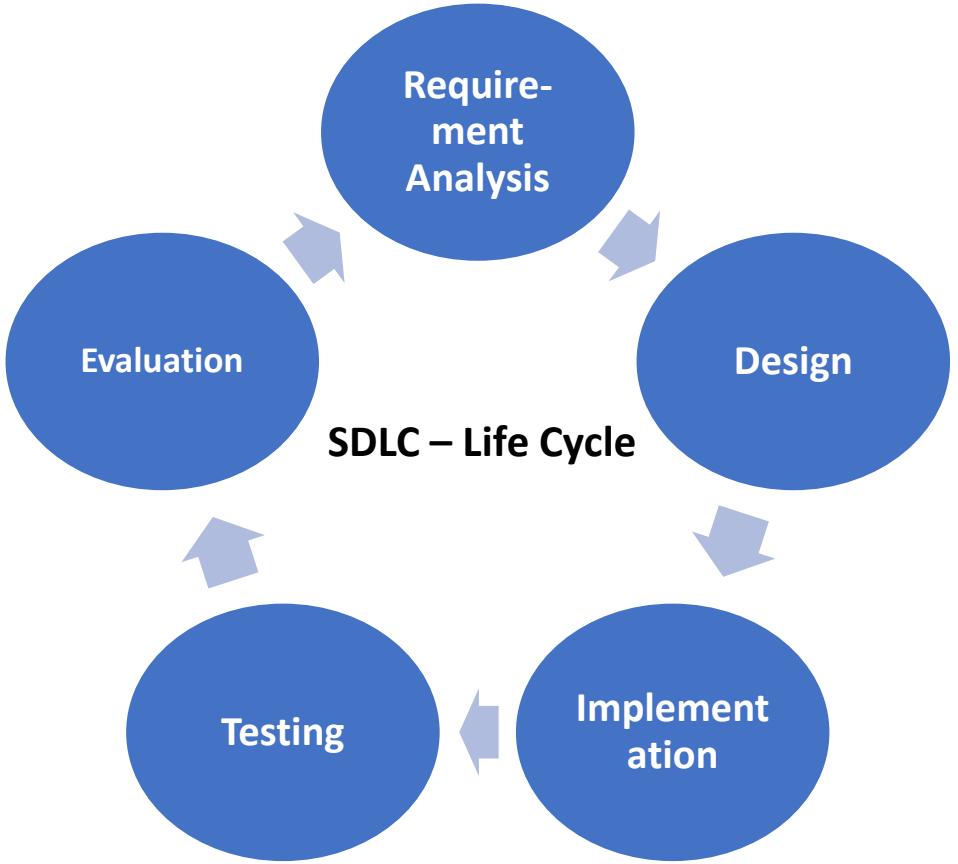


SDLC Model

- The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. in detail, the SDLC methodology focuses on the following phases of software development:

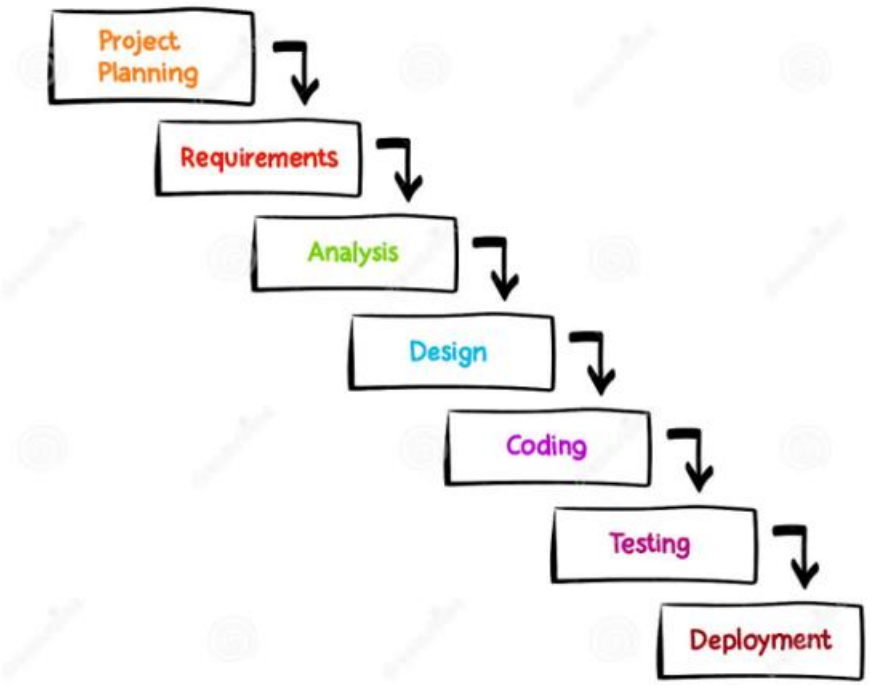
SDLC Model

- The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. in detail, the SDLC methodology focuses on the following phases of software development:

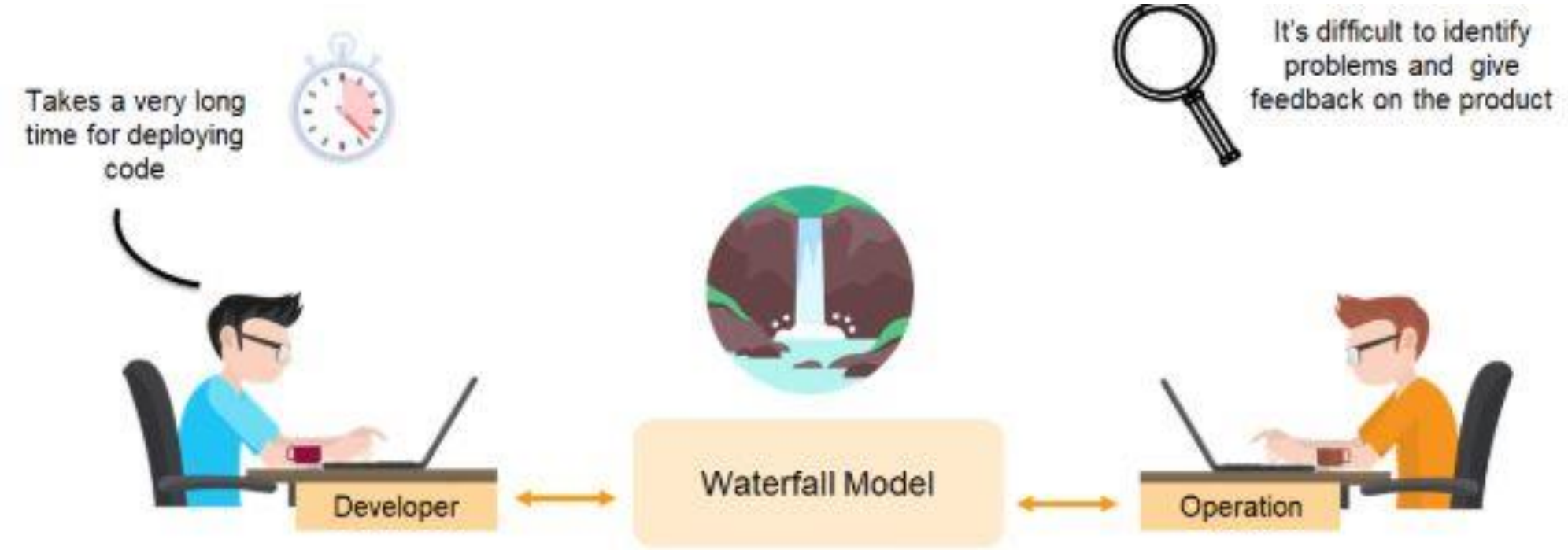


Waterfall Model

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

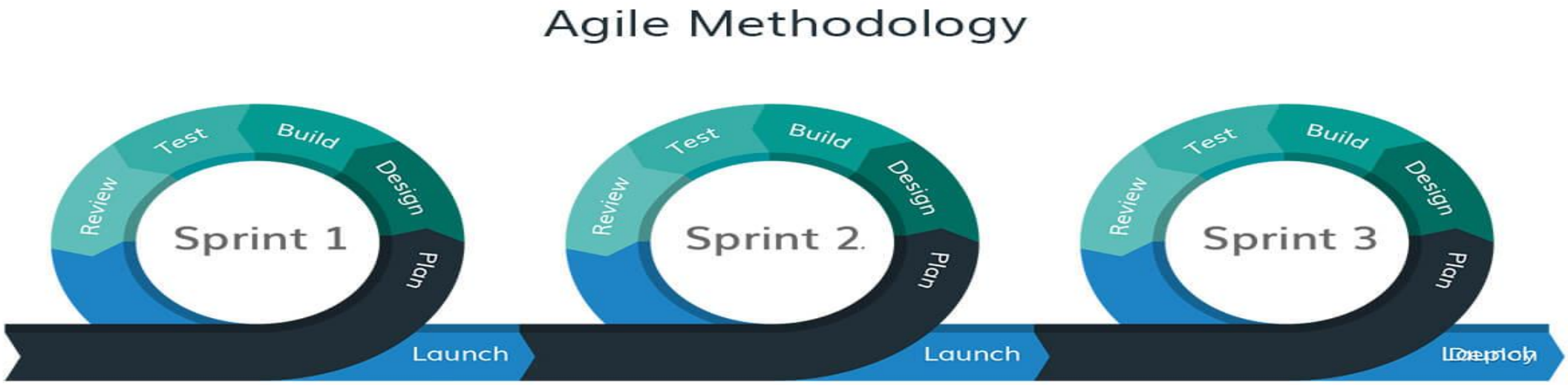


Waterfall Model



Agile

The Agile methodology is a way to manage a project by breaking it up into several phases. It involves constant collaboration with stakeholders and continuous improvement at every stage. Once the work begins, teams cycle through a process of planning, executing, and evaluating. Continuous collaboration is vital, both with team members and project stakeholders



Agile Drawback

- Lack of Documentation
- Inflexibility
- Dependence on team collaboration
- Customer involvement
- Risk management

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.

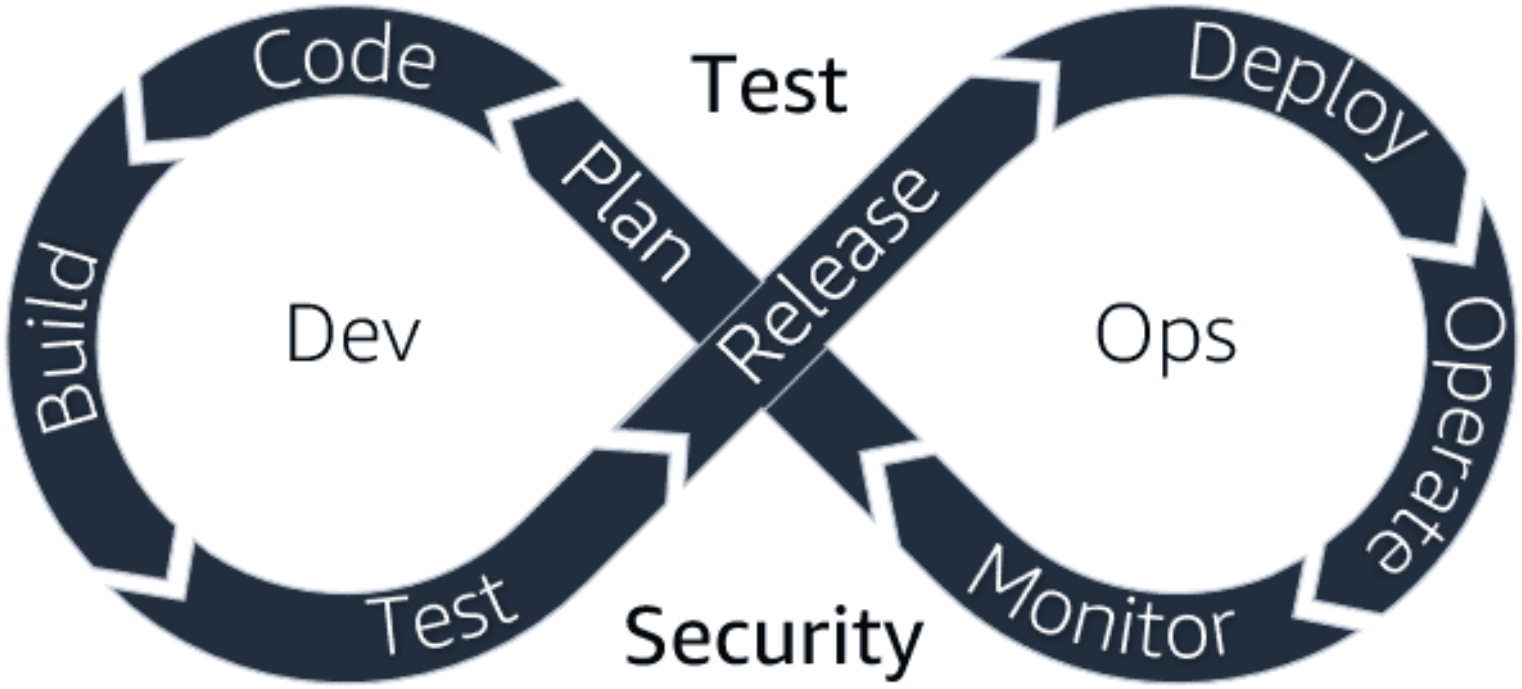
DevOps

Why DevOps? Why not other methods?

What is DevOps

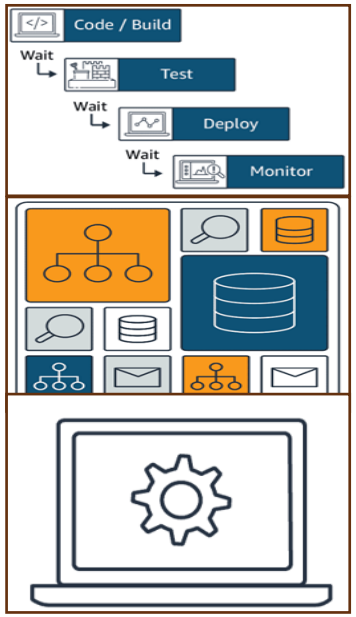
1. DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes.
2. This speed enables organizations to better serve their customers and compete more effectively in the market.
3. DevOps emphasizes better collaboration and efficiencies so teams can innovate faster and deliver higher value to businesses and customers.
4. DevOps is short for Development (Dev) and Operations (Ops). Dev are the people and processes that create software. Ops are the teams and processes that deliver and monitor the software.

What is DevOps



An infinity loop formed by the software lifecycle stages is a common depiction of the ongoing collaboration and efficiencies suggested by DevOps.

Problems with Traditional Development Practices

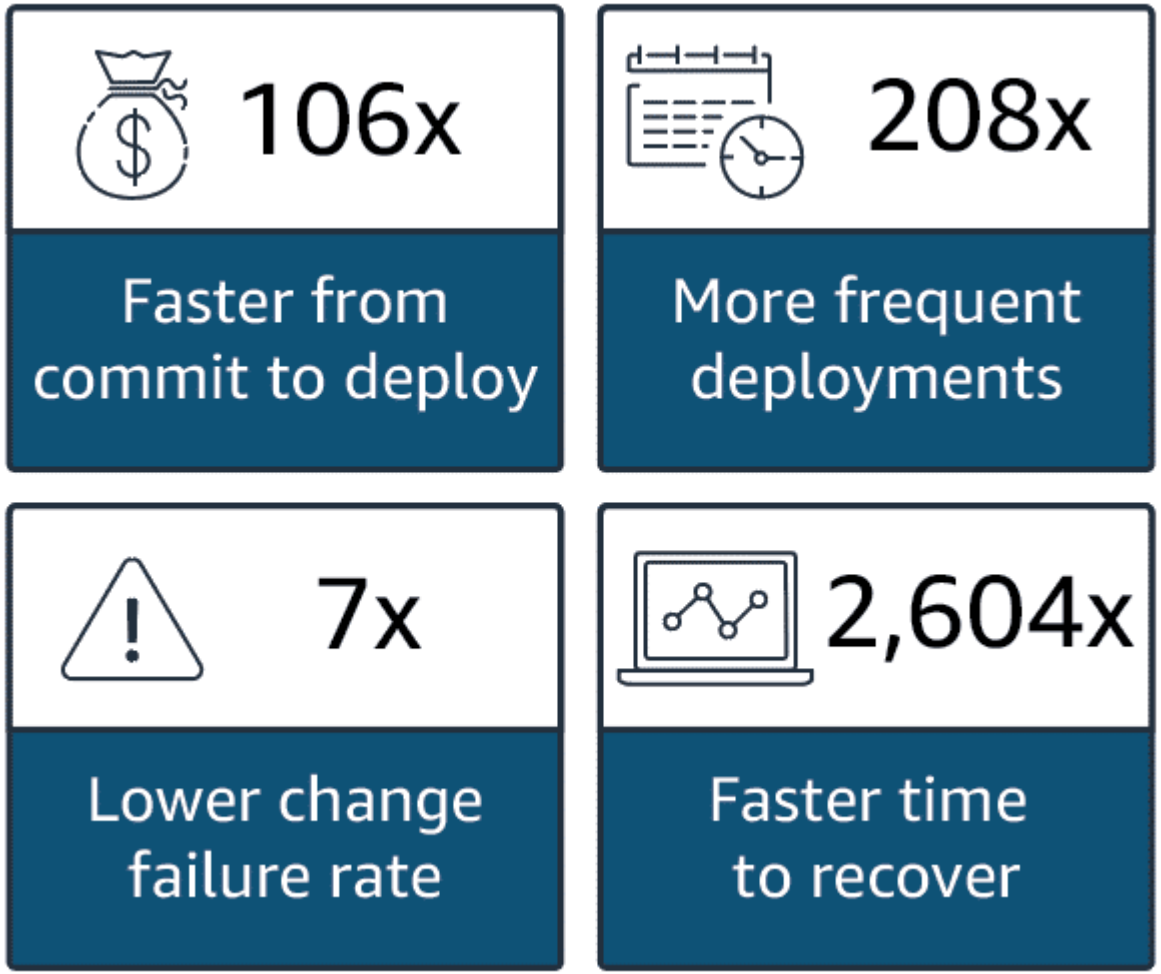


Traditional ways of developing software have proven slow and inefficient, and fail to support teams' efforts to quickly deliver stable, high-quality applications.

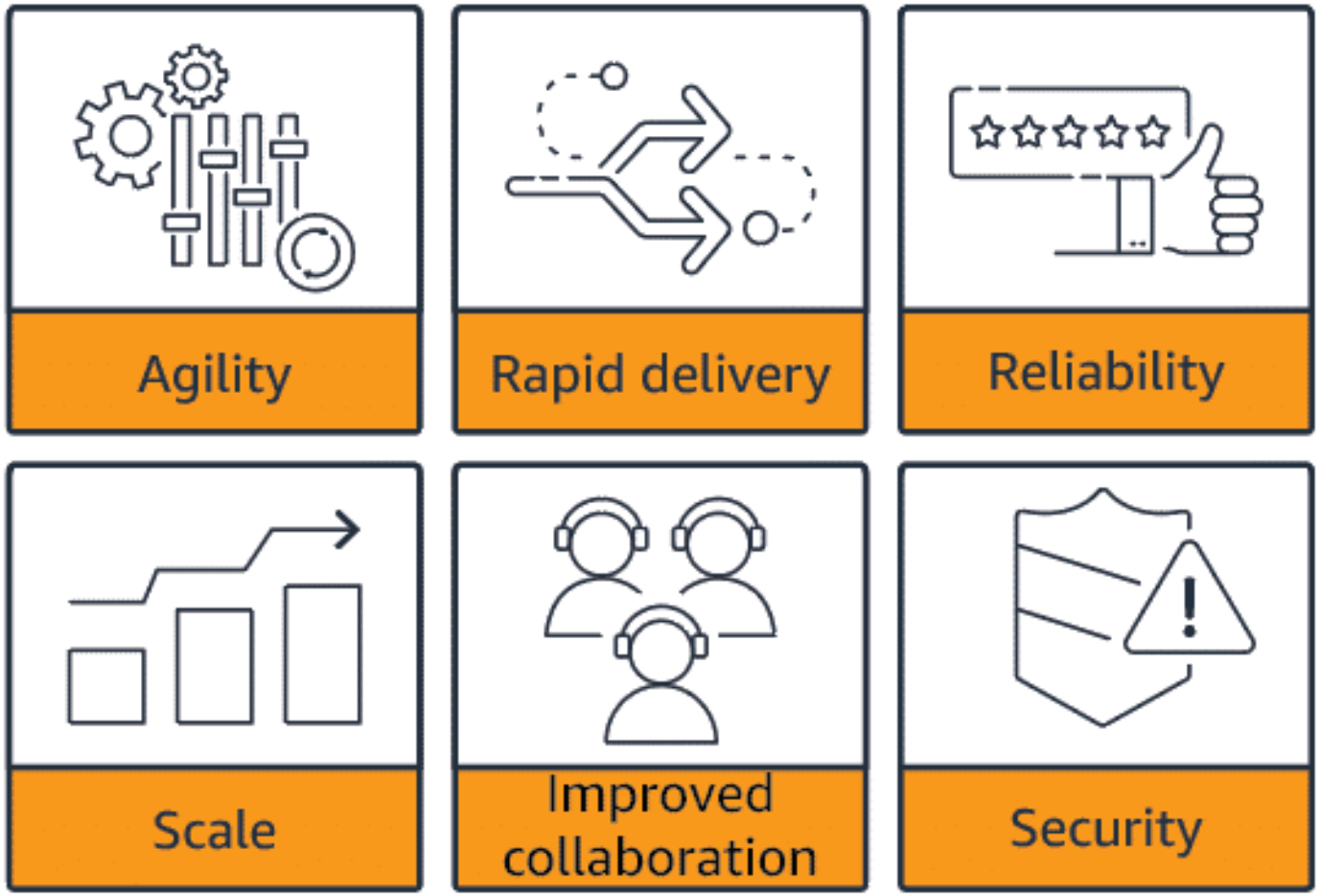
Monolithic applications are hard to update and deploy

Manual processes throughout the application lifecycle are slow, inconsistent, and error-prone.

Why DevOps?


106x
Faster from commit to deploy
208x
More frequent deployments
7x
Lower change failure rate
2,604x
Faster time to recover

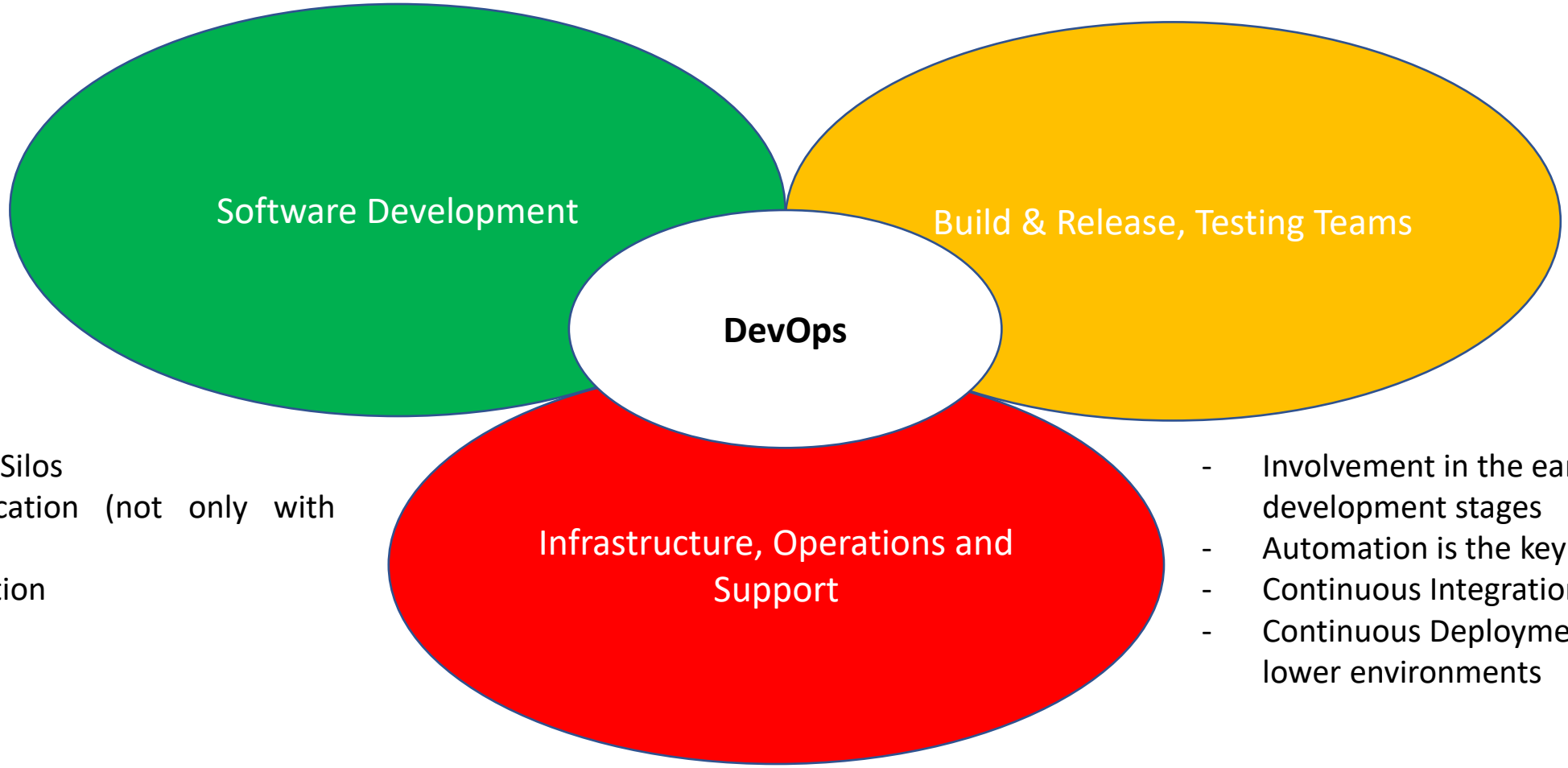
The benefits of DevOps



DevOps Culture

There are seven core principles that can help you achieve a DevOps culture.

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

DevOps Model

Collaboration

Communication

Continuity

Automation

Test

Build

Deploy

Project Management

Customer

Planning

Monitoring

Continuous Feedback

Continuous Improvement

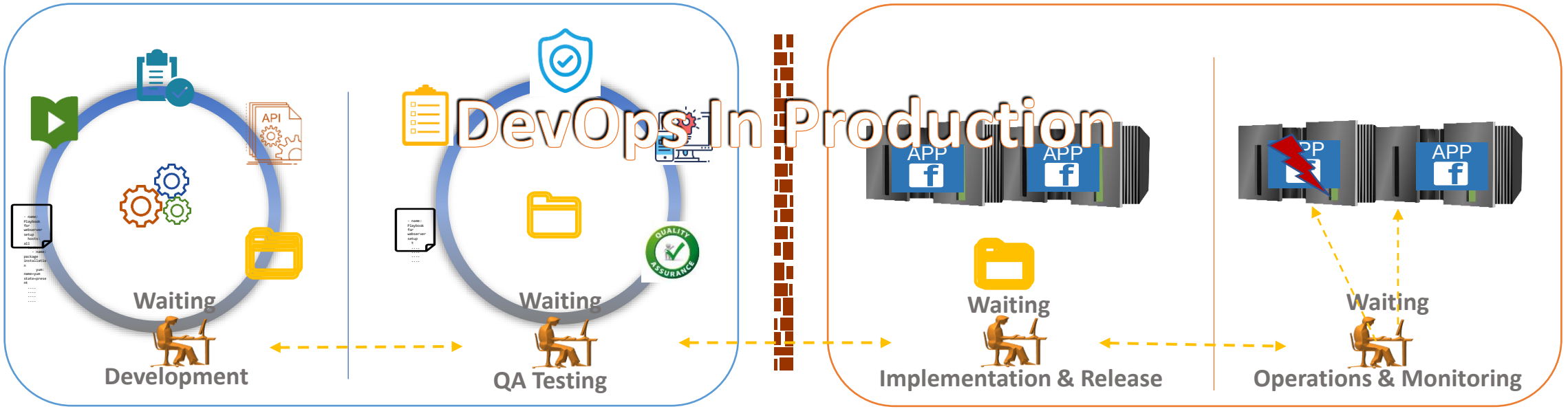
Continuous Planning

Continuous Integration

Continuous Delivery

Continuous Deployment

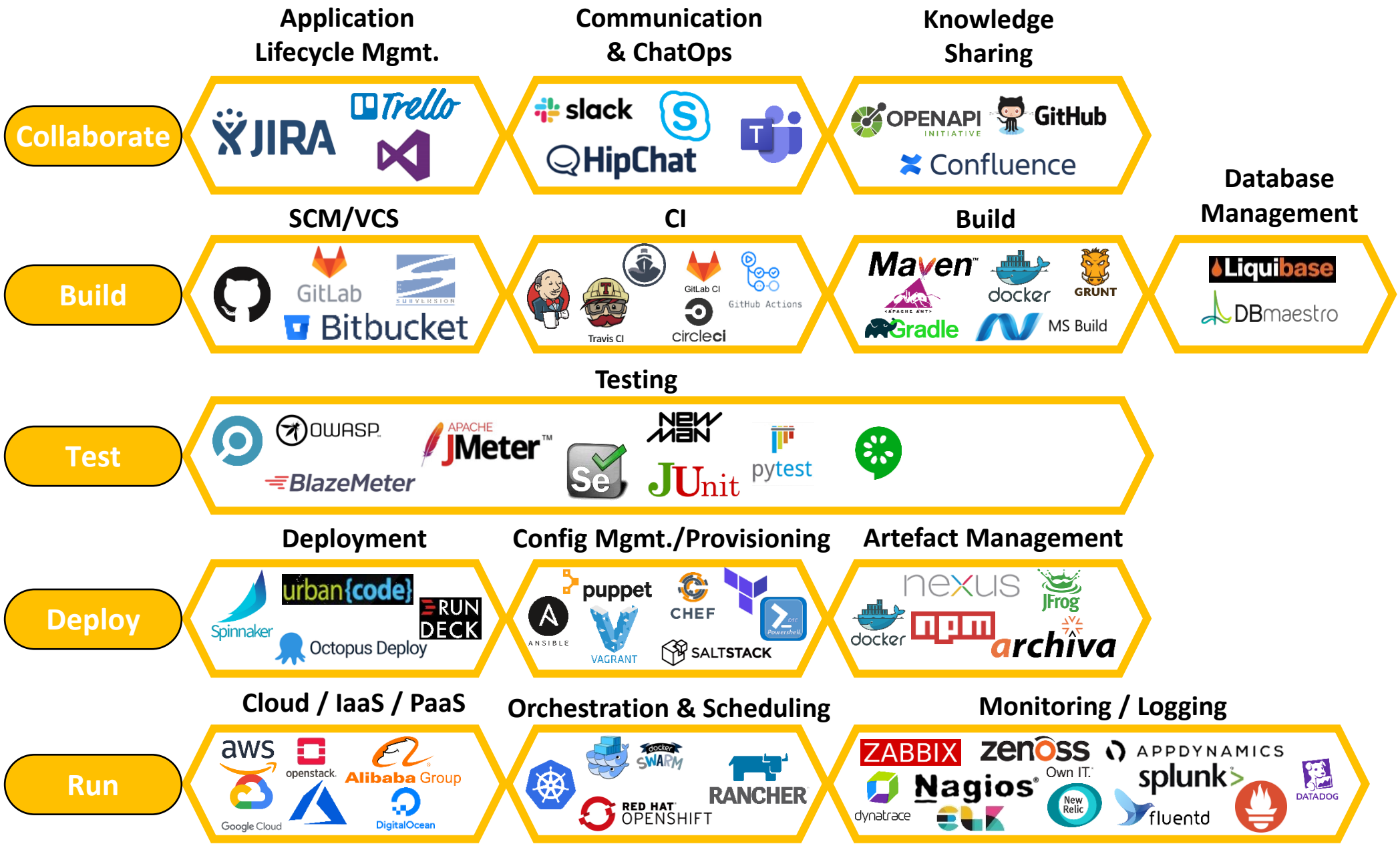
Continuous Monitoring



Development

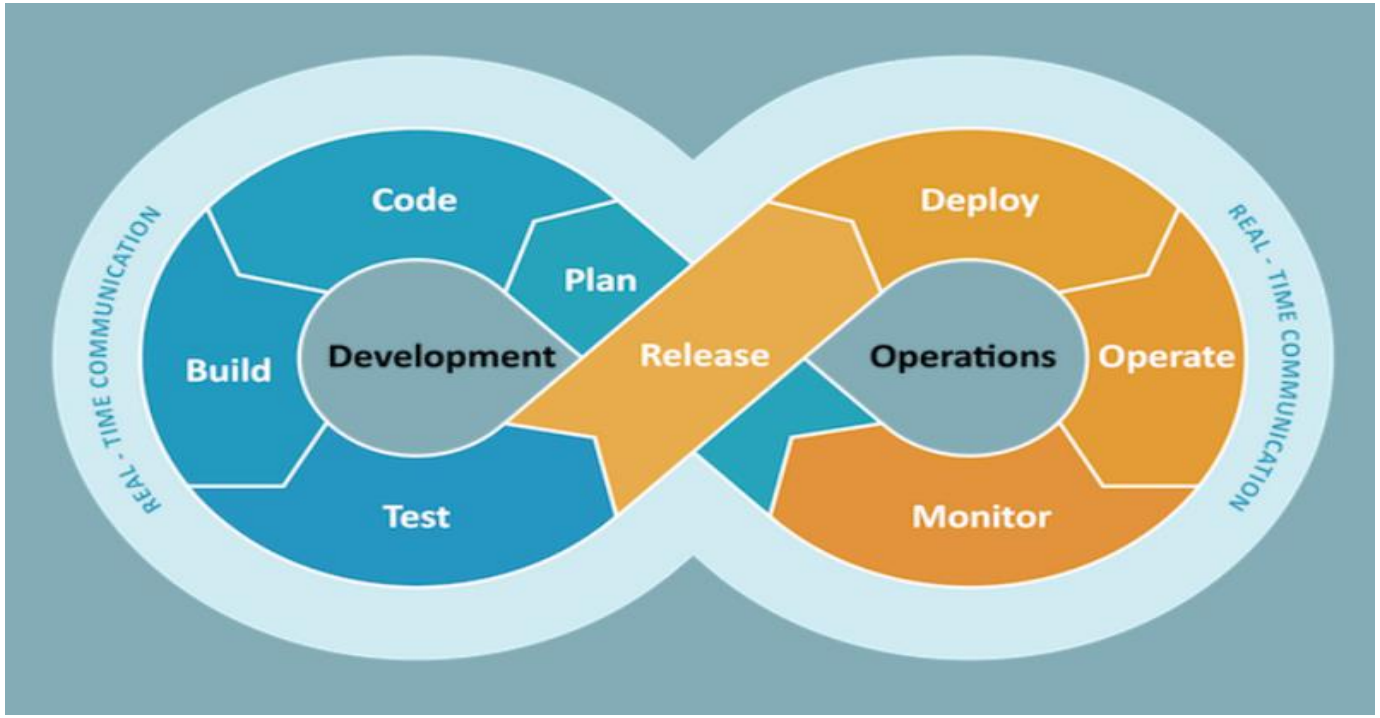
Operations

DevOps Toolsets



DevOps Lifecycle

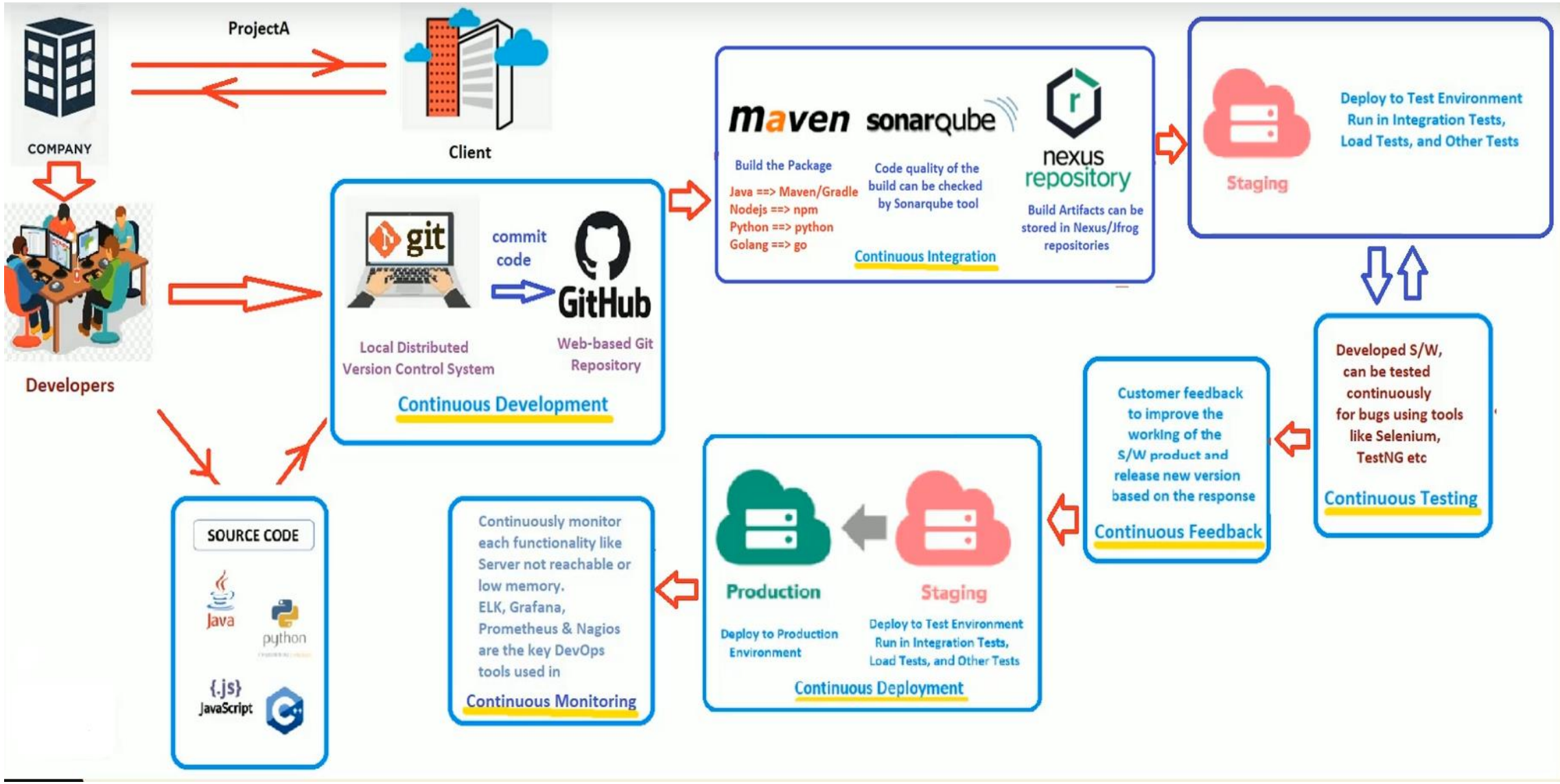
1. Planning
2. Building
3. Testing and Deploying
4. Delivering
5. Monitoring and Logging
6. Gathering



DevOps Lifecycle

1. Continuous Development
2. Continuous Integration
3. Continuous Testing
4. Continuous Monitoring
5. Continuous Feedback
6. Continuous Deployment
7. Continuous Operations

DevOps Lifecycle

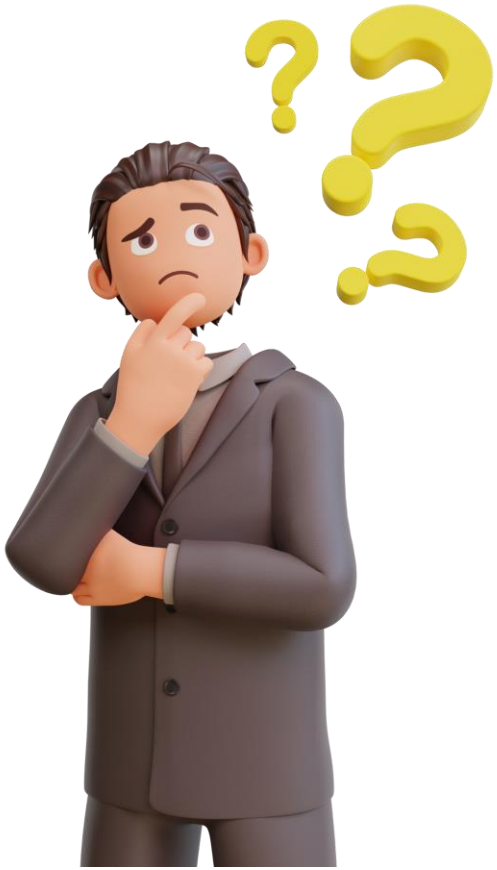


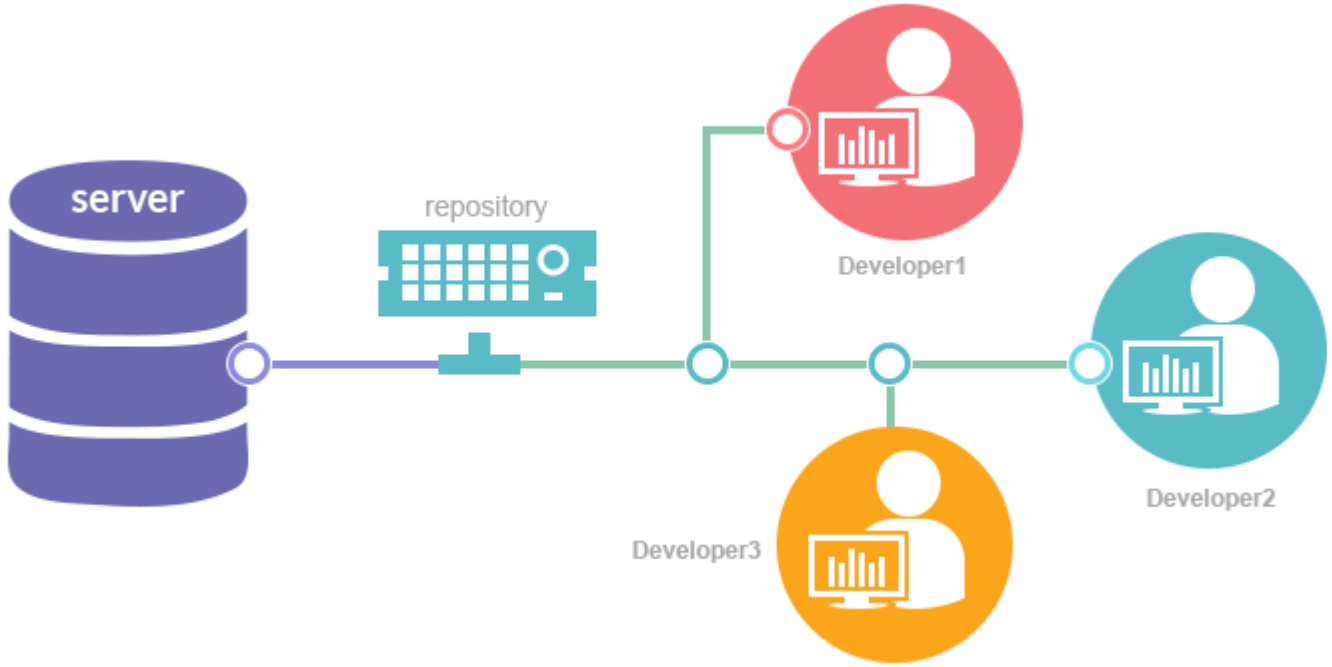
Git– Let the Journey Begin

Sandeep Kumar

Version Control Systems with GIT

What is The Need of Version Control System





What is Version Control System

- Version control (or revision control) is a system that records changes to a file or a group of files and directories over time, so that you can review or go back to specific versions later.
- Quite literally, version control means maintaining versions of your work.
- You may like to think of version control as a tool that takes snapshots of your work across time, creating checkpoints. You can return to those checkpoints any time you want.
- Not only are the changes recorded in these checkpoints, but also information about who made the changes, when they made them, and the reasons behind the changes

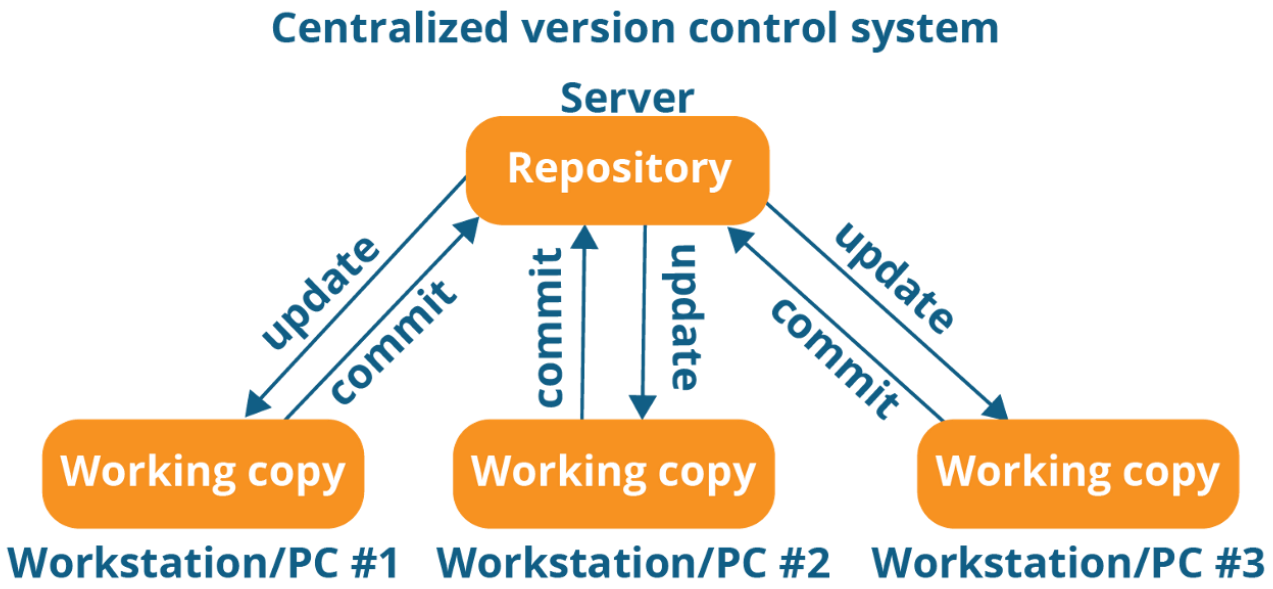
Types of Version Control System

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

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

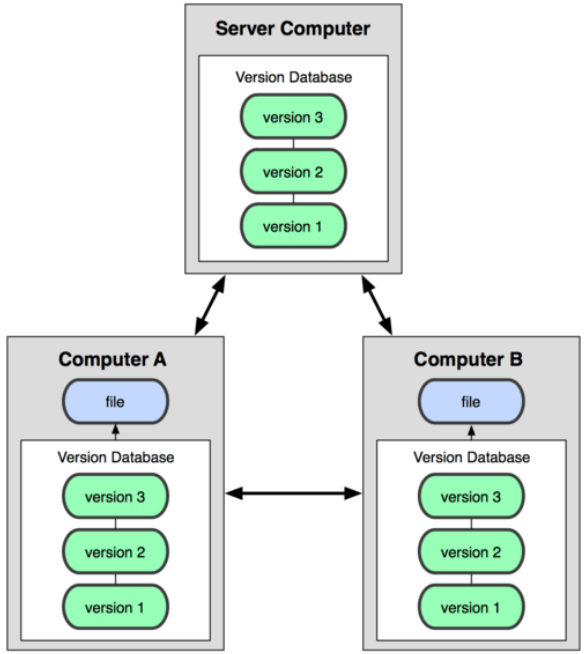
Centralized version control systems (CVCS)

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.



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.

TOP VERSION CONTROL SYSTEMS

Introduction of Git

Git History

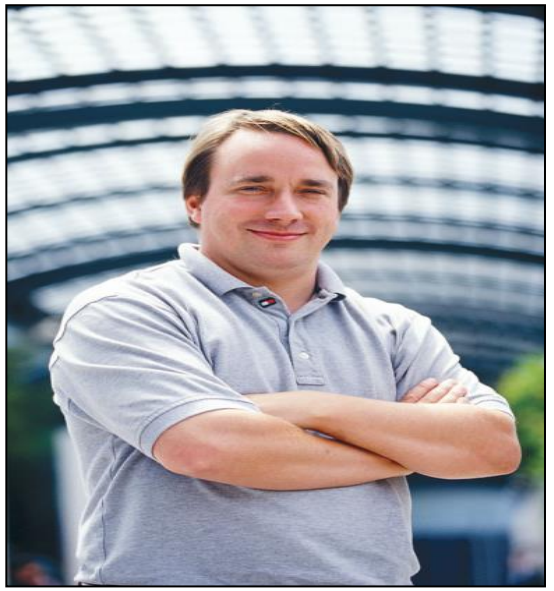
Linus uses BitKeeper to manage Linux code

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



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.

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.

Features of Git

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development

Git repository

A Git repository is the directory where all of your project files and the related metadata resides.

Git records the current state of the project by creating a tree graph from the index(staging area).

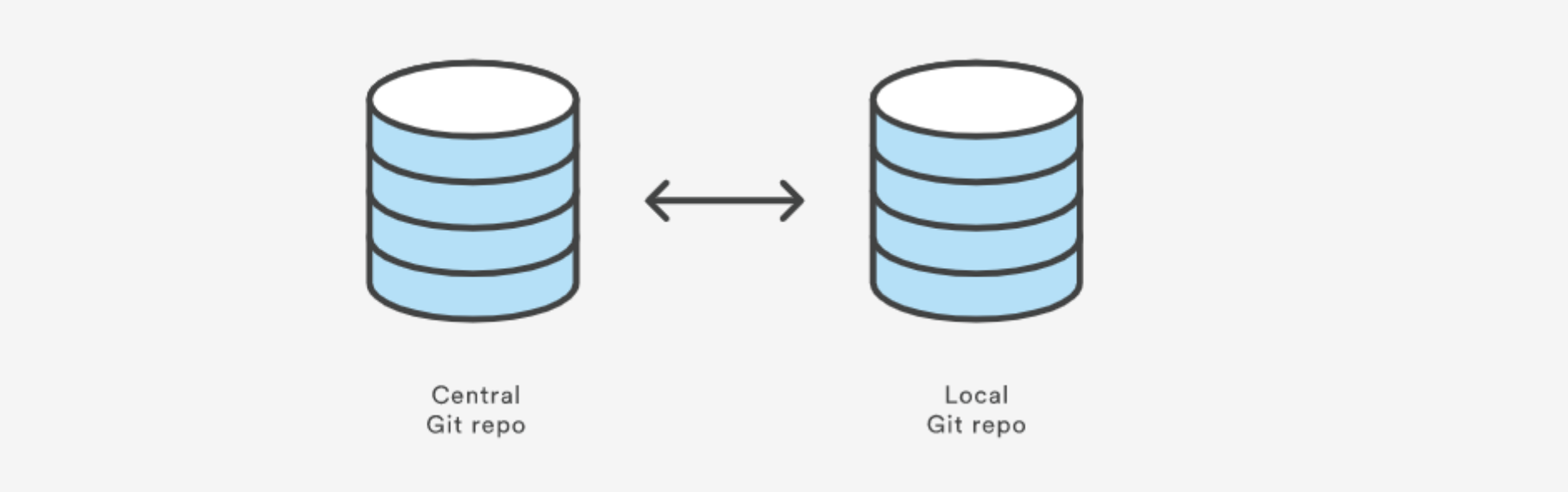
It uses a data structure called the Directed Acyclic Graph (DAG) for managing different commits(each commit is a node in this DAG).

Local Git repository

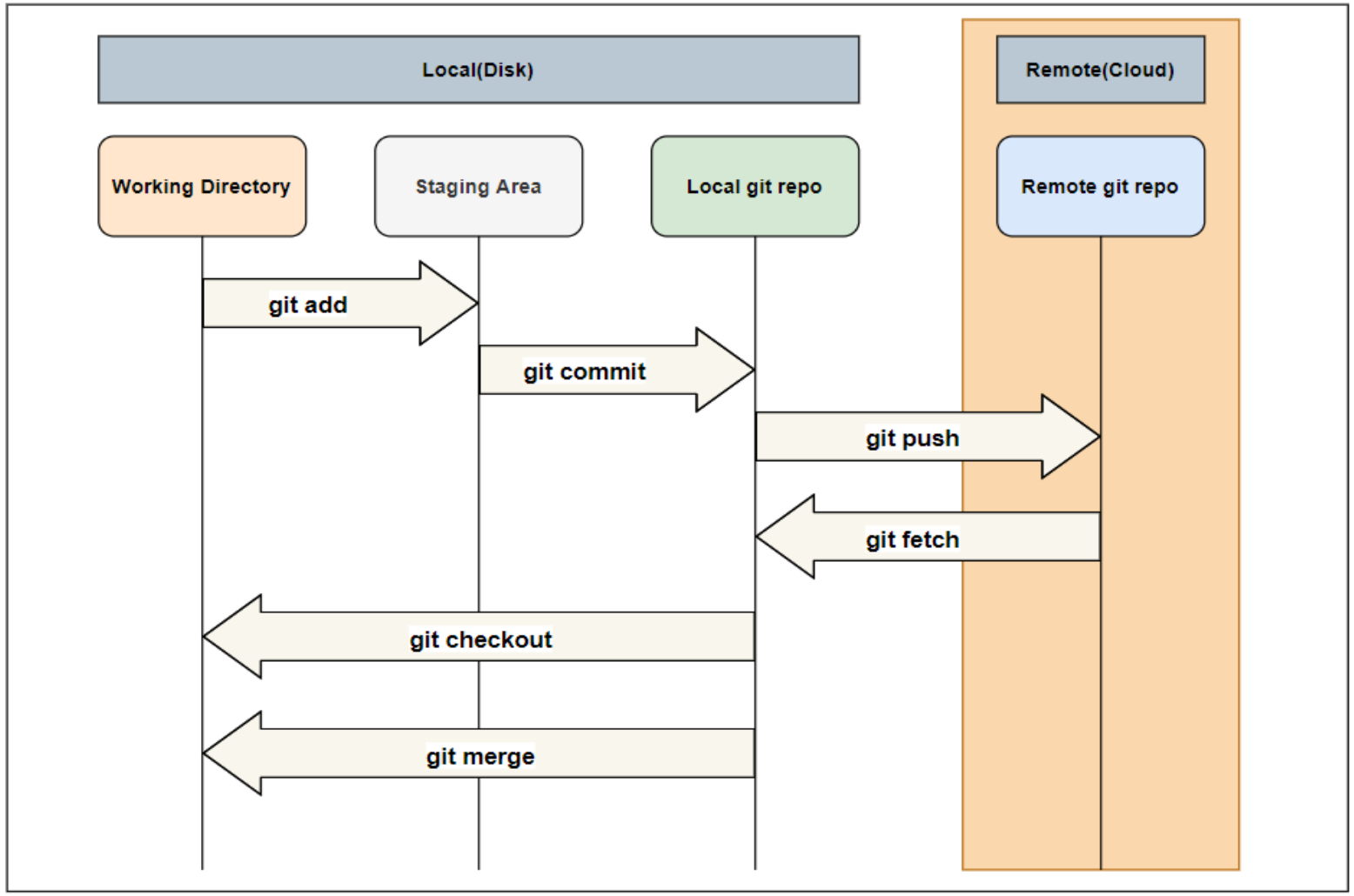
A local Git repository is a storage of your project files and metadata on your computer’s disk. It allows you to save versions of your code, which you can access whenever needed.

Remote Git repository

A remote Git repository is a virtual storage of your project on git hosting servers like GitHub or Bitbucket. It has the same copy of your project as you have locally.



Basic Flow of Commands Between Local and Remote Repositories



GIT Architecture

The Working Tree

The Working Tree is the area where you are currently working. It is where your files live. This area is also known as the “untracked” area of git. Any changes to files will be marked and seen in the Working Tree.

The Staging Area (Index):

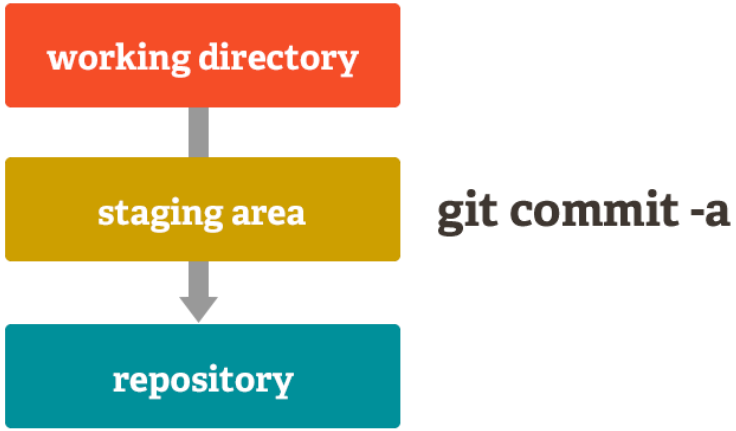
The Staging Area is when git starts tracking and saving changes that occur in files. These saved changes reflect in the .git directory.

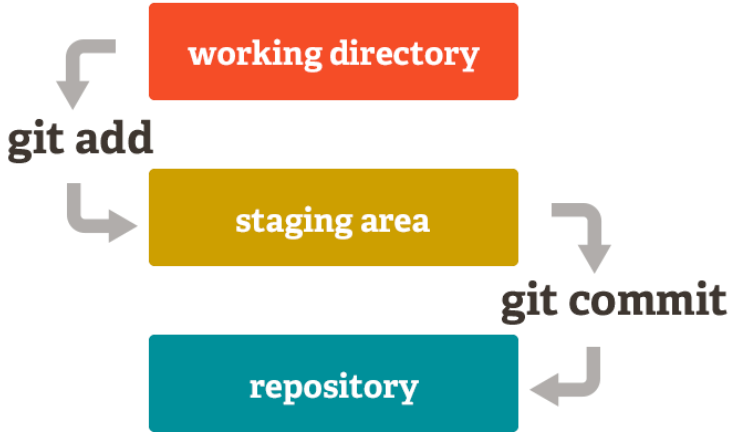
The Local Repository:

The Local Repository is everything in your .git directory. Mainly what you will see in your Local Repository are all of your checkpoints or commits. It is the area that saves everything

Staging Area

- The git staging area is in the middle of the workspace and repository. It is a file which stores the information of the added files. Before a commit, you add changed files or new files. The information of those files will be saved in the file which represents the index or staging area.
- You can add files or remove files from the staging area





Let’s Start With Git Basic Commands

Installing Git with Default Packages

- First, use the apt package management tools to update your local package index.

sudo apt update

- With the update complete, you can install Git:

sudo apt install git

- You can confirm that you have installed Git correctly by running the following command and checking that you receive relevant output.

git --version

Setup Git

Before you start using Git, you need to set it up on your local machine. Here's how:

- **Install Git:** If you haven't already, download and install Git from the official website (https://git-scm.com/downloads). Once installed, open your terminal or command prompt to verify the installation by typing git --version.

- **Configuration**: Set your name and email address for Git to use in your commits using git config and This command sets the author name and email address respectively to be used with your commits.

  git config --global user.name "Your Name"

  git config --global user.email "youremail@example.com"

1.

git init

git init <directory> : This command creates empty git repo in the specified <directory>. If you run it with no arguments then it initializes the current directory as a git repository.

git init <directory>

After running this command you will find the .git folder in your directory where you initialized the local repository. This contains all objects, refs, and config related to the local repository.

git Clone

- git clone is a Git command used to create a copy of an existing Git repository.
- It essentially allows you to copy a repository, including all its files, branches, commit history, and configurations from a remote repository (such as one hosted on GitHub, GitLab, Bitbucket, etc.) to your local machine

Syntax :

git clone <repository_URL> [<directory_name>]

✓ <repository_URL>: The URL of the remote repository you want to clone.

✓ [<directory_name>] (optional): The name of the directory where you want to clone the repository. If not specified, the repository will be cloned into a directory with the same name as the repository itself.

Git Add Command

- git add is a command in Git that adds changes in the working directory to the staging area, preparing them to be included in the next commit.
- It allows you to select specific files or parts of files (with the interactive mode) to be included in the commit.

git add [<file_pathspec>...]

- **git add** moves changes from the working directory to the staging area, preparing them for the next commit. After staging changes using **git add**, you need to commit them using **git commit** to permanently save those changes to the Git history.
- It's possible to add only specific parts of a file to the staging area using the interactive mode.

git add -p filename.txt

This command allows you to interactively choose which changes within the file filename.txt to stage.

Git commit Command

- git commit is a command in Git that records the changes made to the files in the staging area as a new commit in the version history of the repository.

git commit -m "Commit message“

Here -m "Commit message": Flag used to specify a commit message, describing the changes made in the commit. The commit message should be meaningful and concise, summarizing the changes made.

Git commit Command

- After staging changes using **git add**, you use **git commit** to permanently save those changes to the Git repository's history.
- When you run **git commit**, Git opens a text editor (unless the **-m** flag is used to specify the message inline) where you can provide a detailed commit message explaining what changes were made.
- A commit is a snapshot of the changes made to the repository at a particular point in time. Each commit has a unique identifier (hash) and contains information about the changes, including the author, timestamp, and the commit message.
- It's good practice to create commits that represent logical units of work, such as implementing a new feature, fixing a bug, or making specific changes. This helps in maintaining a clear and comprehensible history of the project.
- Commits serve as checkpoints in the project's history, enabling you to track changes, revert to previous states if needed, collaborate with others, and understand the evolution of the codebase.

Git Status Command

- The git status command in Git is used to display the current state of the working directory and the staging area (index), providing information about files that have been modified, staged, or untracked.

Syntax : **git status**

When you run git status, you'll see information categorized into three main sections:

1.Changes not staged for commit:

- Lists modified files that have been changed but not yet staged for the next commit.
- Files that are modified but not added to the staging area using **git add** will appear under this section.

2.Changes to be committed:

- Displays files that are staged and ready to be committed.
- Files that have been added to the staging area using **git add** will be listed here.

3.Untracked files:

- Lists files in the working directory that are not tracked by Git.
- These files have not been added to the staging area and are not part of the repository history.

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   newfile.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_untracked_file.txt
```

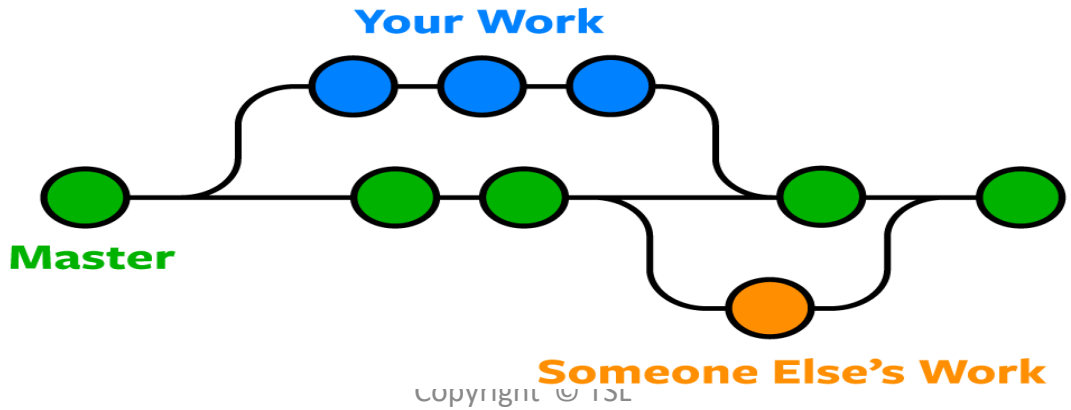
Git diff

- The git diff command in Git is used to show changes between different Git objects, such as commits, branches, files, and the working directory. It provides a detailed view of the differences in content between two points in the Git history

Syntax : **git diff**

Branch in Git

- Git branching is the process of creating multiple, independent lines of development within a single repository. Each branch represents a separate version of the codebase, allowing developers to work on new features, bug fixes, or experiments without affecting the main codebase.
- Once the work on a branch is complete and tested, it can be merged back into the main branch (ideally with a pull request where you request to merge your code, and your colleagues check and approve it), integrating the changes with the rest of the code.



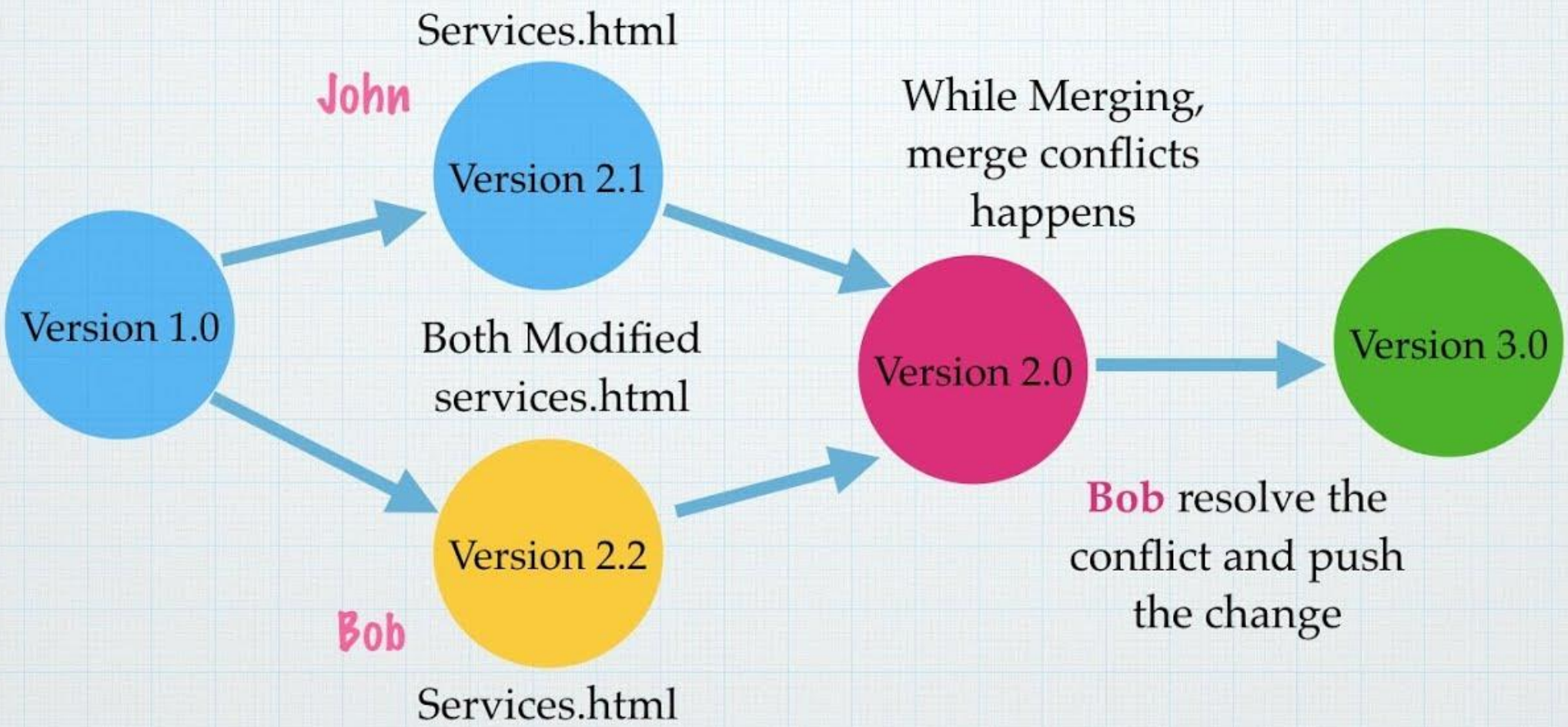
Why Use Branches?

- **Isolated development** — Branches provide a safe environment for you to mess around without worrying about screwing up the main branch.
- **Collaboration** — Multiple developers can work simultaneously on different branches.
- **Easy integration** — we can easily merge our code from different branches back into the main branch for easy integration.
- **Experimentation** — Branches allow developers to experiment with new ideas or techniques without the risk of breaking the main codebase. If you screw up, you can simply delete the branch and start over.
- Code reviews become more focused and manageable.

Git Merge Conflicts

Git merge conflicts occur when there are conflicting changes between two branches that Git is attempting to merge. This typically happens when two branches have made different modifications to the same part of a file, and Git cannot automatically determine which changes to keep. When a merge conflict occurs, Git requests manual intervention to resolve the conflict

Work with merge conflict in Git



Git Stash

- The **git stash** command is used to temporarily save changes that are not ready to be committed.
- This is useful when you need to switch to a different branch or work on a different task without committing your changes.
- When you run **git stash**, Git will take the changes that you've made to tracked files (i.e., files that are already being tracked by Git) and save them in a "stash" or "temporary commit."
- Git will then revert the working directory to the state it was in when you last committed changes.

Git Stash Commands

- git stash → used for send the file from WD to stash
- git stash apply stashid → used for fetch file from stash to WD
- git stash show → to know the status of stash
- git stash clear → for clear stash storage
- git stash list → list the all files in stash area

Git Reset and Revert

- The **git reset** command is used to reset the current branch to a specific commit or to unstage changes that have been staged for commit. This means that git reset changes the commit history by removing commits from the branch. If you use git reset to undo a commit, the commit and all its changes will be permanently removed from the branch history. This means that any changes that were made in the removed commit will be lost.
- The **git revert** command is used to create a new commit that undoes the changes made by a previous commit. This means that git revert does not remove any commits from the branch history. Instead, it adds a new commit that contains the changes needed to undo the previous commit. This allows you to keep a complete history of all changes made to the branch.

Git Reset and Revert

- Git reset file : reset file from staging area to WD
- Git reset –Hard : remove file from staging area and WD
- Git reset commitid : It will reset the commit
- Git revert commit id : it will revert the commit and remove all changes in files also

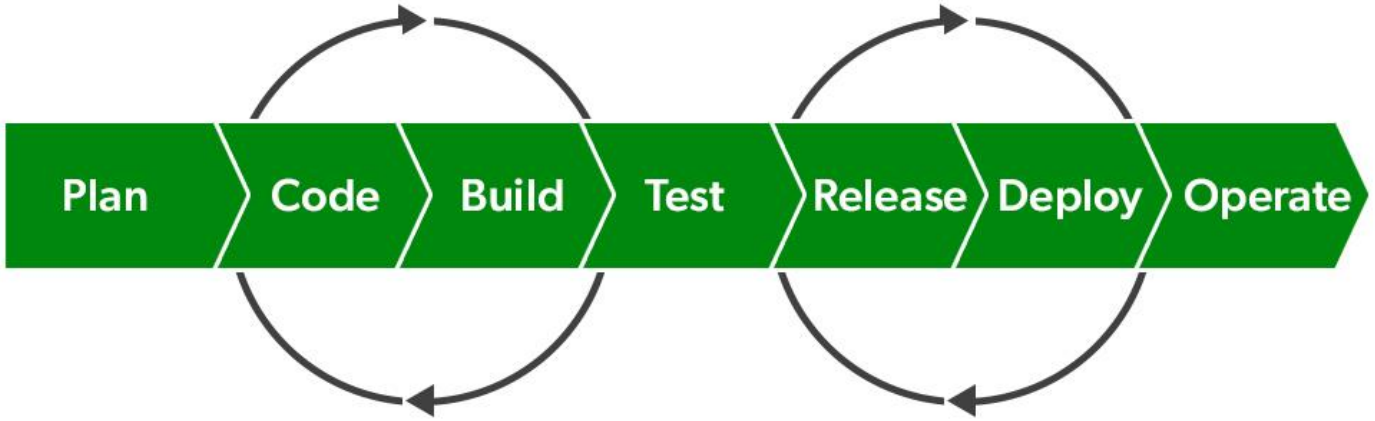
Git Tag

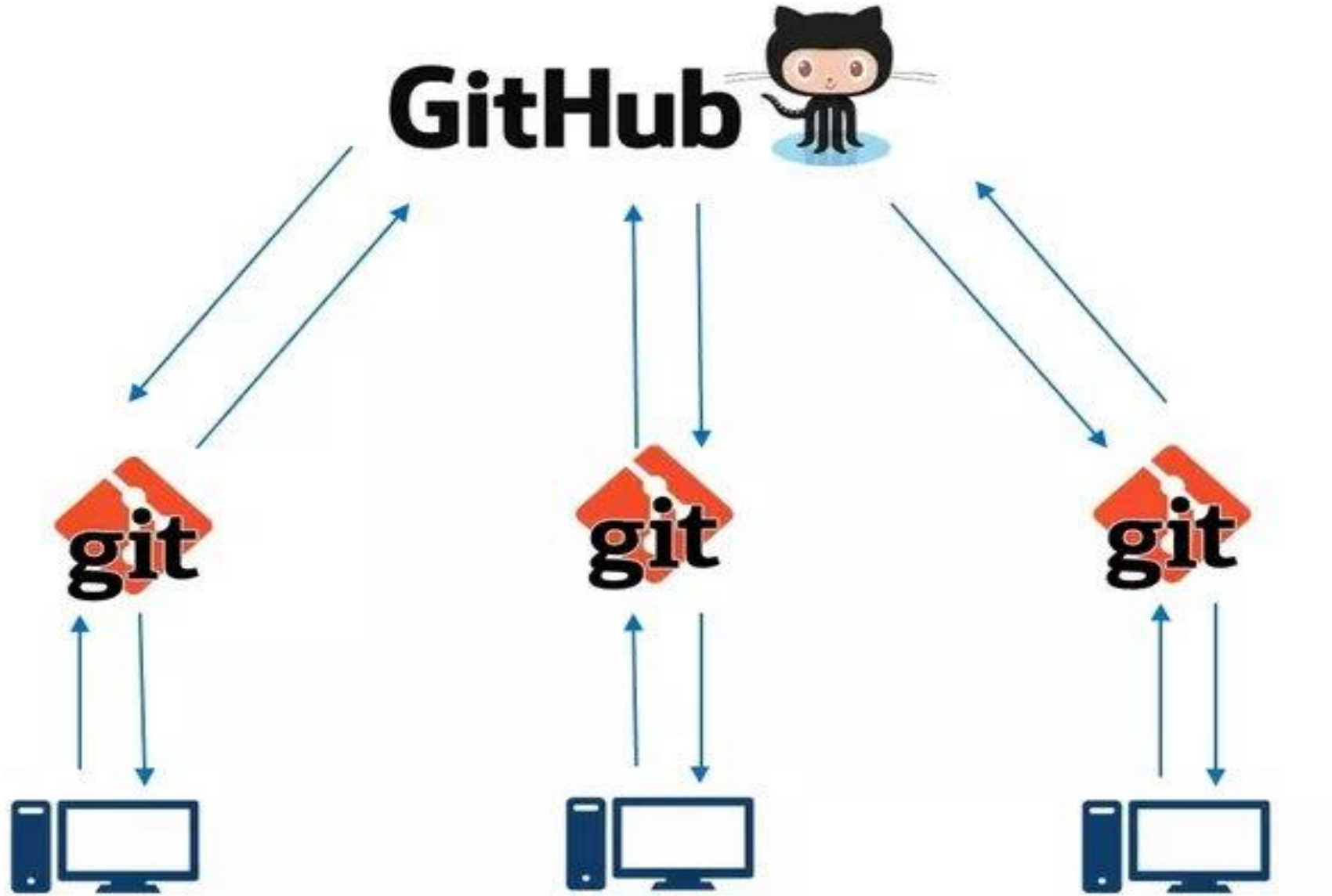
- In Git, a tag is a label that points to a specific commit in a repository's history. It is a way to mark a specific version or release of the code at a certain point in time.

- A Git tag is like a bookmark in a book. It allows you to mark a specific commit and easily refer back to it later. This can be useful for keeping track of releases or marking important milestones in a project's development history.

Git Hub Action

GitHub Actions is a CI/CD (Continuous Integration/ Continuous Deployment) platform for automating the builds, test, and deployment process. Using GitHub actions, we can build and test every pull request in the repository using workflows, or push the merged pull requests to production with workflows




GitHub
git
git
git

Knowledge Check

What is a branch in Git?

a) A branch is a separate line of development in a Git repository where changes can be made without affecting the main branch.

b) A branch is a cloud-based storage service for Git repositories.

c) A branch is a type of commit used for merging changes from one branch to another in a Git repository.

d) A branch is a folder within a Git repository where specific files are stored.

Knowledge Check

What is the default branch name for a newly created repository?

a) Master

b) Develop

c) Release

d) Main

Version Control helps in

- Its not just for code, it also helps in
    - Backups & Restoration
    - Synchronization
    - Reverts
    - Track Changes
    - Most importantly in Parallel Development

Centralized Version Control System

- Traditional version control system
    - Server with database
    - Clients have a working version
- Examples
    - CVS
    - Subversion
    - Visual Source Safe
- Challenges
    - Multi-developer conflicts
    - Client/server communication

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

GIT Installation

- GIT comes as default offering for all major Linux flavors
- Though you can download the installers from below link for Windows, Mac OS X, Linux, Solaris
  - https://git-scm.com/downloads
  - you can install same with yum too:
    - yum install git
- Installing GIT Bash on Windows

GIT VERSION

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

Setting Identity in GIT

```
[root@Techlanders ~]# git config --global  user.email “raman.khanna@Techlanders.com”
[root@Techlanders ~]# git config --global  user.name “raman.khanna”
[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]#
```

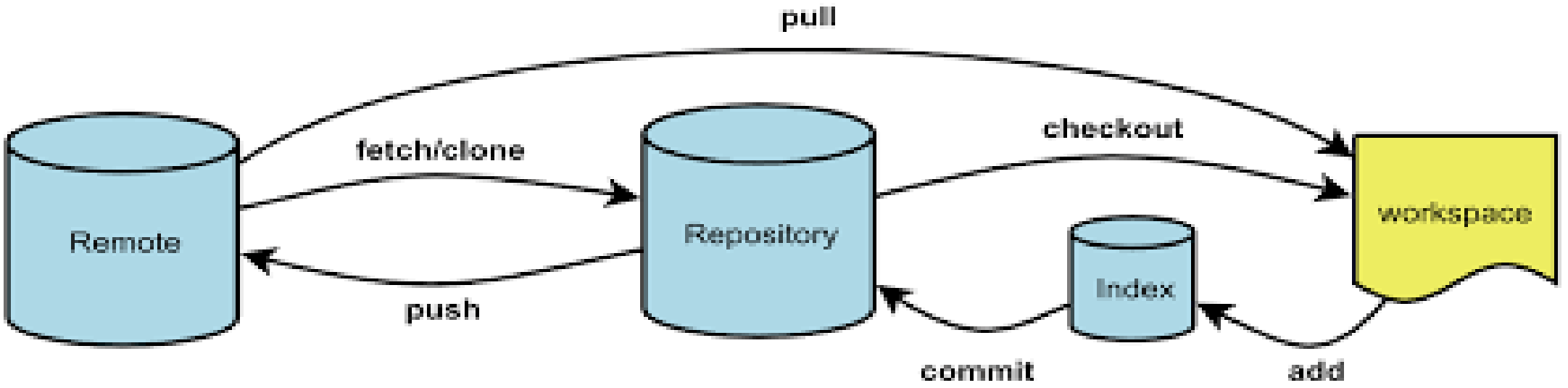
Git Log

```
[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


pull
fetch/clone
checkout
Remote
Repository
workspace
Index
push
commit
add

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]#
```

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 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(+)
```

- 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
```

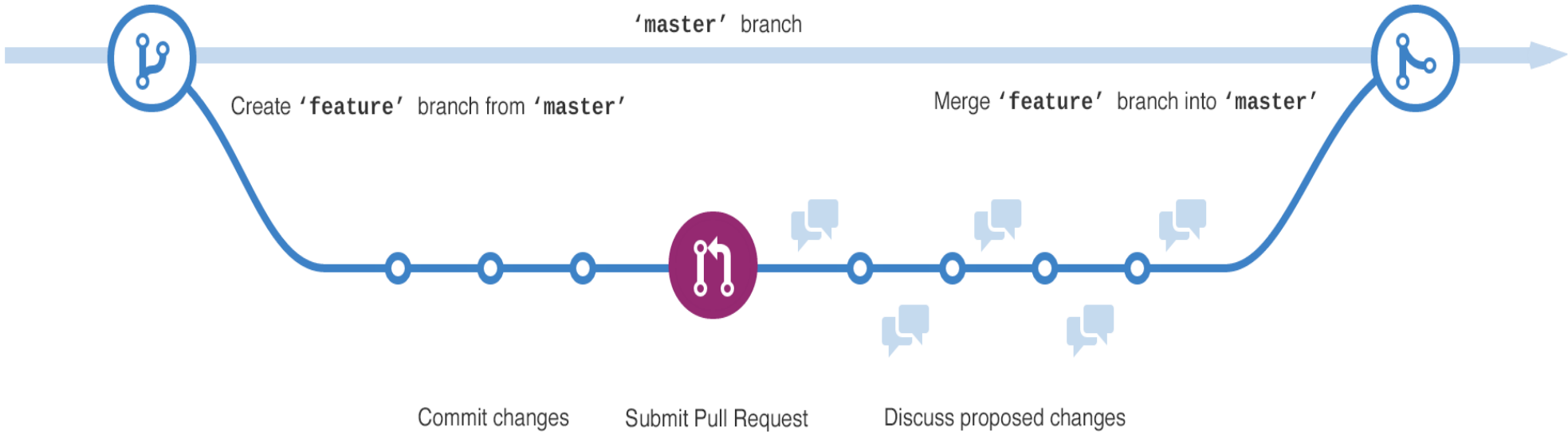
GIT Branch

- 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



BitBucket/GitHub/GitLab

- **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.

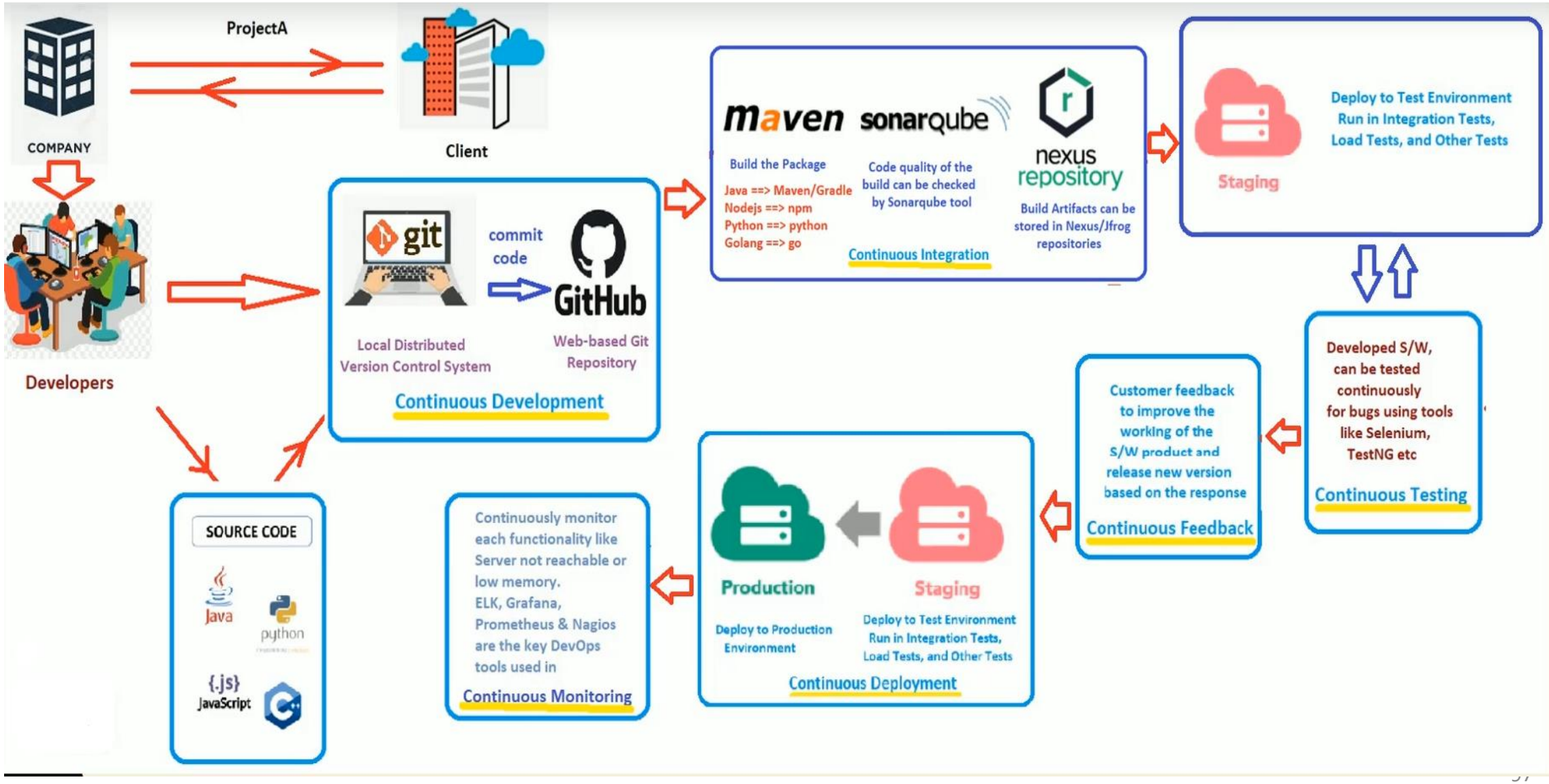
- Visit "https://bitbucket.org/" and click "Get Started" to sign up for free account.
- Visit "https://github.com/" for Github details

Maven

1. Apache maven is a software project management and comprehensive tool and based on the concept of project object model POM, Maven can manage a project, build reporting and documentation from a central piece of information.

2. For the person building a project, This means that it is only necessary to learn a small set of commands to build any Maven project and the POM will ensure they will get the results they desire

Maven


ProjectA
COMPANY
Client
Developers
git
commit
code
GitHub
Local Distributed
Version Control System
Web-based Git
Repository
Continuous Development
maven
sonarqube
nexus
repository
Build the Package
Java ==> Maven/Gradle
Nodejs ==> npm
Python ==> python
Golang ==> go
Code quality of the
build can be checked
by Sonarqube tool
Build Artifacts can be
stored in Nexus/Jfrog
repositories
Continuous Integration
Staging
Deploy to Test Environment
Run in Integration Tests,
Load Tests, and Other Tests
Developed S/W,
can be tested
continuously
for bugs using tools
like Selenium,
TestNG etc
Continuous Testing
Customer feedback
to improve the
working of the
S/W product and
release new version
based on the response
Continuous Feedback
Production
Staging
Deploy to Production
Environment
Deploy to Test Environment
Run in Integration Tests,
Load Tests, and Other Tests
Continuous Deployment
Continuously monitor
each functionality like
Server not reachable or
low memory.
ELK, Grafana,
Prometheus & Nagios
are the key DevOps
tools used in
Continuous Monitoring
SOURCE CODE
Java
python
{.js}
JavaScript

Maven Life Cycle

1. The **default** lifecycle which handles your project deployment.
2. The **clean** lifecycle, which hinders project cleaning.
3. The **site** lifecycle, handles the creation of your project's website.

Maven Repo

There are 3 types of repositories.

1. Local -> $(HOME)/.m2/repository
2. Remote and private
3. Central and public-> Public from Maven Website

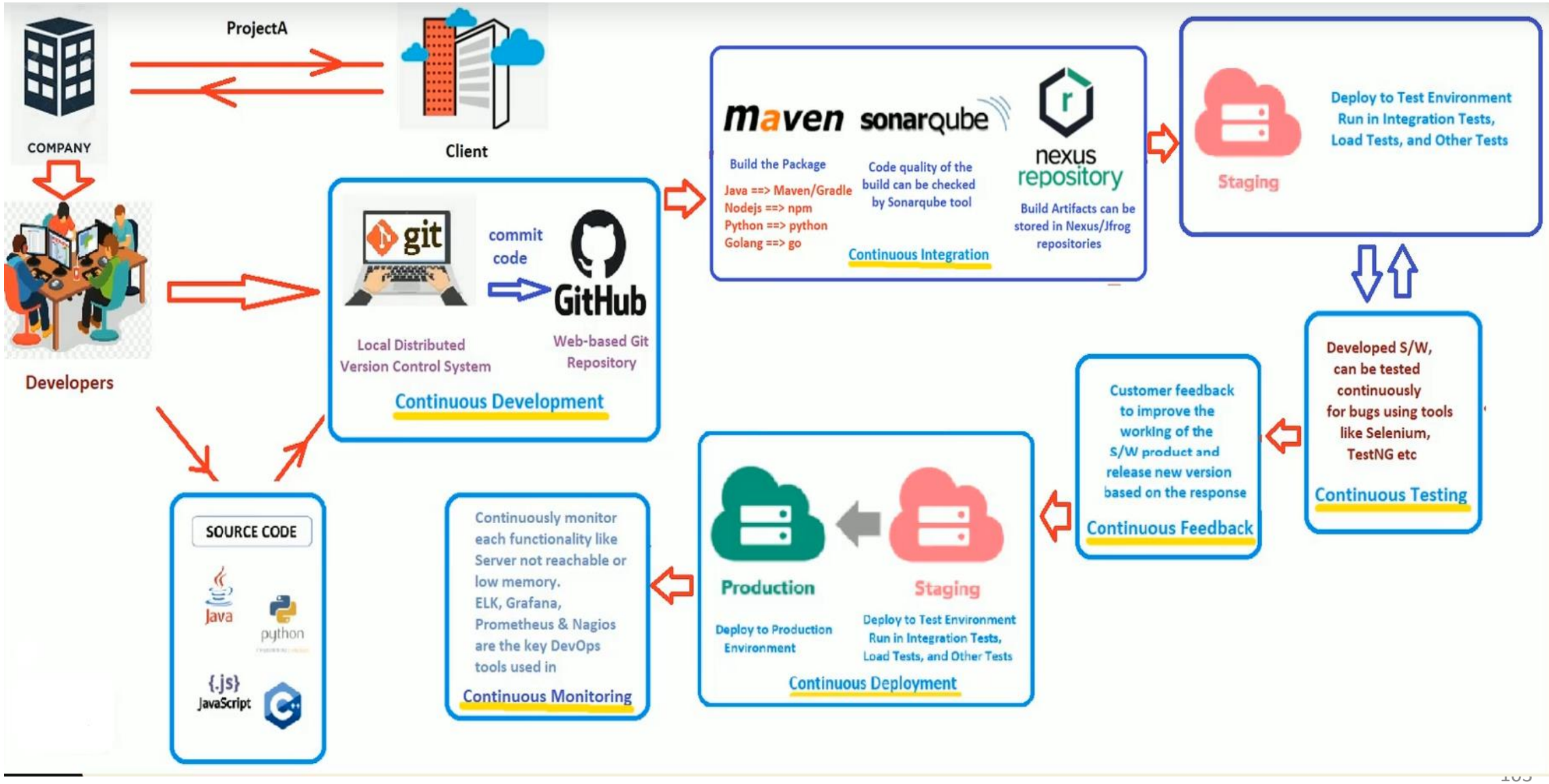
Lab : Installation and configuration of Maven


SonarQube

SonarQube

- SonarQube is a Code Quality Assurance tool that collects and analyzes source code, and provides reports for the code quality of your project.
- It combines static and dynamic analysis tools and enables quality to be measured continually over time.
- SonarQube also ensures code reliability, Application security, and reduces technical debt by making your code base clean and maintainable.
- SonarQube also provides support for 27 different languages, including C, C++, Java, Javascript, PHP, GO, Python, and much more.
- SonarQube also provides Ci/CD integration, and gives feedback during code review with branch analysis and pull request decoration.

SonarQube


ProjectA
COMPANY
Client
Developers
git
commit
code
GitHub
Local Distributed
Version Control System
Web-based Git
Repository
Continuous Development
maven
sonarqube
nexus
repository
Build the Package
Java ==> Maven/Gradle
Nodejs ==> npm
Python ==> python
Golang ==> go
Code quality of the
build can be checked
by Sonarqube tool
Build Artifacts can be
stored in Nexus/Jfrog
repositories
Continuous Integration
Staging
Deploy to Test Environment
Run in Integration Tests,
Load Tests, and Other Tests
Developed S/W,
can be tested
continuously
for bugs using tools
like Selenium,
TestNG etc
Continuous Testing
Customer feedback
to improve the
working of the
S/W product and
release new version
based on the response
Continuous Feedback
Production
Staging
Deploy to Production
Environment
Deploy to Test Environment
Run in Integration Tests,
Load Tests, and Other Tests
Continuous Deployment
Continuously monitor
each functionality like
Server not reachable or
low memory.
ELK, Grafana,
Prometheus & Nagios
are the key DevOps
tools used in
Continuous Monitoring
SOURCE CODE
Java
python
{.js}
JavaScript

SonarQube

- The code should follow a specific convention
- The code should be following established good practices
- Checked for potential bugs and performance, security, or vulnerabilities issues
- Is the code duplicated anywhere
- Does the code make logical sense, or is it too complex
- Does the public API have good documentation and comments
- Does the code have unit tests
- Doe the code follow good software design and architecture principle

Dynamic Code Analysis

Dynamic Code Analysis relies on studying how the code behaves during execution. The objective is to find errors in a program while it is running, rather than by repeatedly examining the code offline

Static Code Analysis

- Static code analysis is done without executing any of the code. It is a collection of algorithms and techniques to analyze source code to automatically find potential errors and poor coding practices. This is done with compiler errors and run-time debugging techniques such as white box testing.

Lab : Installation and configuration of SonarQube

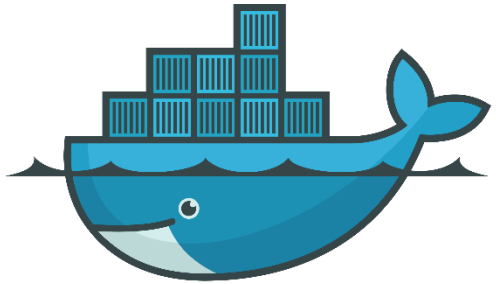
Quality profiles

- Quality profiles are a core component of SonarQube where you define sets of rules that, when violated, raise issues on your codebase (for example, methods should not have a cognitive complexity higher than 15). Each individual language has its own quality profile.

Quality Gates

- Quality gates enforce a quality policy in your organization by answering one question: is my project ready for release? To answer this question, you define a set of conditions against which projects are measured. For example:
- No new blocker issues
- Code coverage on new code greater than 80%

Lab : Create a Quality Gates



Docker

What is Container?

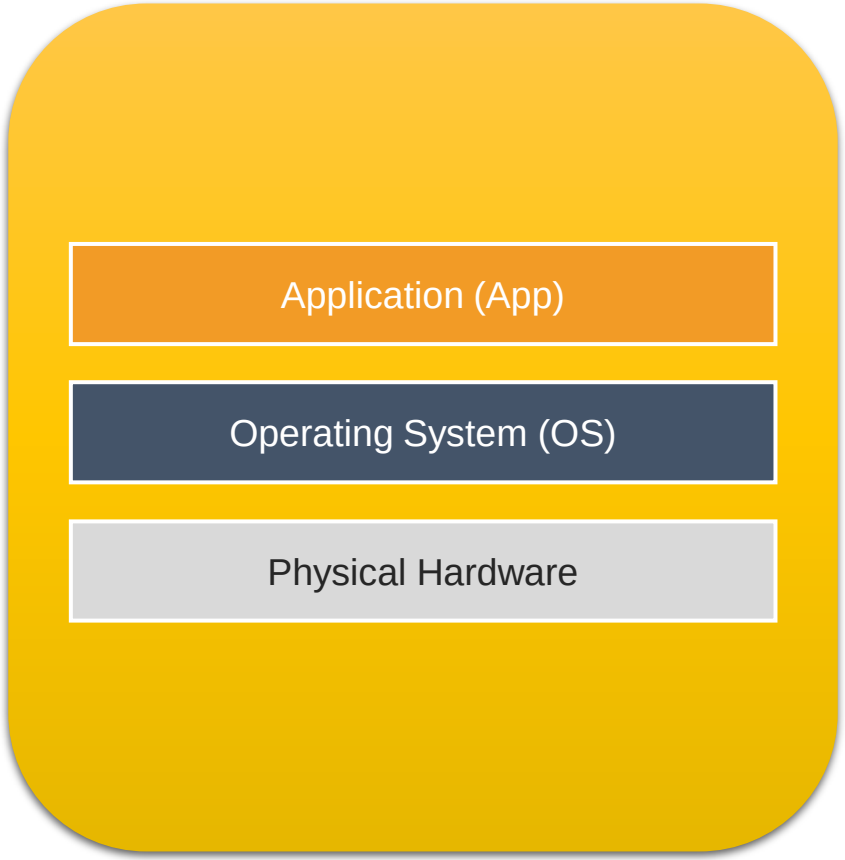
Before proceeding with the new terms and technologies lets have a look on the history/traditional approaches used for the application since ages.

Physical Servers

Unused Resources

Difficult Migrations

Vendor dependency



Slow Deployment time

Difficult to Scale

Huge cost in Infrastructure

Traditional Approach

Virtualization

OverHead

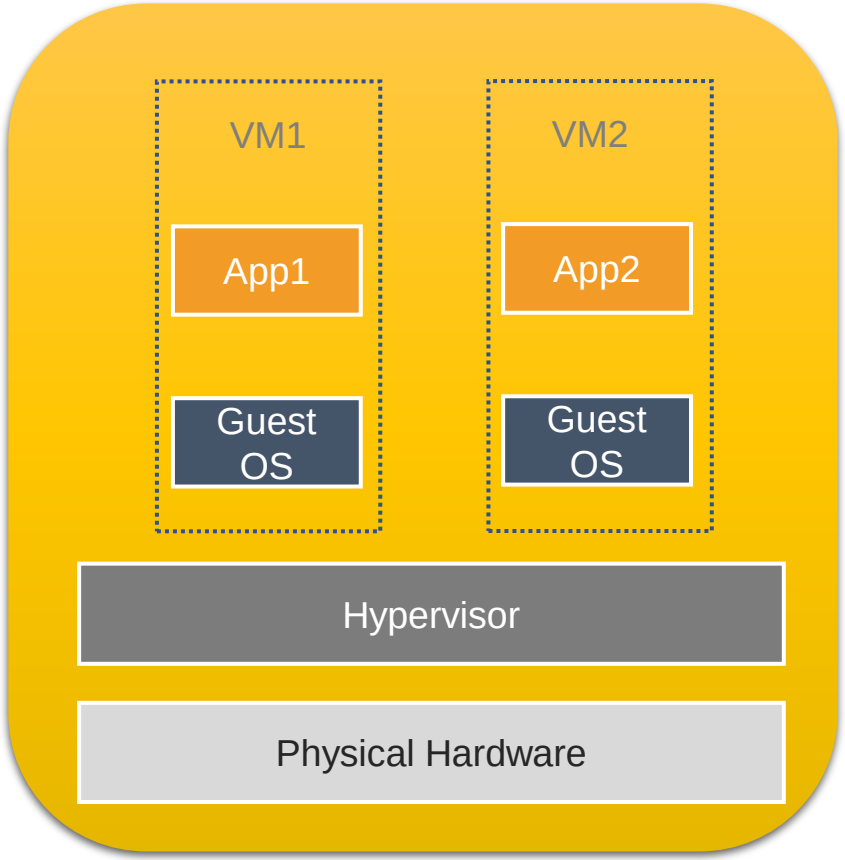
Portability issues

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

Containers

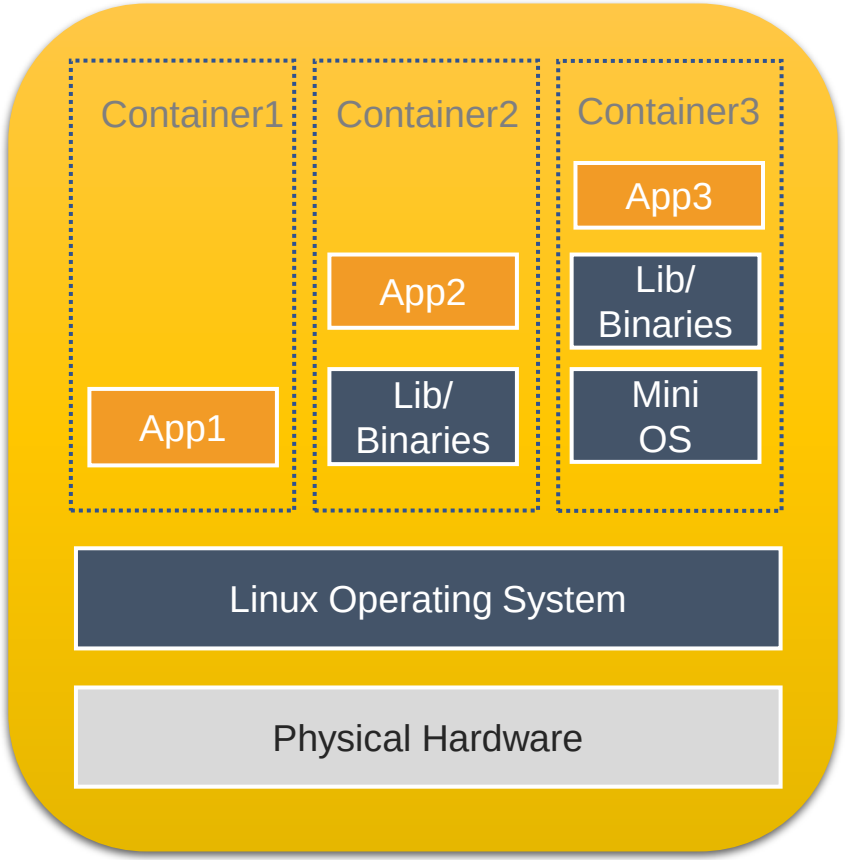
Less OverHead

Highly Portable

Scaling in Hybrid Env

High Migrations success ratio

Cost Effective Solution



Containerization

Much Better Resource Pooling

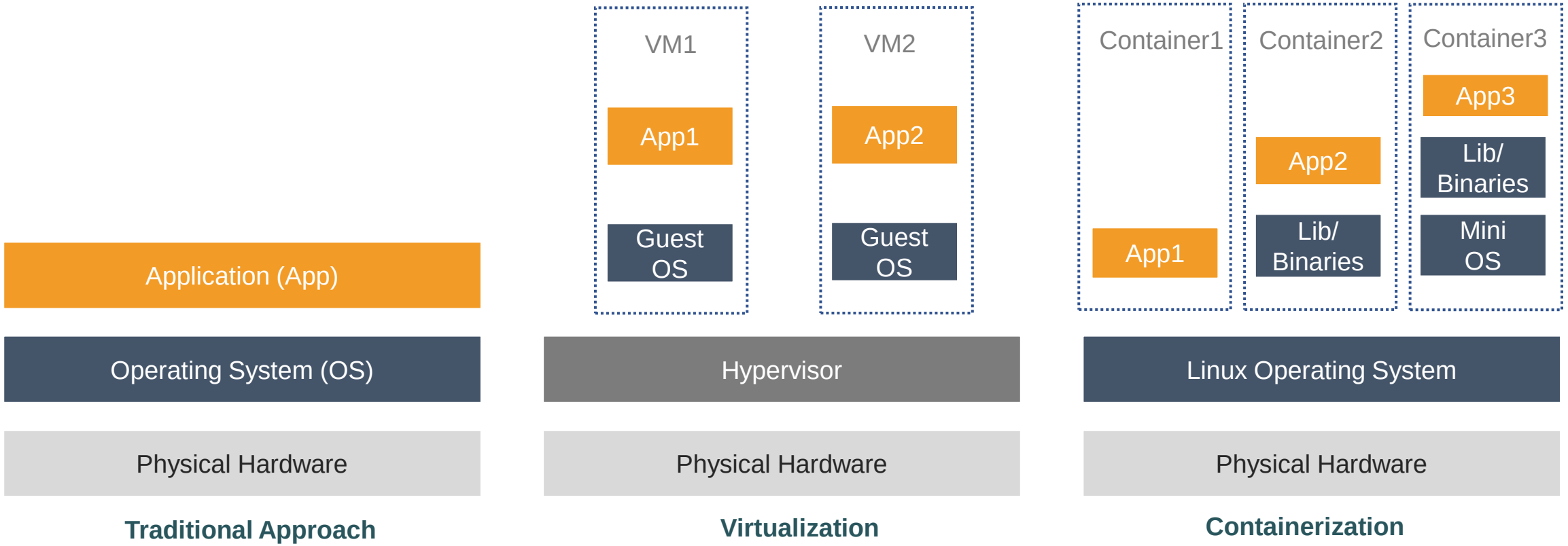
Extended Scaling

Flexibility & Easy Migration

Faster Deployments (Seconds)

Faster Boot time (Seconds)

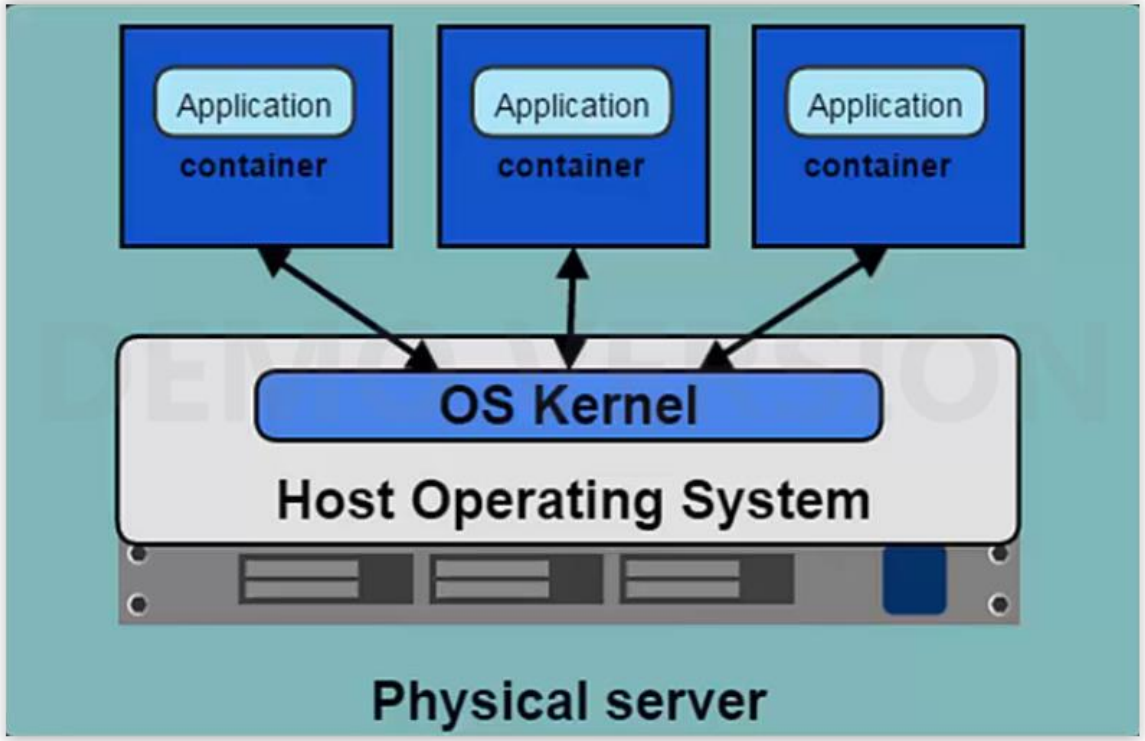
Containers


Application (App)
Operating System (OS)
Physical Hardware
Traditional Approach
VM1
App1
Guest OS
VM2
App2
Guest OS
Hypervisor
Physical Hardware
Virtualization
Container1
App1
Container2
App2
Lib/ Binaries
Container3
App3
Lib/ Binaries
Mini OS
Linux Operating System
Physical Hardware
Containerization

Introducing Containers

A container is a lightweight, portable, and executable software package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers provide a consistent and isolated environment for applications, ensuring that they run reliably and consistently across various computing environments.

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



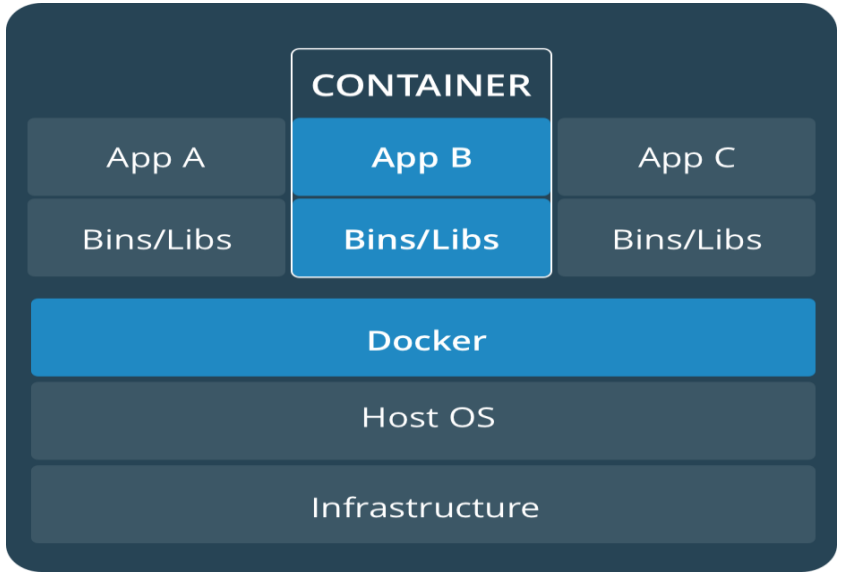
Key characteristics of containers

1. **Isolation:** Containers encapsulate an application and its dependencies, ensuring that they run in an isolated environment. This isolation helps prevent conflicts between different applications or different versions of the same application.
2. **Portability:** Containers can run consistently across different environments, such as development, testing, and production. This portability is possible because containers include all the dependencies needed for an application to run, reducing compatibility issues.
3. **Efficiency:** Containers share the host operating system's kernel and resources, making them lightweight compared to virtual machines. They start up quickly and consume fewer resources, allowing for more efficient use of hardware.
4. **Scalability:** Containers are well-suited for scalable and distributed applications. They can be easily deployed and managed using container orchestration tools like Kubernetes, which automates the deployment, scaling, and management of containerized applications.
5. **Versioning and Reproducibility:** Containers can be versioned, allowing developers to package and distribute applications with specific dependencies and configurations. This ensures that the software behaves consistently across different environments.
6. **Microservices Architecture:** Containers are often used in microservices architectures, where complex applications are broken down into smaller, independent services. Each microservice can run in its own container, facilitating easy development, deployment, and maintenance.

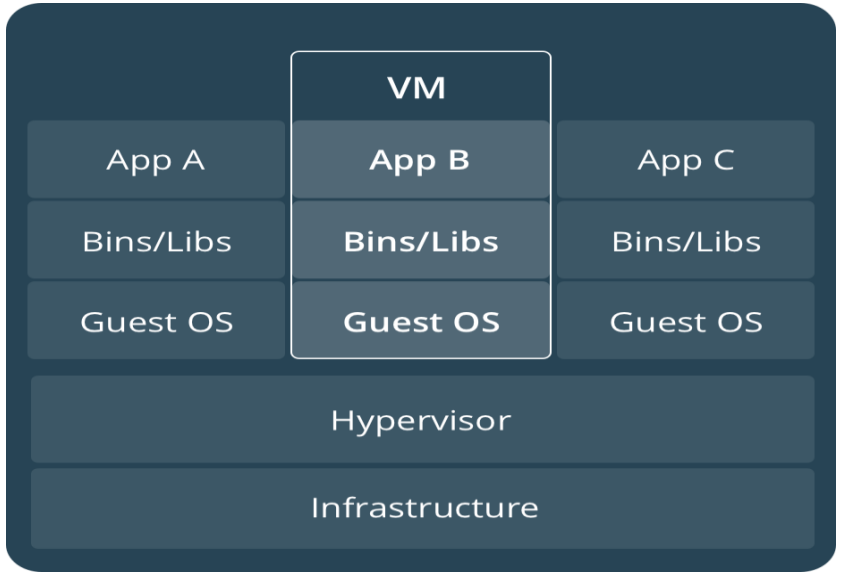
Containers VS VM's

Feature	Containers	Virtual Machines (VMs)
Abstraction Level	Operating system level (user space)	Hardware level (full OS emulation)
Resource Efficiency	Lightweight, share host OS resources	Heavier, each VM includes a full OS
Isolation	Process-level isolation	Stronger isolation with separate OS
Portability	Highly portable	Less portable due to larger size
Performance	Generally better performance	May have higher overhead
Deployment Speed	Quick startup and shutdown	Longer startup times
Management Tools	Docker, Kubernetes	Hypervisor-specific tools (e.g., VMware vSphere, Hyper-V)
Scaling	Rapid scaling with quick startup	Slower scaling with longer startup
Use Cases	Microservices, cloud-native development	Legacy applications, strong isolation
Orchestration	Docker, Kubernetes	Hypervisor-specific orchestration tools (e.g., vCenter, Hyper-V Manager)
OS Dependency	Shares host OS kernel	Each VM has its own OS kernel

Containers VS VM's

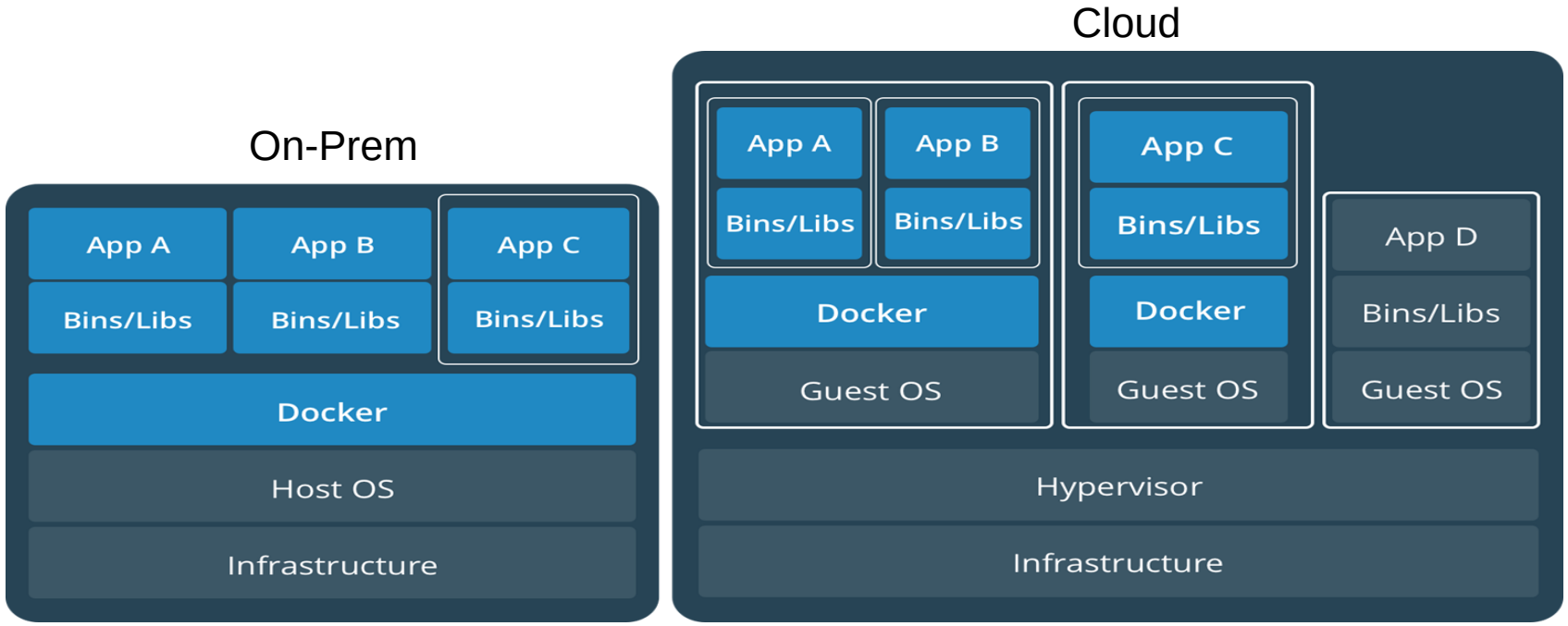


Containers are an app level construct



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.

Docker

- 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 taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

The Docker Platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- ✓ Develop your application and its supporting components using containers.
- ✓ The container becomes the unit for distributing and testing your application.
- ✓ When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

Docker is a platform to Build, Ship and Run containerized applications

```
Dockerfile

FROM    centos:latest
RUN     yum -y update
RUN     yum -y install httpd
RUN     echo "Hello Docker" >
/var/www/html/index.html
EXPOSE 80
CMD     ["httpd", "-D", "FOREGROUND"]
```

BUILD

Docker File

SHIP

Docker Image Registry

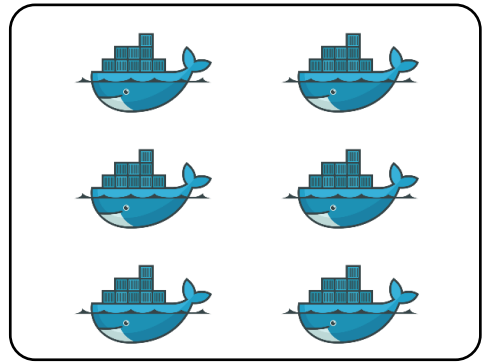
Docker Hub

ubuntu CentOS NGINX

MySQL HTTPD TOMCAT

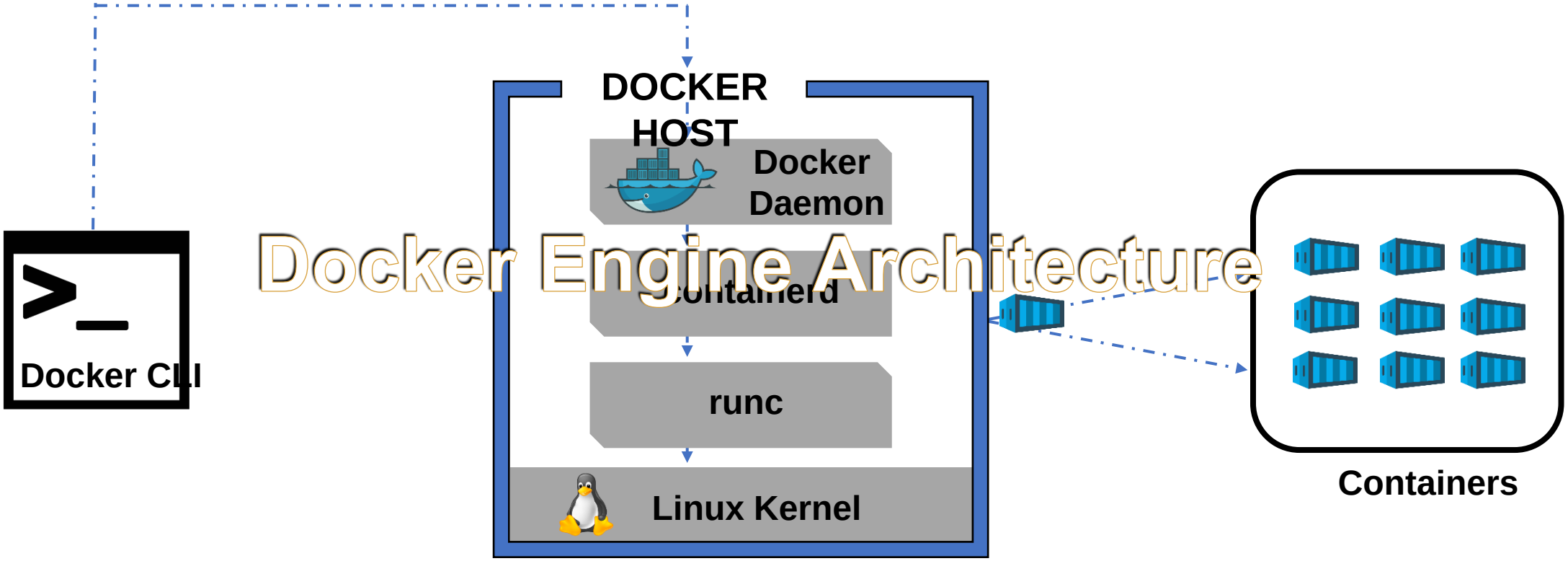
Docker Images

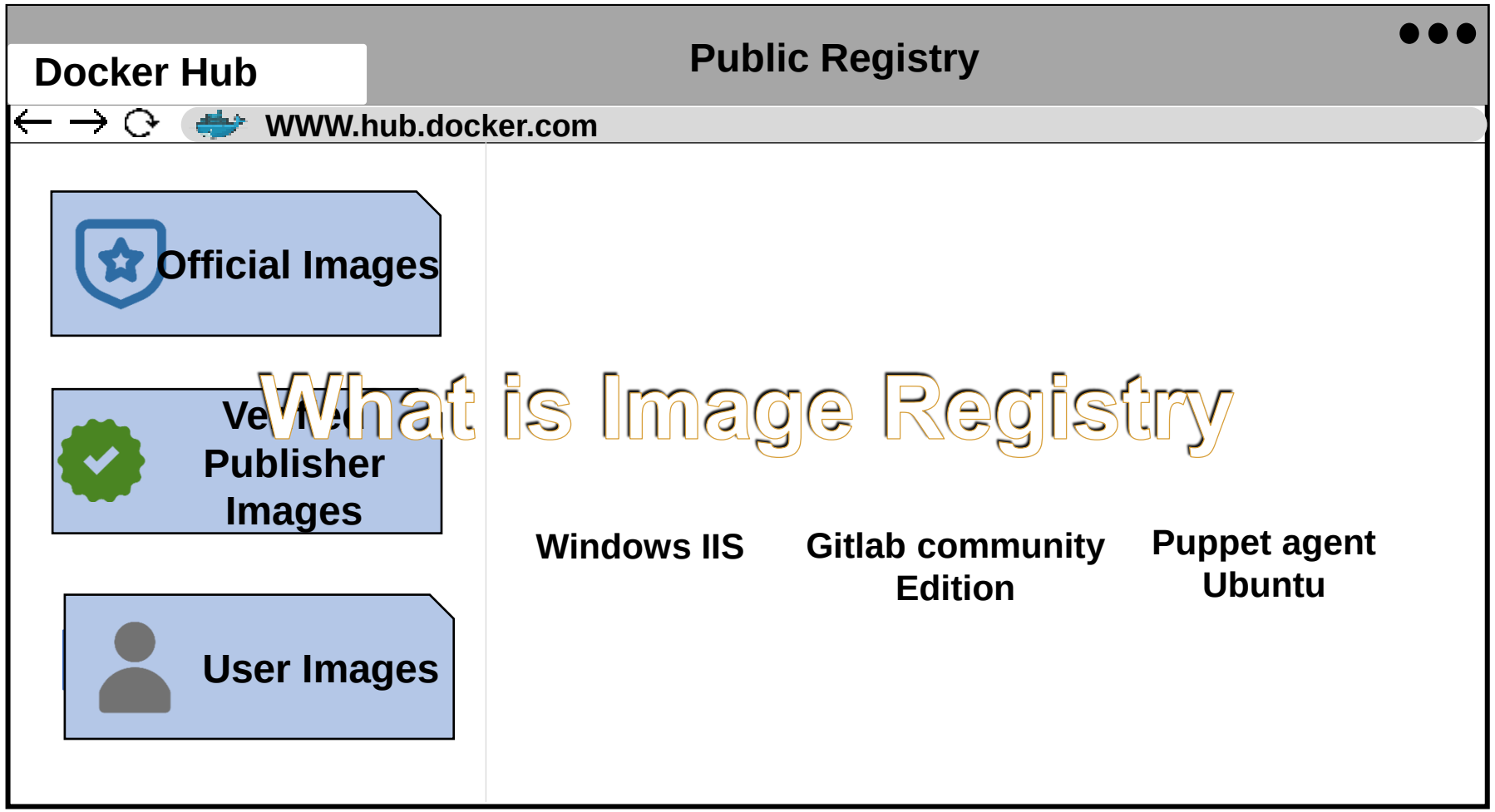
RUN



Docker Containers

What is Docker?


Docker Engine Architecture
DOCKER HOST
Docker Daemon
containerd
runc
Linux Kernel
Docker CLI
Containers


Docker Hub
Public Registry
WWW.hub.docker.com
Official Images
Publisher Images
User Images
What is Image Registry
Windows IIS
Gitlab community Edition
Puppet agent Ubuntu

Session: 2

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. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

The key components of Docker include the Docker Engine, Docker CLI (Command-Line Interface), Docker Compose, and Docker Registry.

Docker Architecture

The key components of Docker include the Docker Engine, Docker CLI (Command-Line Interface), Docker Compose, and Docker Registry.

Image

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

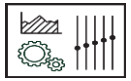
Container

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

Engine

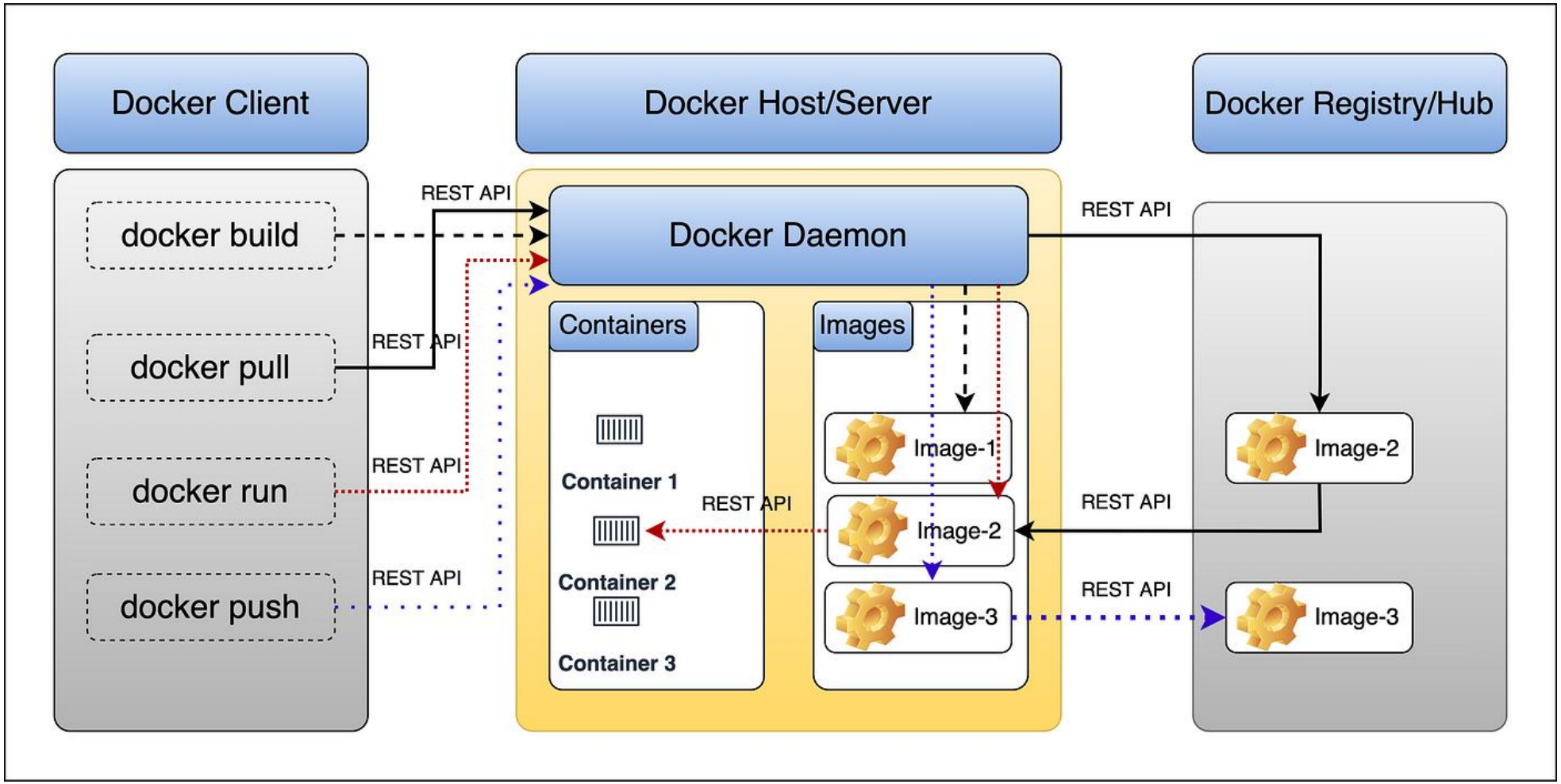
The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

Registry



Stores, distributes and manages Docker images

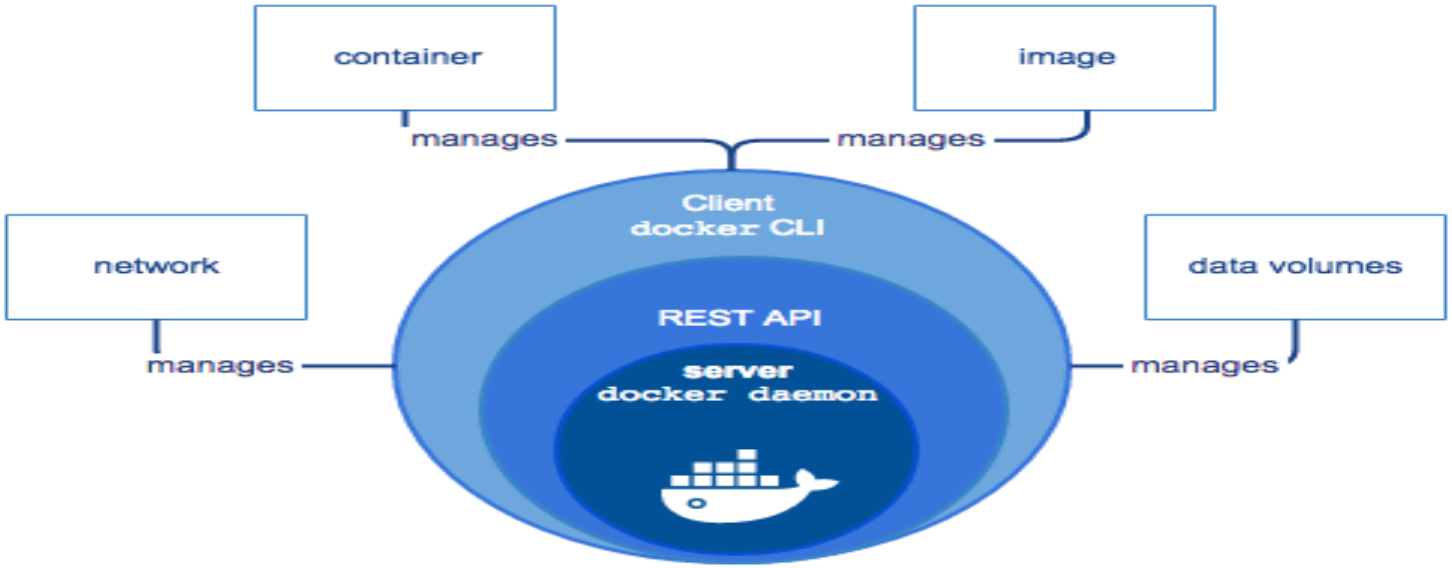
Docker Architecture



Docker Engine

The Docker Engine is the core component responsible for creating and managing containers. It includes several subcomponents, such as the Docker daemon, REST API, and the Docker command-line interface.

- Docker Daemon: This is a background process that manages Docker containers on a host system. It listens for Docker API requests and manages container lifecycles.
- Docker API: The API allows interaction with the Docker daemon, enabling users and applications to control Docker containers programmatically..



Docker CLI (Command-Line Interface)

The Docker CLI is a command-line tool that allows users to interact with the Docker Engine. It provides a set of commands for building, managing, and inspecting containers.

Common Commands:

- ✓ *docker build: Builds a Docker image from a Dockerfile.*
- ✓ *docker run: Creates and starts a container based on a specific image.*
- ✓ *docker ps: Lists running containers.*
- ✓ *docker images: Lists available Docker images.*
- ✓ *docker exec: Runs a command inside a running container.*

Docker Registry:

A Docker Registry is a centralized repository for storing and distributing Docker images. It allows users to share and access container images, facilitating collaboration and deployment.

Public Registries: Examples include Docker Hub, where users can find and share public Docker images.

Private Registries: Organizations can set up their own private registries for secure storage and distribution of custom Docker images.

Docker Images

Description: Docker images are lightweight, standalone, and executable packages that contain everything needed to run an application, including the code, runtime, libraries, and system tools.

Layered File System: Docker images are composed of layers, each representing a specific set of changes. This layered file system allows for efficient image sharing and distribution.

Docker Container

- A Docker container is a running instance of a Docker image or Docker containers are the run component of Docker.
- 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.
- Containers share the host OS kernel but have their own user space, providing process-level isolation.
- Container are more light weight
- No need to install dedicated guest OS,
- Less CPU, RAM, Storage Space required
- More containers per machine than VM’s
- Great Portability

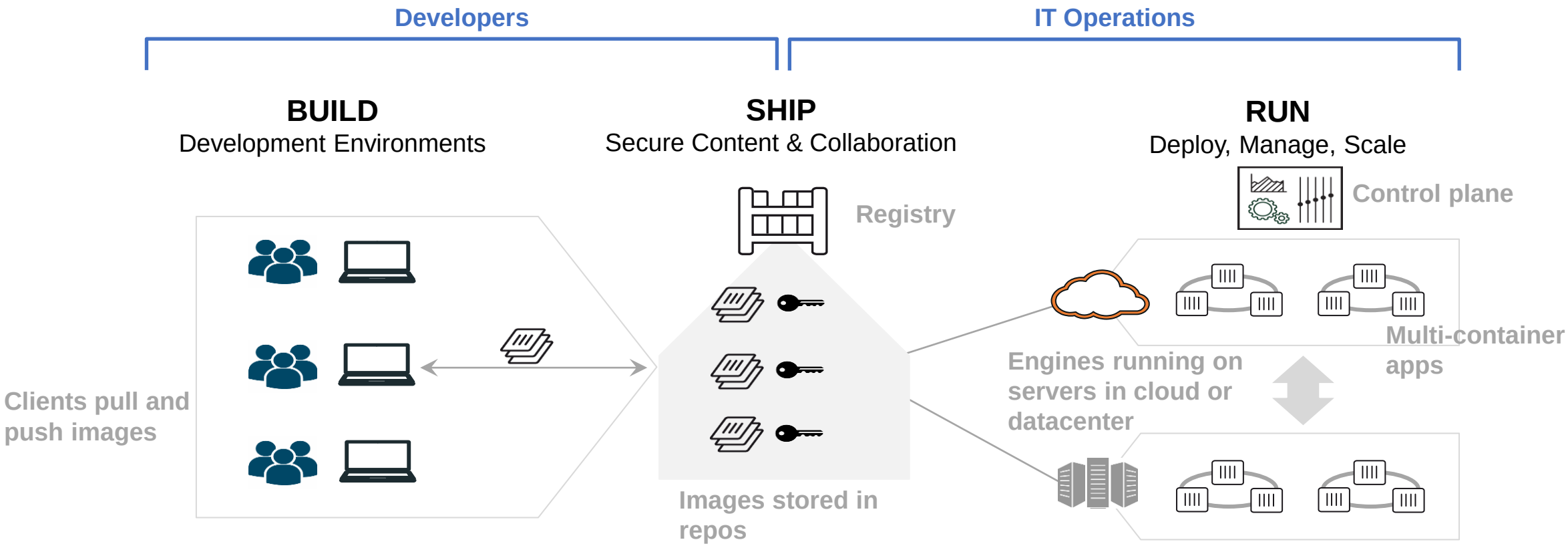
Docker Features

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 source and work with all OS : 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

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

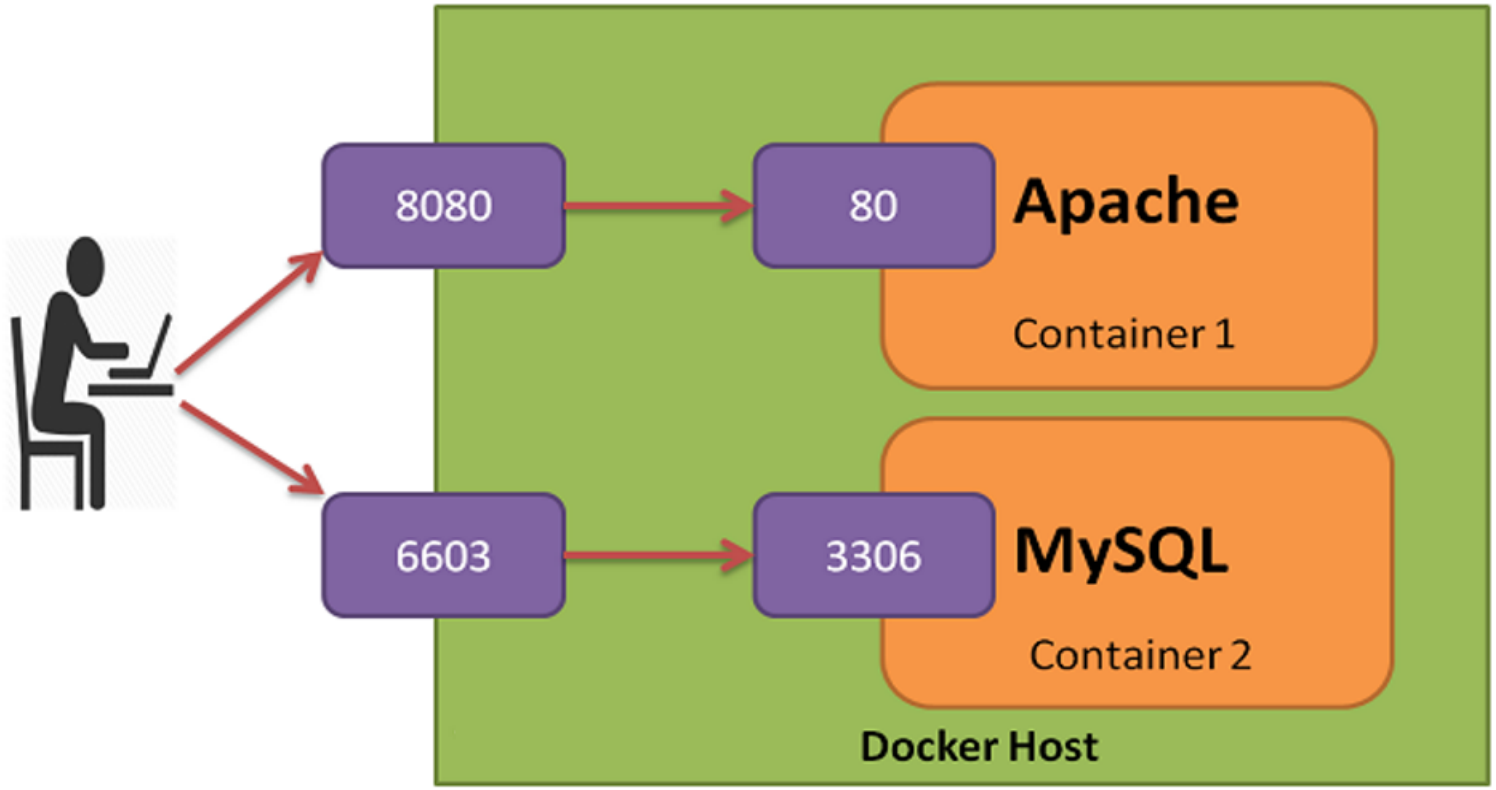
Docker inspect c2 | grep –I cpu

Session: 5

Ports and Volumes

Docker Ports

In Docker, ports are used to expose and publish network services running within a container to the host machine or to the external network. Docker containers can run processes that listen on specific ports, and these ports need to be mapped to the host system for external access



Published ports

By default, when you create or run a container using docker create or docker run, the container doesn't expose any of its ports to the outside world. Use the --publish or -p flag to make a port available to services outside of Docker. This creates a firewall rule in the host, mapping a container port to a port on the Docker host to the outside world. Here are some examples:

Flag value	Description
-p 8080:80	Map port 8080 on the Docker host to TCP port 80 in the container.
-p 192.168.1.100:8080:80	Map port 8080 on the Docker host IP 192.168.1.100 to TCP port 80 in the container.
-p 8080:80/udp	Map port 8080 on the Docker host to UDP port 80 in the container.
-p 8080:80/tcp -p 8080:80/udp	Map TCP port 8080 on the Docker host to TCP port 80 in the container, and map UDP port 8080 on the Docker host to UDP port 80 in the container.

Port Mapping

Description: Port Mapping allows you to map a specific port on the host machine to a port on the container. This enables external access to services running inside the container.

Syntax: **docker run -p host_port:container_port**

Example:

docker run -p 8080:80 nginx

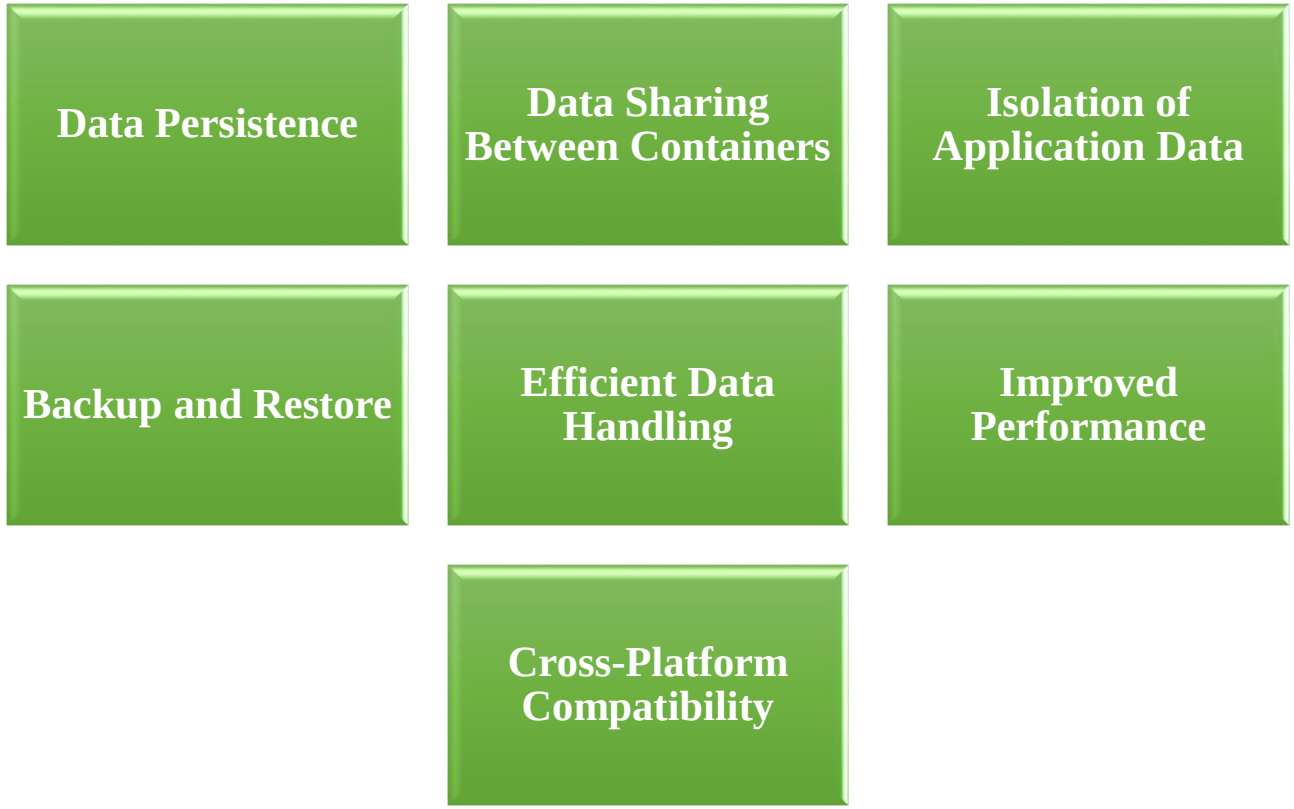
Docker Volume

By default all files created inside a container are stored on a writable container layer. This means that:

- ✓ The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.
- ✓ A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.
- ✓ Writing into a container's writable layer requires a storage driver to manage the filesystem. The storage driver provides a union filesystem, using the Linux kernel. This extra abstraction reduces performance as compared to using data volumes, which write directly to the host filesystem.

Primary Purposes and Benefits of Volume

Volumes serve the purpose of providing persistent and shared storage between containers and between the host machine and containers. Volumes are used to store and manage data that needs to persist beyond the lifecycle of a single container. They facilitate data sharing and are a key component for handling stateful applications.



Docker Storage Types

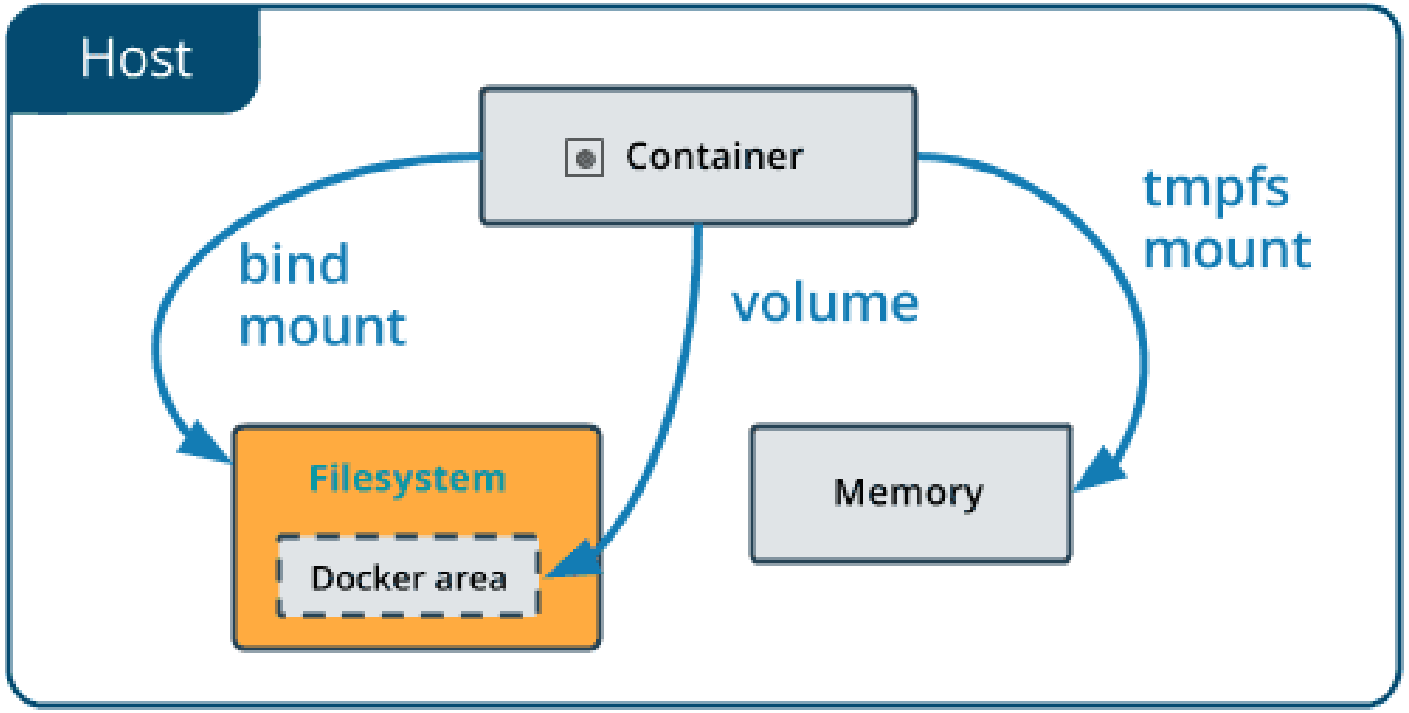
Docker has two options for containers to store files on the host machine, so that the files are persisted even after the container stops.

1. volumes
2. Bind mounts

Note : Docker has two options for containers to store files on the host machine, so that the files are persisted even after the container stops: volumes, and bind mounts.

Docker Volume

Docker volumes are completely handled by Docker itself and therefore independent of both your directory structure and the OS of the host machine. When you use a volume, a new directory is created within Docker's storage directory on the host machine, and Docker manages that directory's contents. Volumes are easier to back up or migrate than bind mounts.



Bind mounts

- Bind mounts will mount a file or directory on to your container from your host machine, which you can then reference via its absolute path.
- Bind mounts have limited functionality compared to volumes. When you use a bind mount, a file or directory on the host machine is mounted into a container. The file or directory is referenced by its full path on the host machine. The file or directory doesn't need to exist on the Docker host already. It is created on demand if it doesn't yet exist. Bind mounts are fast, but they rely on the host machine's filesystem having a specific directory structure available. If you are developing new Docker applications, consider using named volumes instead. You can't use Docker CLI commands to directly manage bind mounts.

How to Create docker Volume

1. Create a volume : A Docker volume can be created and managed outside of a container by using the following command.

$ docker volume create [volume_name]

A directory for the volume is automatically created under the /var/lib/docker/volume/path folder by Docker on the host.

2. List the volumes : To list the volumes, run the following command.

$ docker volume list

3. **Mounting Docker Volumes :** After creating the Volume, the next step is to mount the Volume to Docker Containers. We will create a Docker Container with the Ubuntu base Image and mount the data Volume to that Container using the -v flag.

Option 1 : docker run -it -v volumename:/shared-volume --name my-container-01 ubuntu

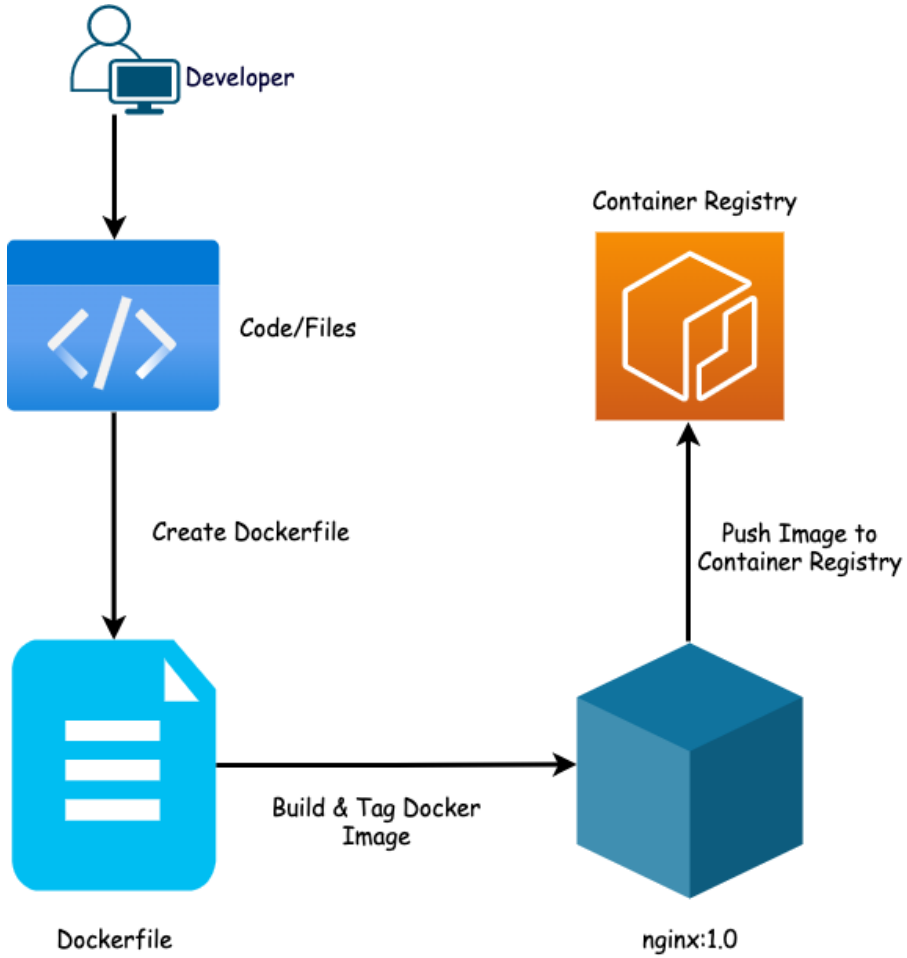
Option 2: docker run --mount source=[volume_name],destination=[path_in_container] [docker_image]

Docker Images

- Docker images are lightweight, standalone, and executable packages that contain everything needed to run an application, including the code, runtime, libraries, and system tools.
- We can create docker images by two way.

1. Using Running Containers

2. Using Docker file



The Process of Creating Images From Containers

At a high level, the process of creating a Docker image from a container involves three steps:

1.**Create a container from an existing image:** The first step is to choose a base image you want to customize and run it as a container.

2.**Make changes to the container:** Once you have the container up and running, you make changes to it. You could modify files, install additional software or do whatever you need to meet your requirements.

3.**Commit the changes to create a new image:** After you've made the desired changes to the container, the next step is to commit those changes to create a brand-new Docker image. Now, you can use the new image to spin up new containers with your customizations.

command

```
docker container commit -a "sandeep" -m "Changed default c1  welcome message" c1  image1
```

Note that it's considered best practice to use the -a flag to sign the image with an author and include a commit message using the -m flag.

Docker File

- Docker builds images automatically by reading the instructions from a Dockerfile which is a text file that contains all commands, in order, needed to build a given image.
- A Docker image consists of read-only layers each of which represents a Dockerfile instruction
- The layers are stacked and each one is a delta of the changes from the previous layer.
- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Docker file is a text document that contains all the commands a user could call on the command line to assemble an image.
- A Dockerfile is a plain text file that contains instructions for building a Docker container image.
- Docker containers are lightweight, portable, and consistent environments that encapsulate an application and its dependencies.
- The default filename to use for a Dockerfile is Dockerfile, without a file extension.
- Using the default name allows you to run the docker build command without having to specify additional command flags

Docker File Syntax

```
# syntax=docker/dockerfile:1


FROM ubuntu:22.04

COPY . /app

RUN make /app

CMD python /app/app.py
```

Docker File Syntax

A Dockerfile consists of a series of instructions that define how to build a Docker image.

Instruction	Description
FROM	Specifies the base image to use for the build.
MAINTAINER (deprecated)	Specifies the author/maintainer of the image (deprecated, use LABEL instead).
LABEL	Adds metadata to the image, typically used for documentation or identification purposes.
RUN	Executes commands in a new layer on top of the current image and commits the results.
CMD	Provides default arguments for the ENTRYPOINT instruction or sets the default command.
EXPOSE	Informs Docker that the container listens on specified network ports at runtime.
ENV	Sets environment variables in the form key=value.
ADD	Copies files, directories, or remote URLs from source to destination in the image.
COPY	Copies files or directories from source to destination in the image.
ENTRYPOINT	Configures a container that will run as an executable when started.
VOLUME	Creates a mount point for externally mounted volumes or other containers.
USER	Sets the username or UID to use when running the image.
WORKDIR	Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions.

Create Image Using Docker File

Step 1: Create a Dockerfile: Write a Dockerfile that defines the instructions for building your image. Place this file in the root directory of your project.

Step 2: Build the Docker Image: Open a terminal, navigate to the directory containing your Dockerfile, and run the following command to build the Docker image.

docker build -t yourimage.

Docker Networking

- A network is a group of two or more devices that can communicate with each other either physically or virtually
- Docker Networking allows you to create a Network of Docker Containers managed by a master node called the manager. Containers inside the Docker Network can talk to each other by sharing packets of information.
- The Docker network is a virtual network created by Docker to enable communication between Docker containers. If two containers are running on the same host they can communicate with each other without the need for ports to be exposed to the host machine. You may use Docker to manage your Docker hosts in any platform manner, regardless of whether they run Windows, Linux, or a combination of the two.

Docker Network Driver

- **Bridge:** If you build a container without specifying the kind of driver, the container will only be created in the bridge network, which is the default network.
- **Host:** Containers will not have any IP address they will be directly created in the system network which will remove isolation between the docker host and containers.
- **None:** IP addresses won’t be assigned to containers. These containments are not accessible to us from the outside or from any other container.

Docker Network

Docker network : The Docker Network command is the main command that would allow you to create, manage, and configure your Docker Network.

docker network ls : To list all the Docker Networks, you can use the list command.

Docker Network Create command : With the help of the “Create” command, we can create our own docker network and can deploy our containers in it.

docker network create --driver <driver-name> <bridge-name>

Using the Docker Network Connect command : Using the “Connect” command, you can connect a running Docker Container to an existing Network.

docker network connect <network-name> <container-name or id>

Using the Docker Network Disconnect command : The disconnect command can be used to remove a Container from the Network.

docker network disconnect <network-name> <container-name or id>

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

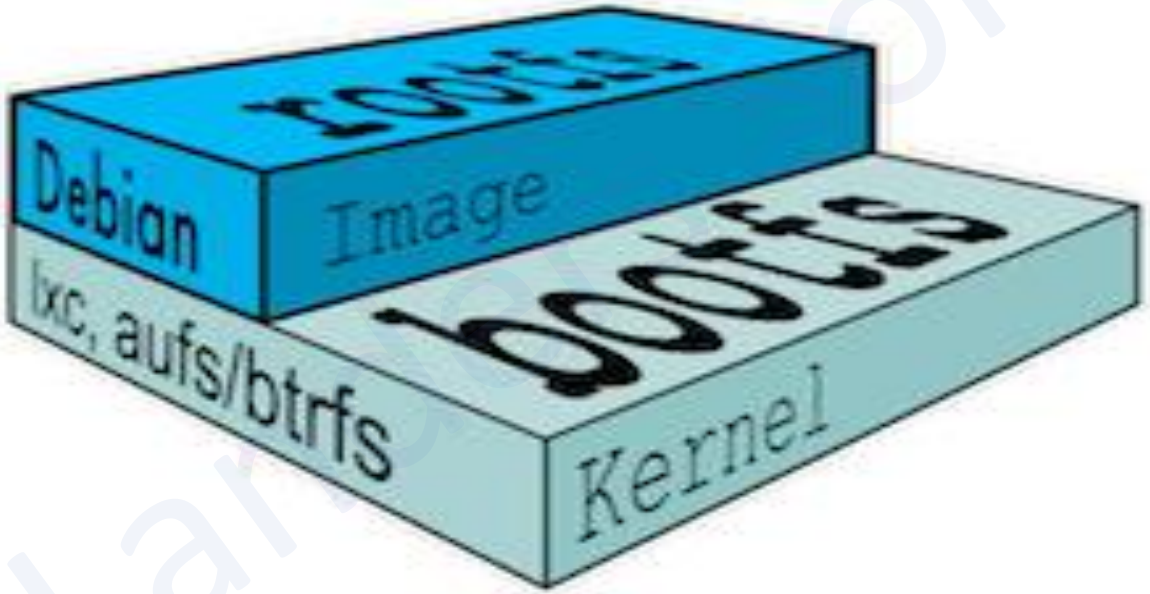
Docker Ports and Volumes

- 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



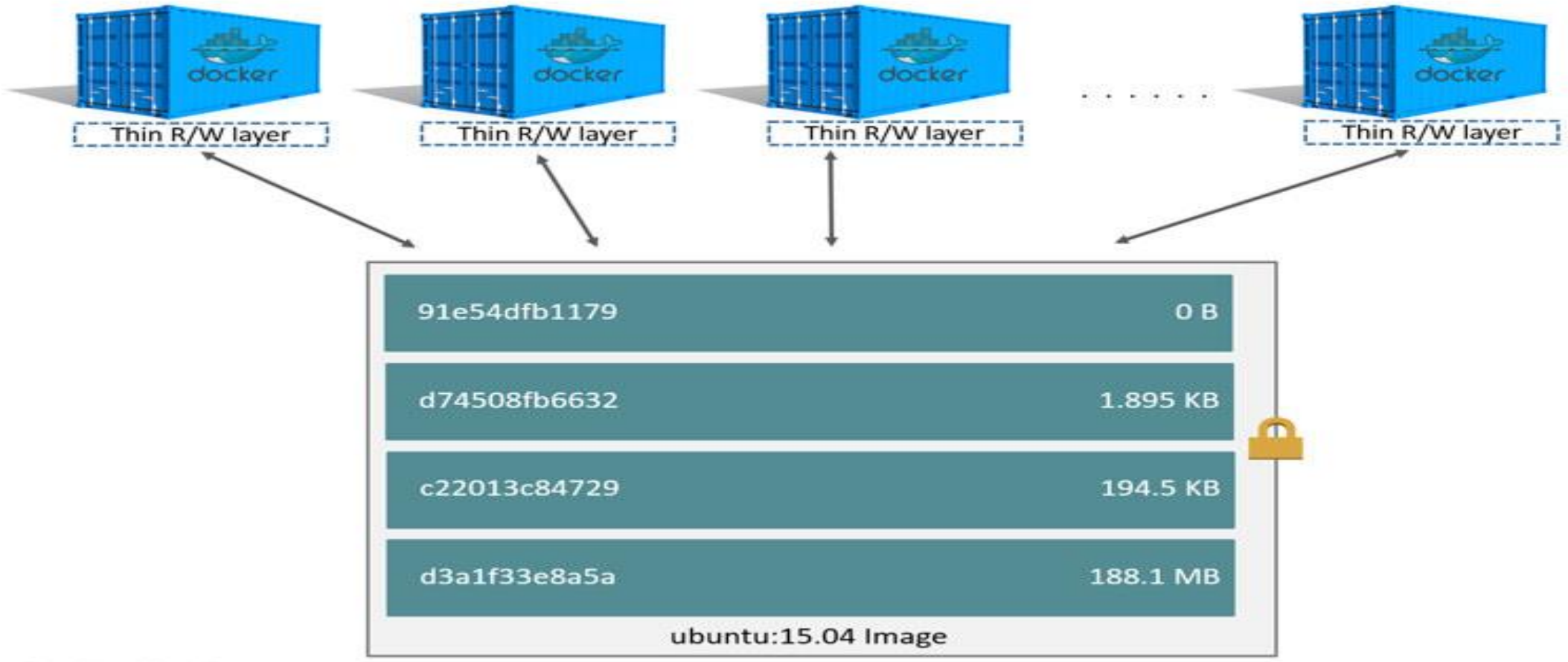
What is an image?

- 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 “sandeep" c1 sandeeptosimage: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 sandeeptosimage:v1

  rpm –qa | grep –i httpd

- Tagging images:

  docker tag <newImageId> newhttpd : docker tag sandeeptosimage: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 " 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 " 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

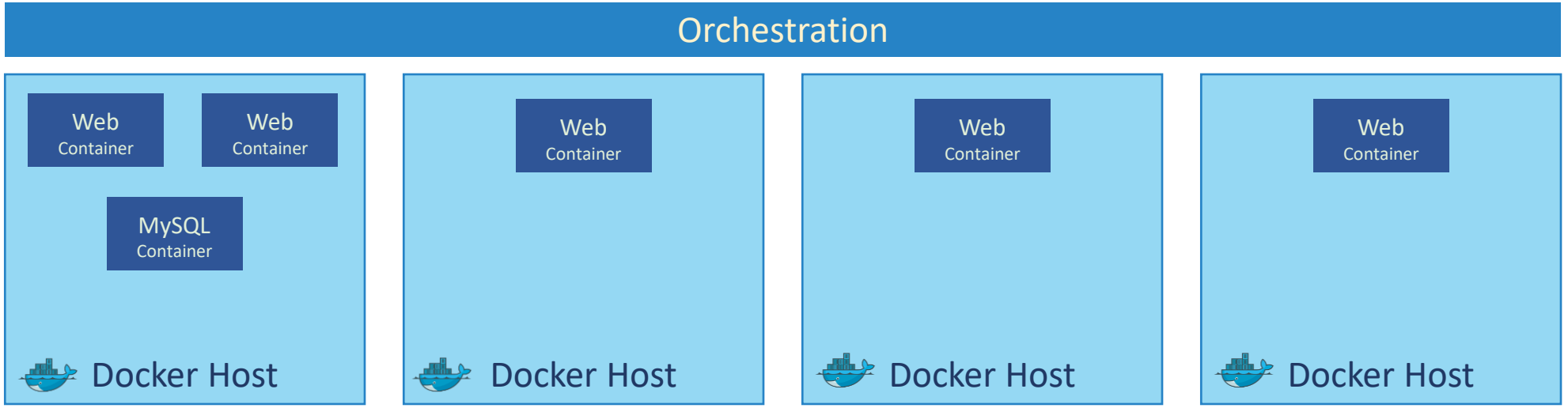
- Kubernetes can schedule and run application containers on clusters of physical or virtual machines.
- **host-centric** infrastructure to a **container-centric** infrastructure.
- Orchestrator
- Load balancing
- Auto Scaling
- Application Health checks
- Rolling updates

Container Orchestration

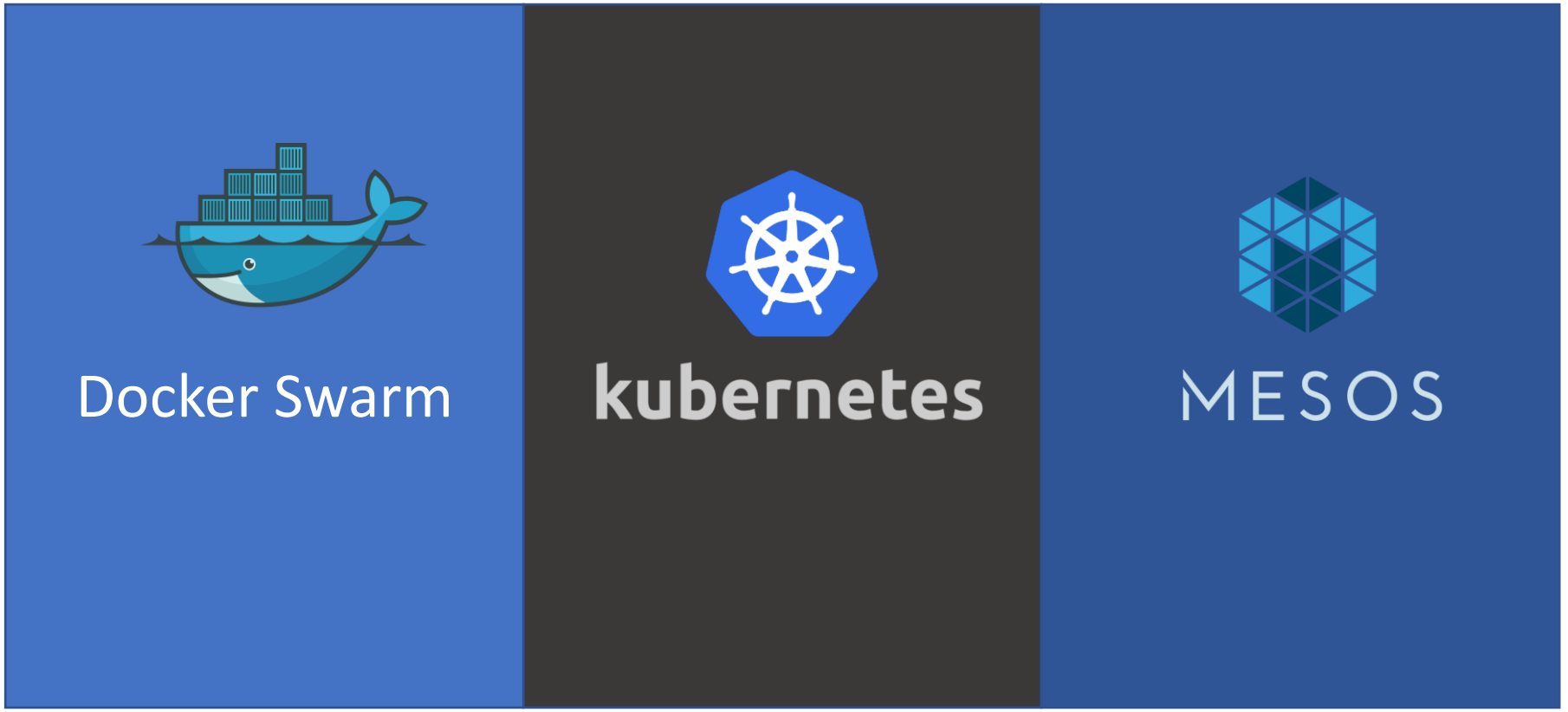
Containers Limitation?

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

Container orchestration



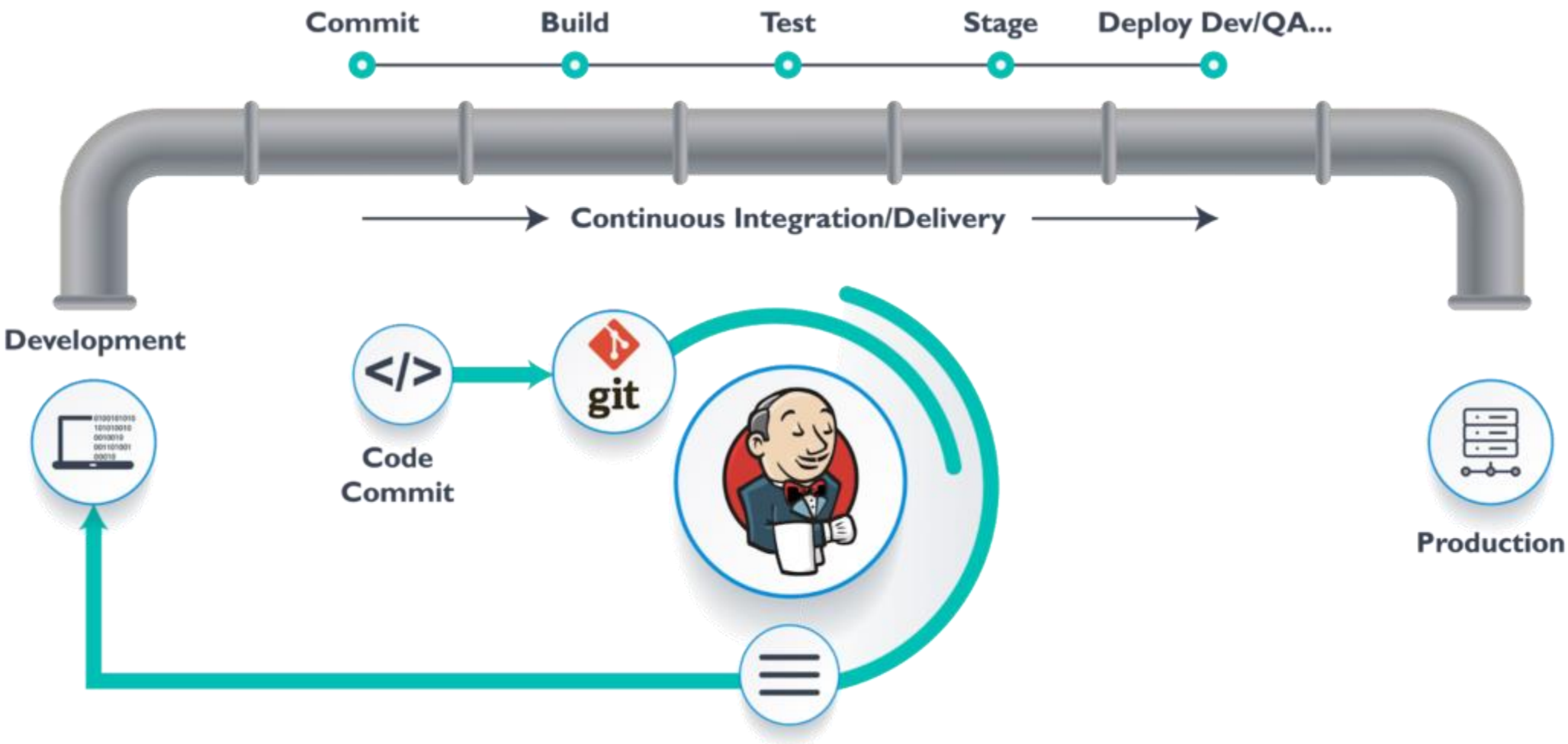
Orchestration Technologies



Jenkins

- Jenkins is an open source automation tool written in Java programming language that allows continuous integration.
- Jenkins builds and tests our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.
- It also allows us to continuously deliver our software by integrating with a large number of testing and deployment technologies.

Jenkins Workflow



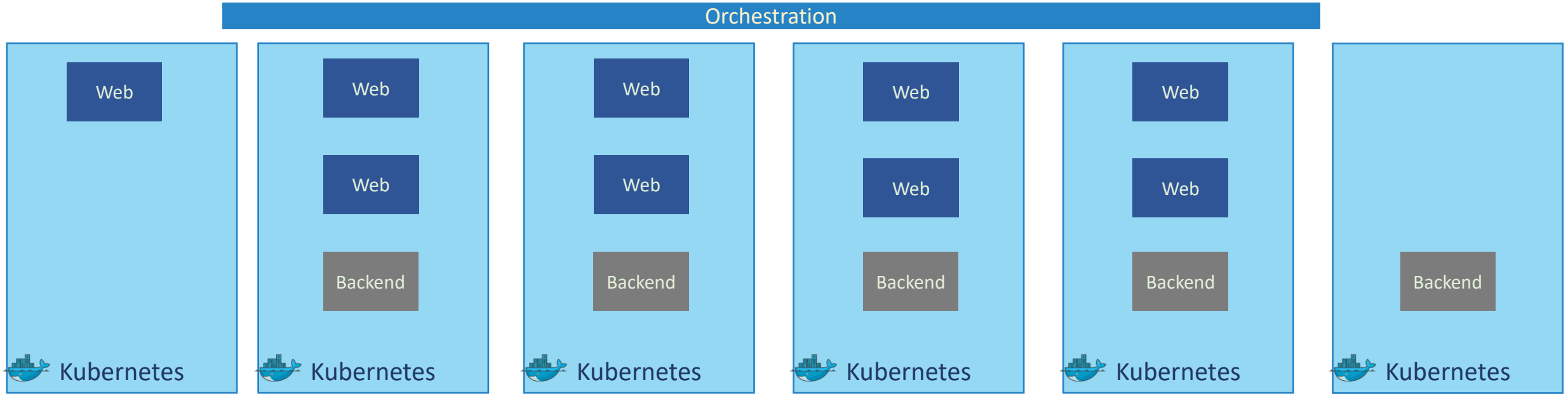
What is Kubernetes?

- 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

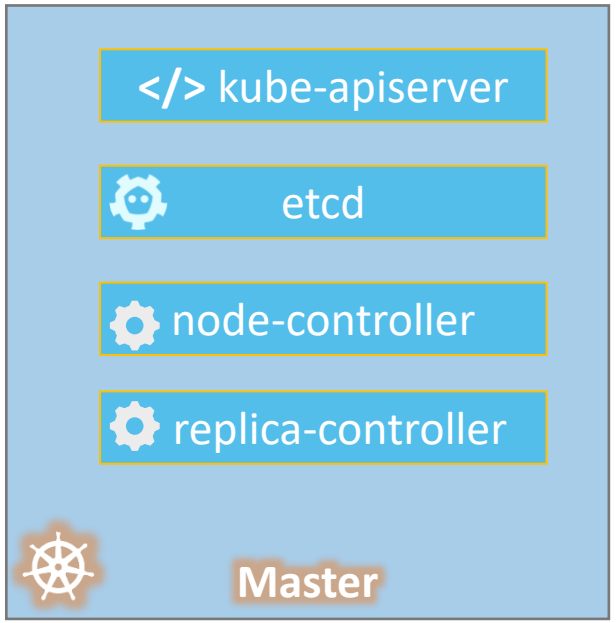
Kubeadm

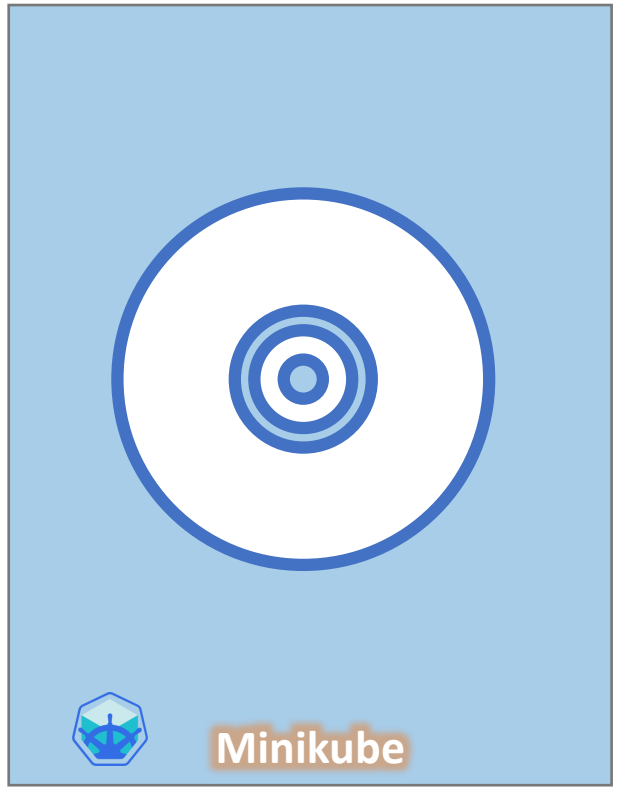
Google Cloud Platform

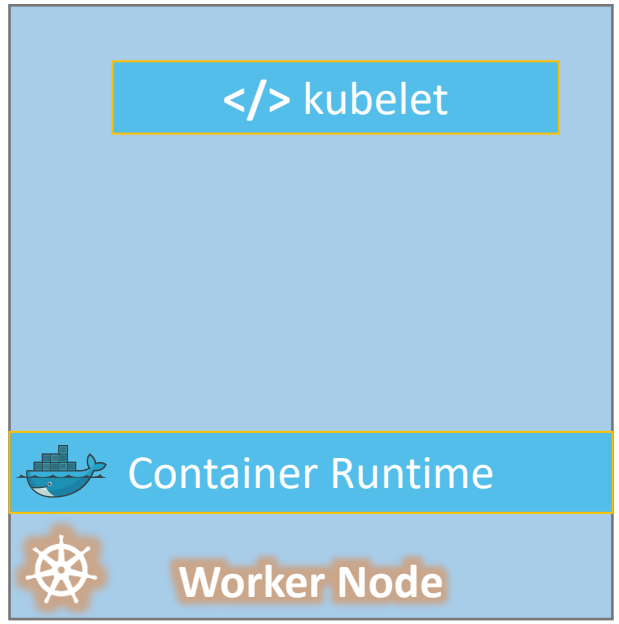
Amazon Web Services

play-with-k8s.com

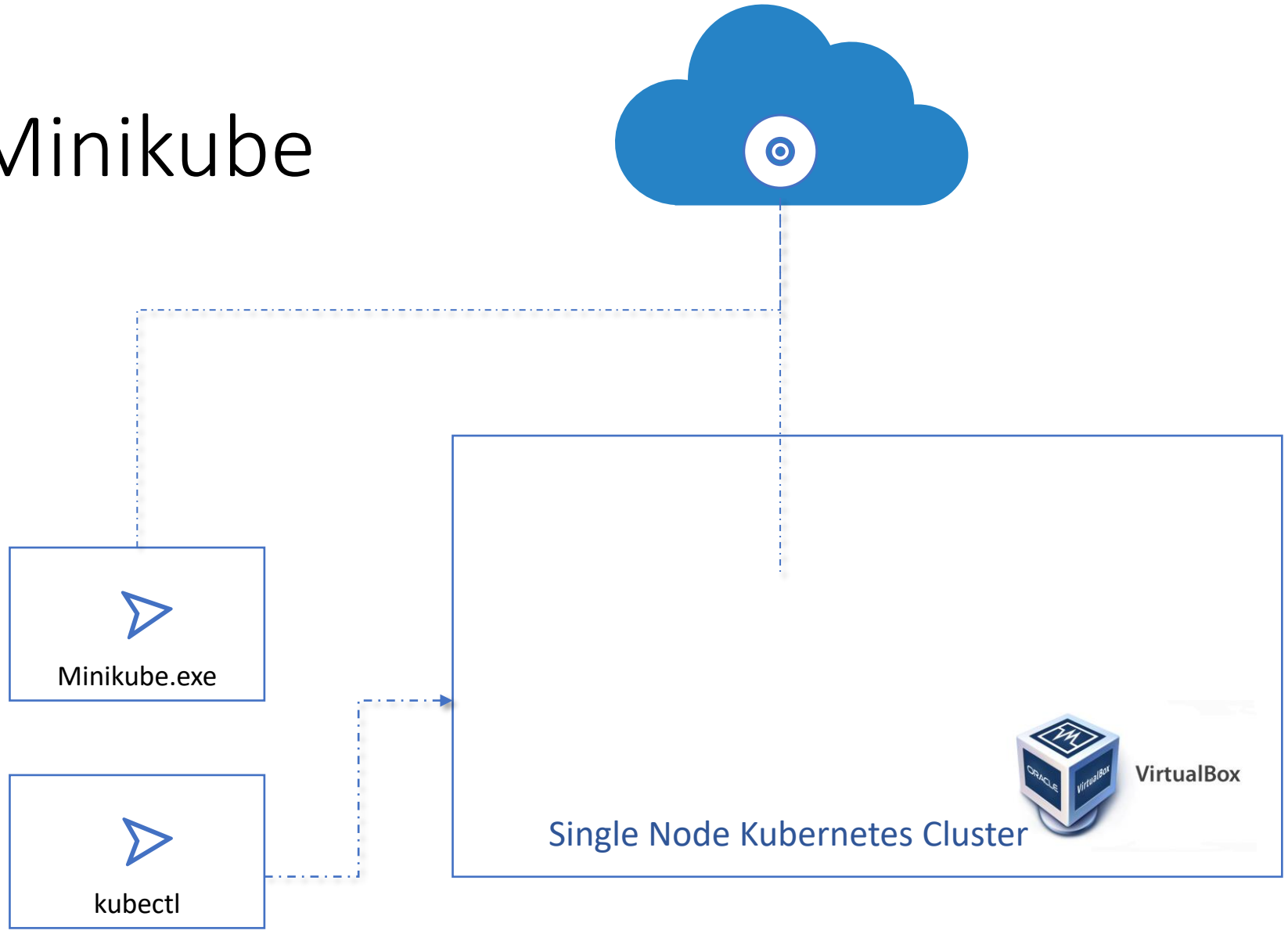
Minikube


kube-apiserver
etcd
node-controller
replica-controller
Master


Minikube


kubelet
Container Runtime
Worker Node

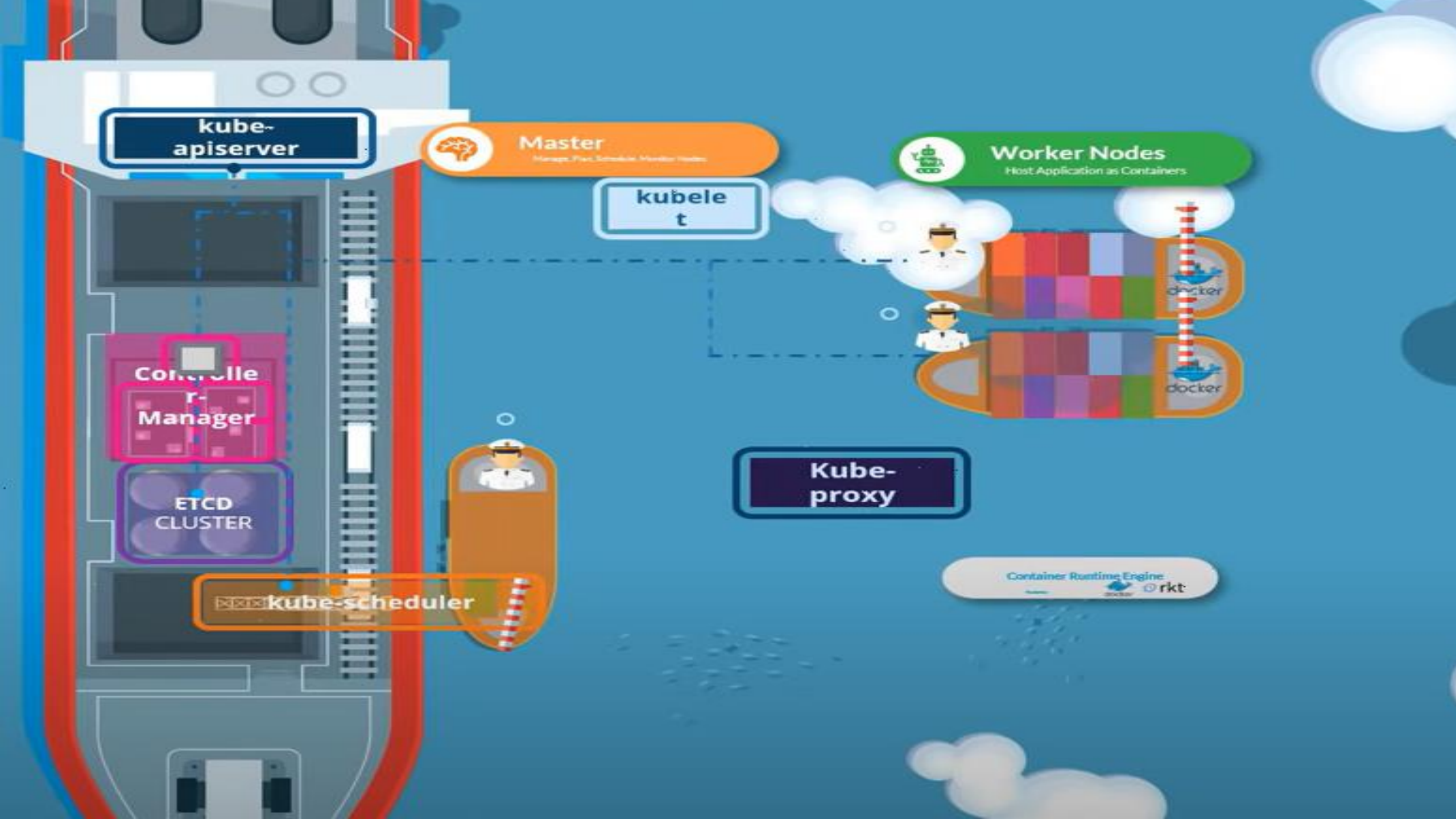
Minikube


Minikube.exe
kubectl
Single Node Kubernetes Cluster
VirtualBox

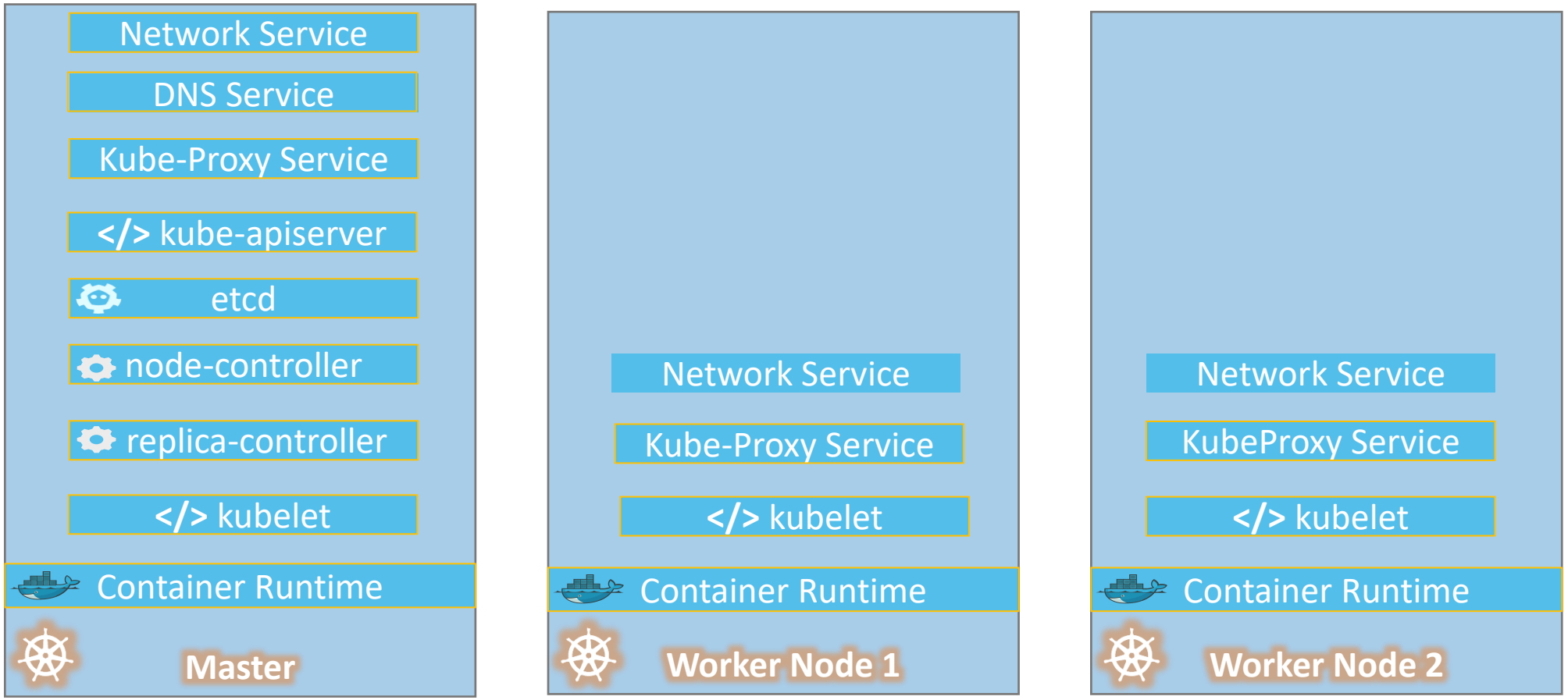
Setup - kubeadm

Kubernetes Cluster

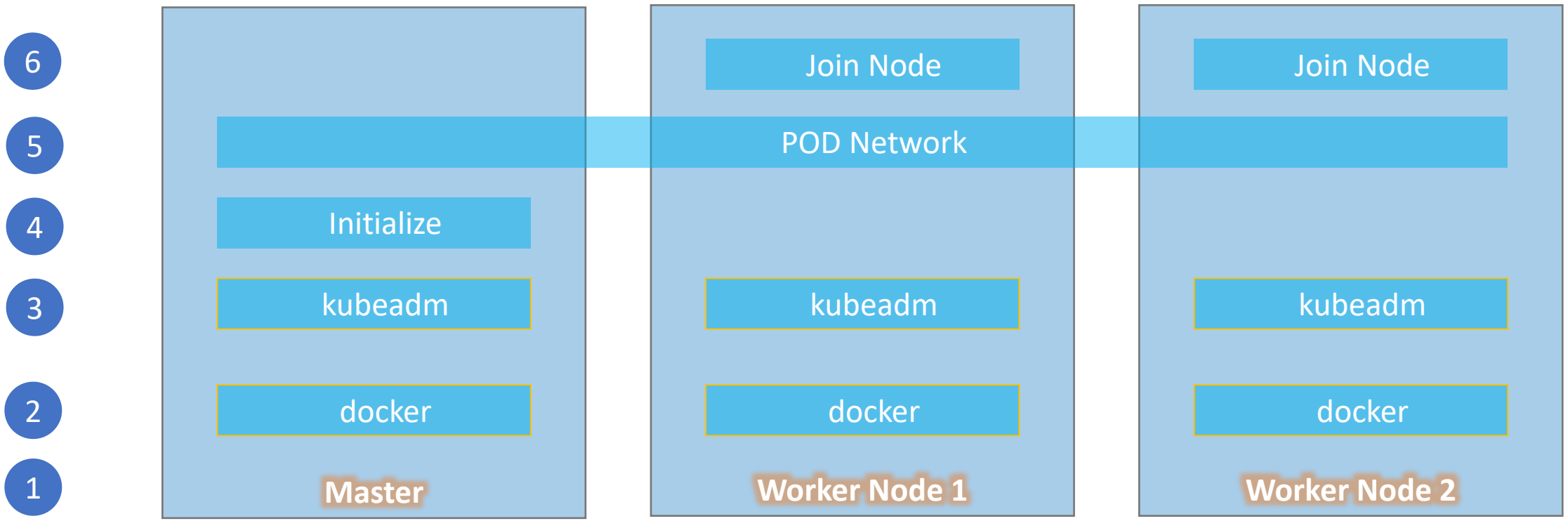
- A Kubernetes cluster consists of two types of resources:
- **Master**: Which coordinates with the cluster
- The Master is responsible for managing the cluster. The master coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, 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

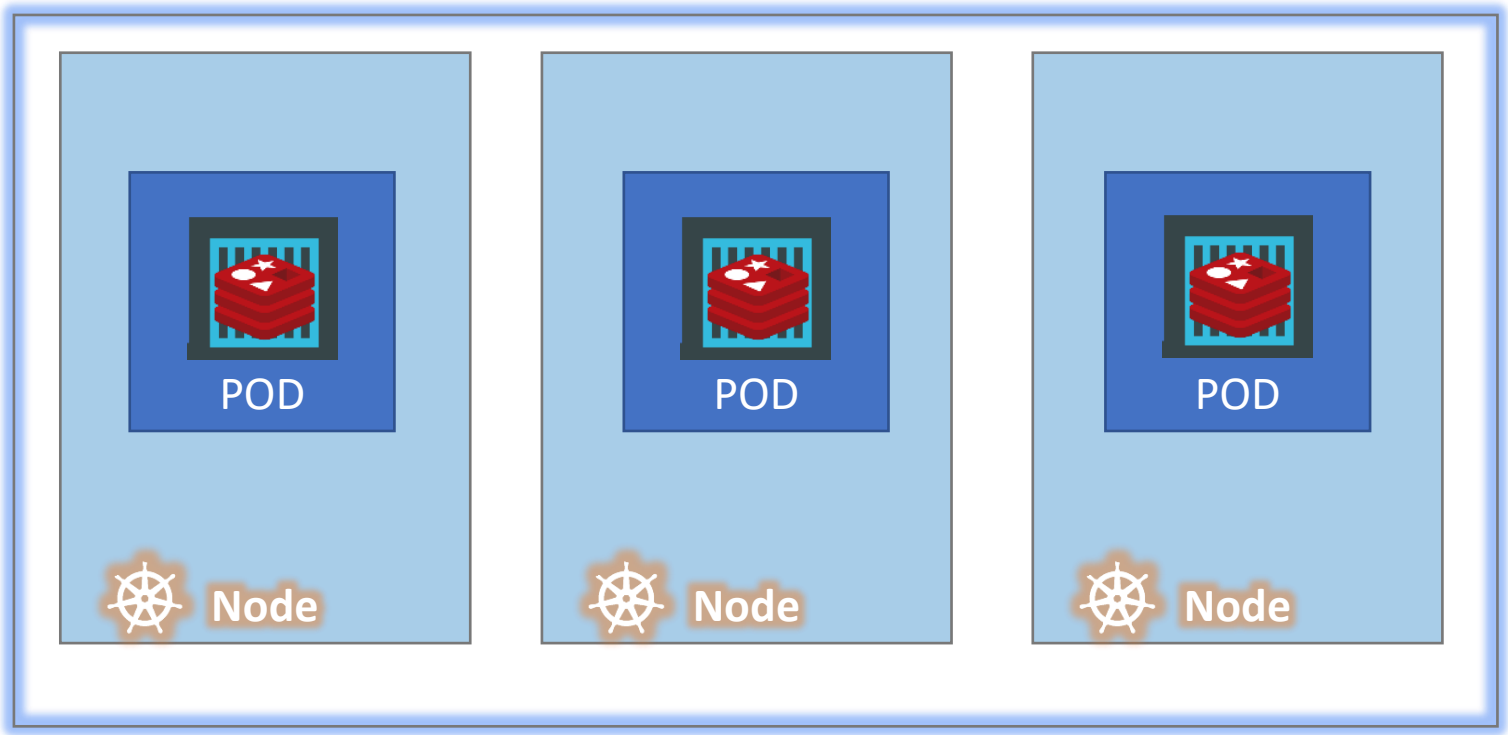
POD

Assumptions

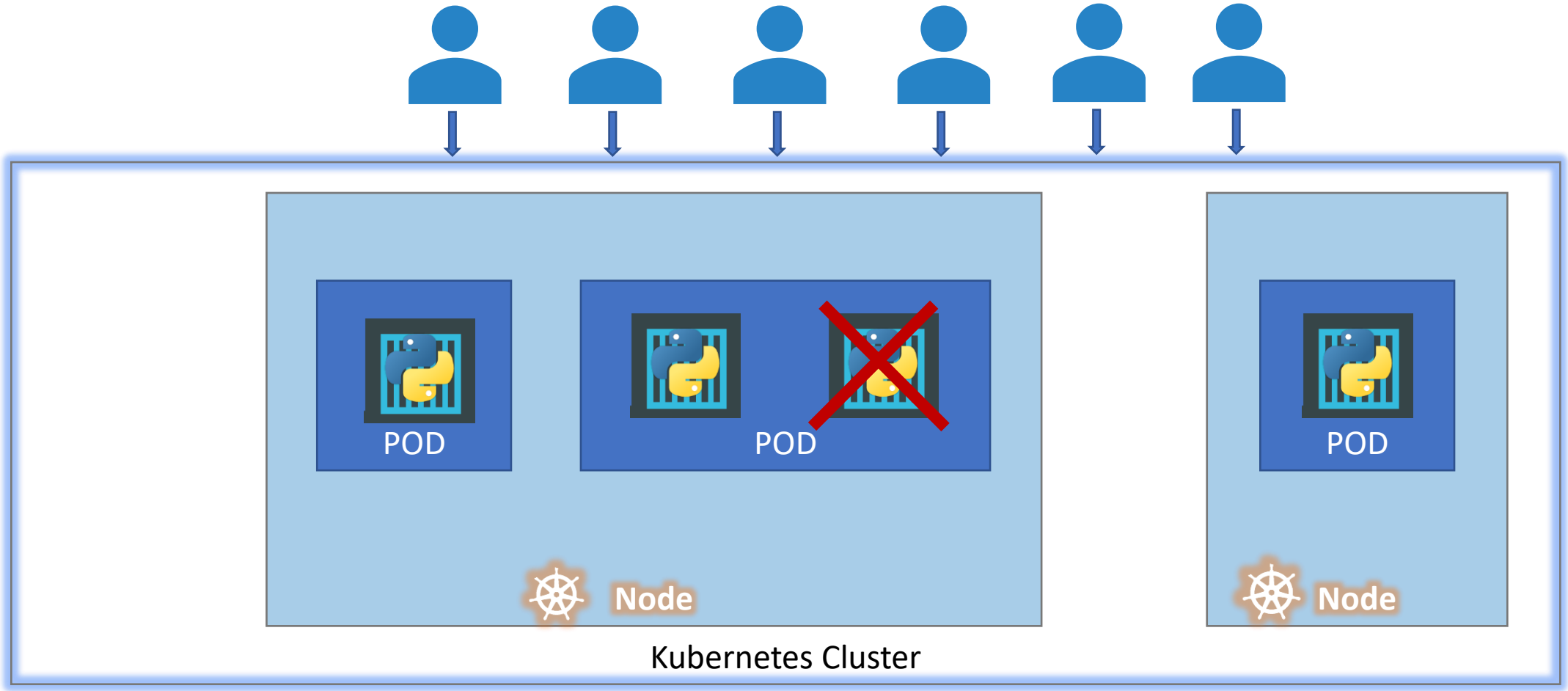
Docker Image

Kubernetes Cluster

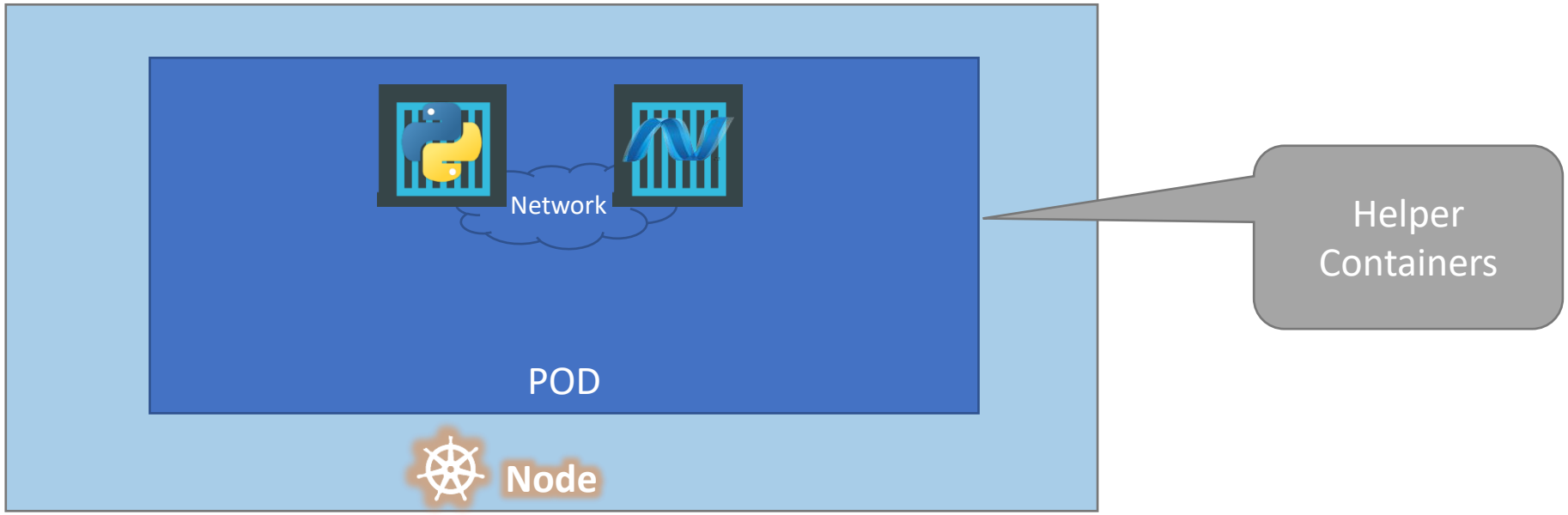
POD



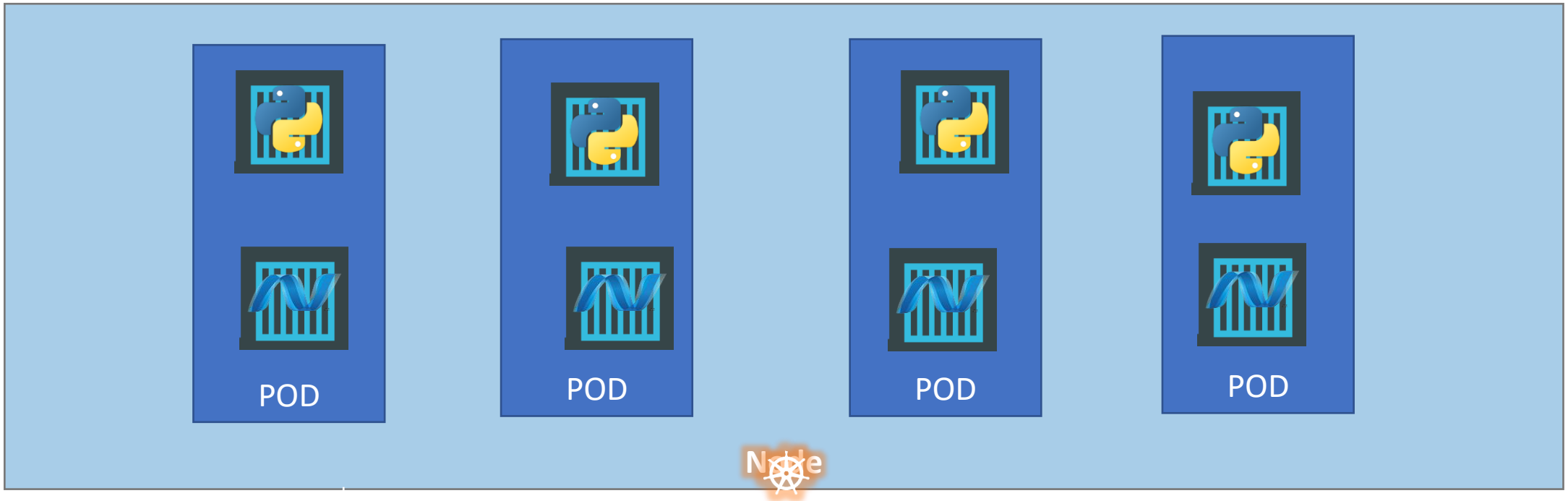
POD



Multi-Container PODs



PODs Again!



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

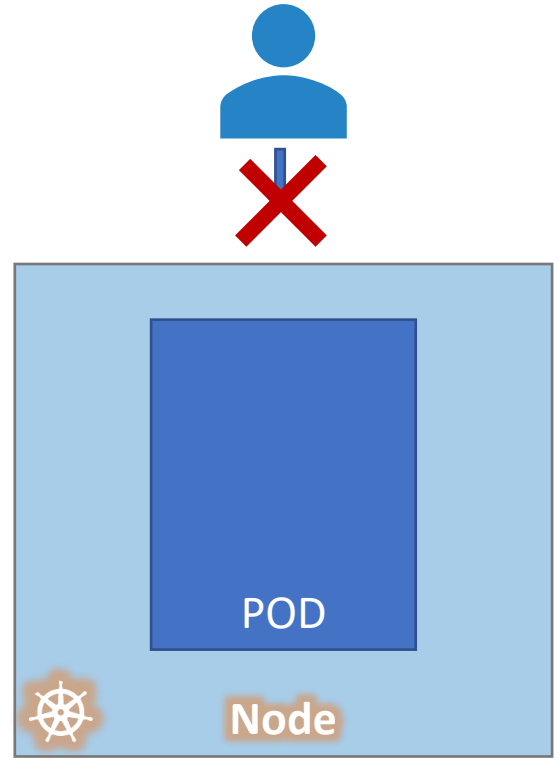
kubectl

- `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

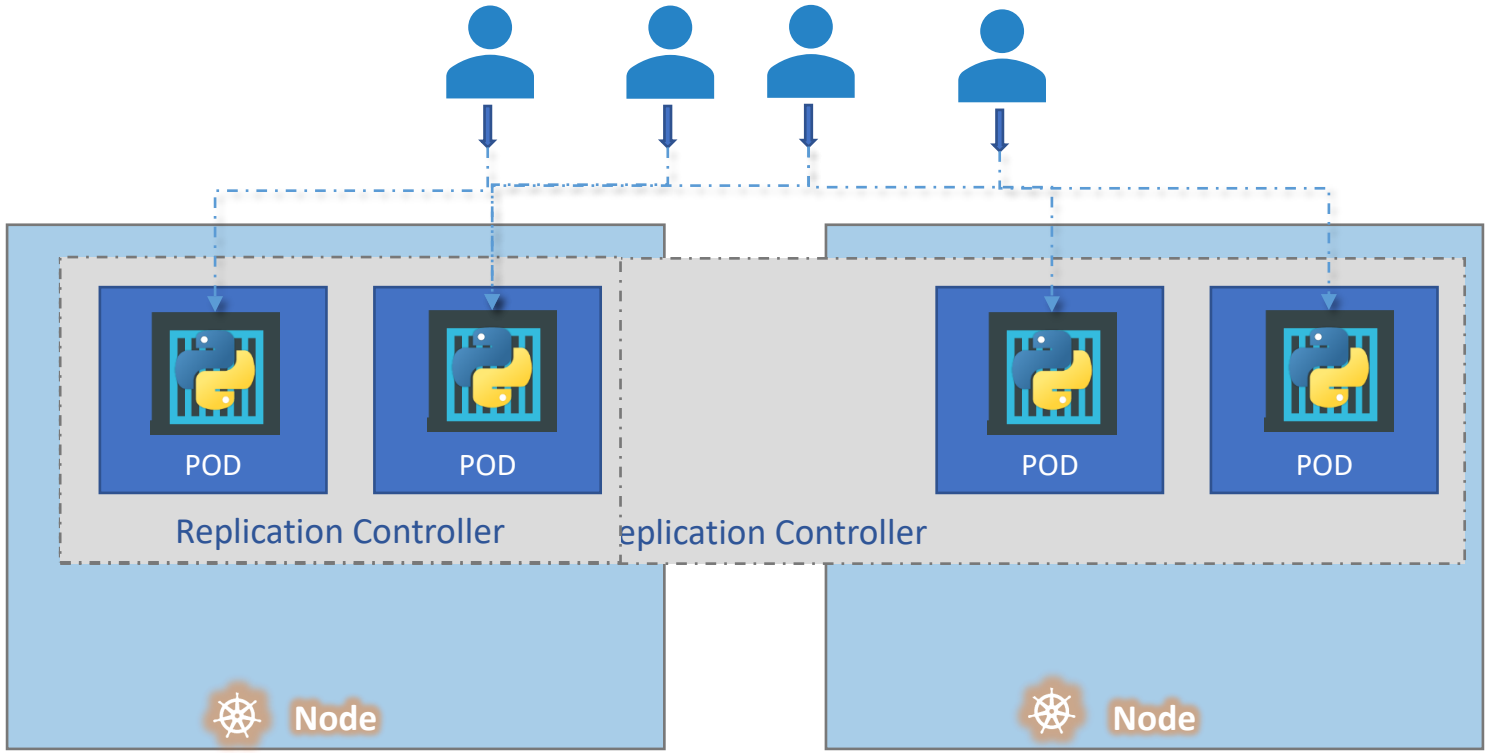
```
> kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
myapp-pod       1/1       Running   0          20s
```

```
> 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

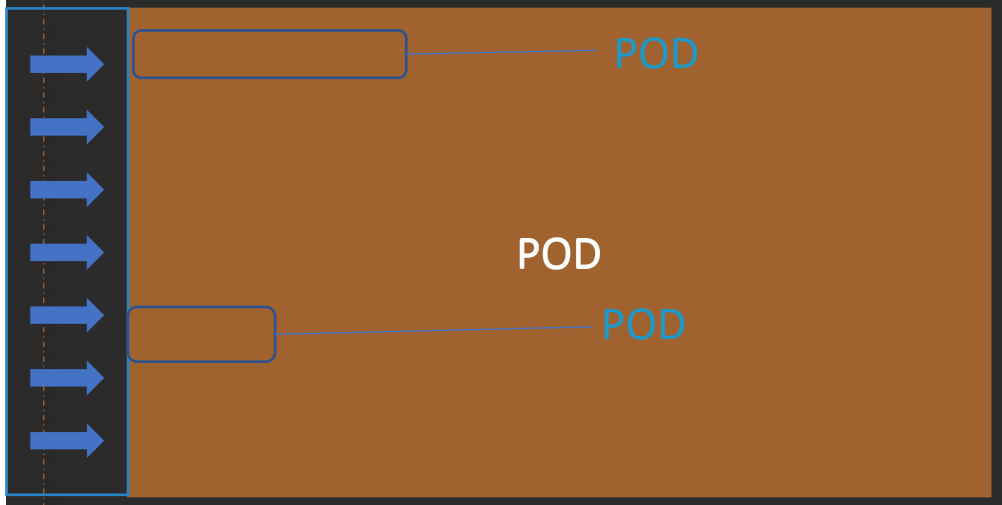
Replica Set

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:
      POD
  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
```

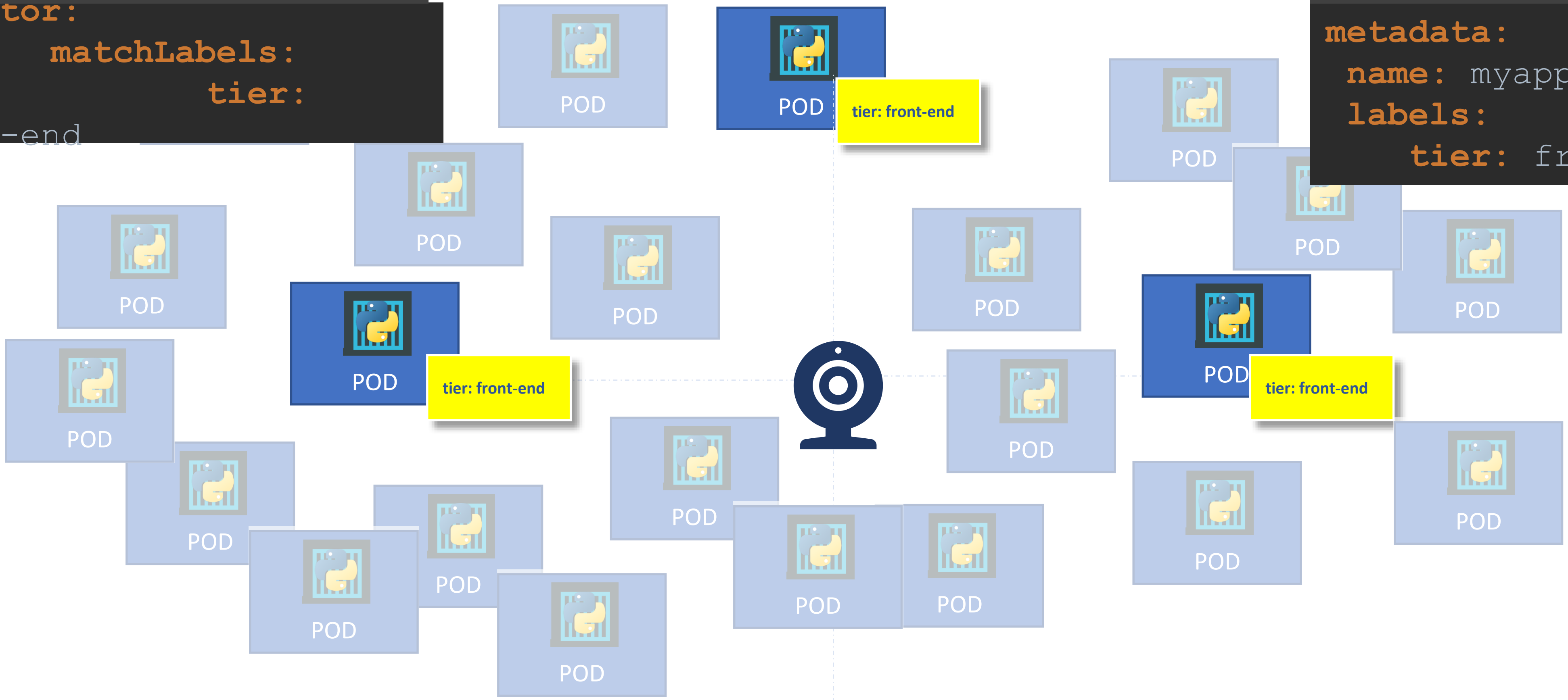
Labels and Selectors

replicaset-definition.yml

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

pod-definition.yml

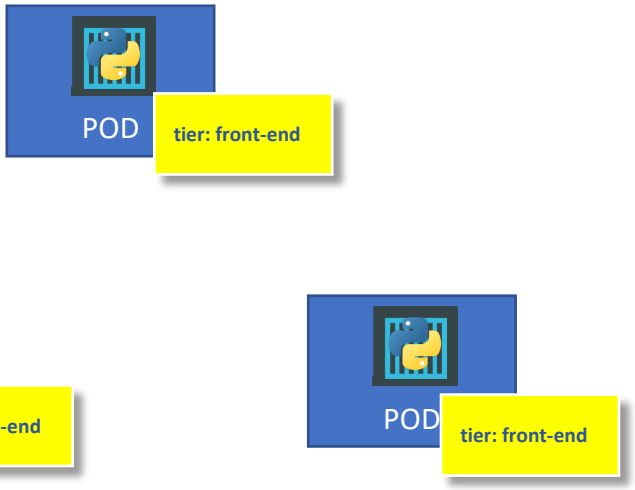
```yaml
metadata:
  name: myapp-pod
  labels:
     tier: front-end
```



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

```
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
```

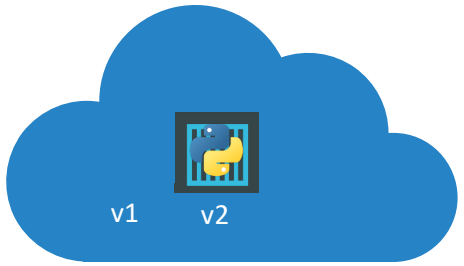
```
> kubectl scale –replicas=6 -f replicaset-definition.yml
```

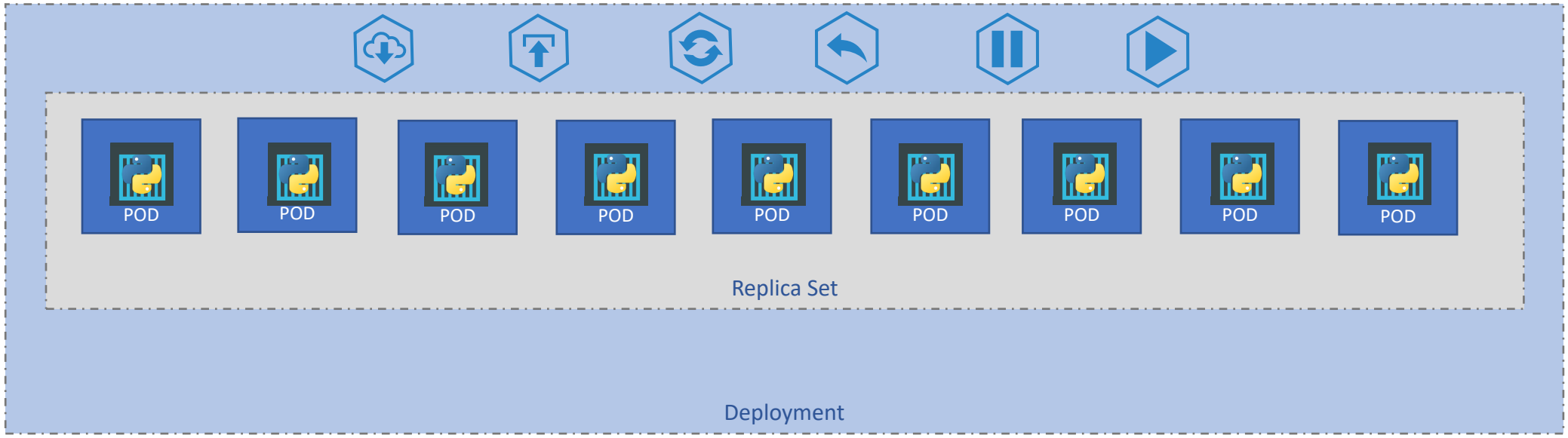
Demo

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

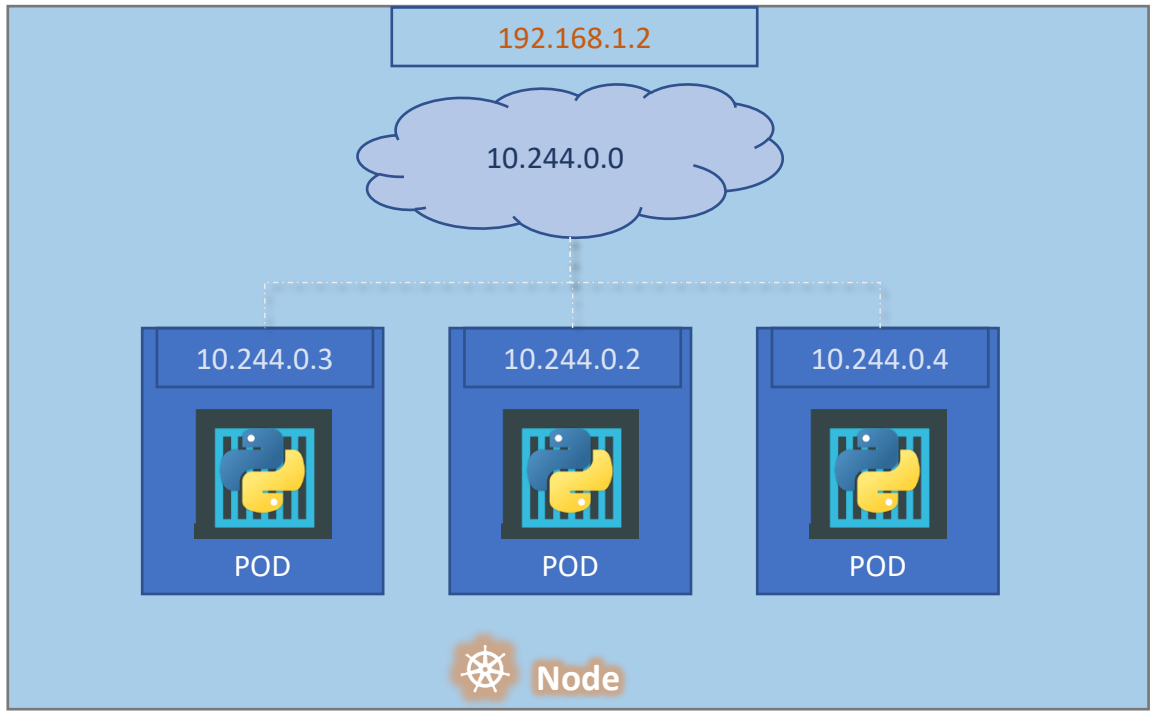
Demo

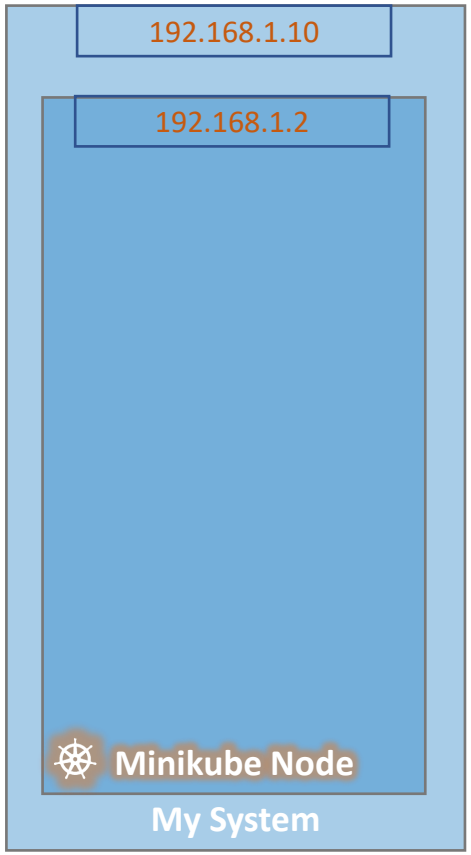
Deployment

Networking 101

Kubernetes Networking - 101

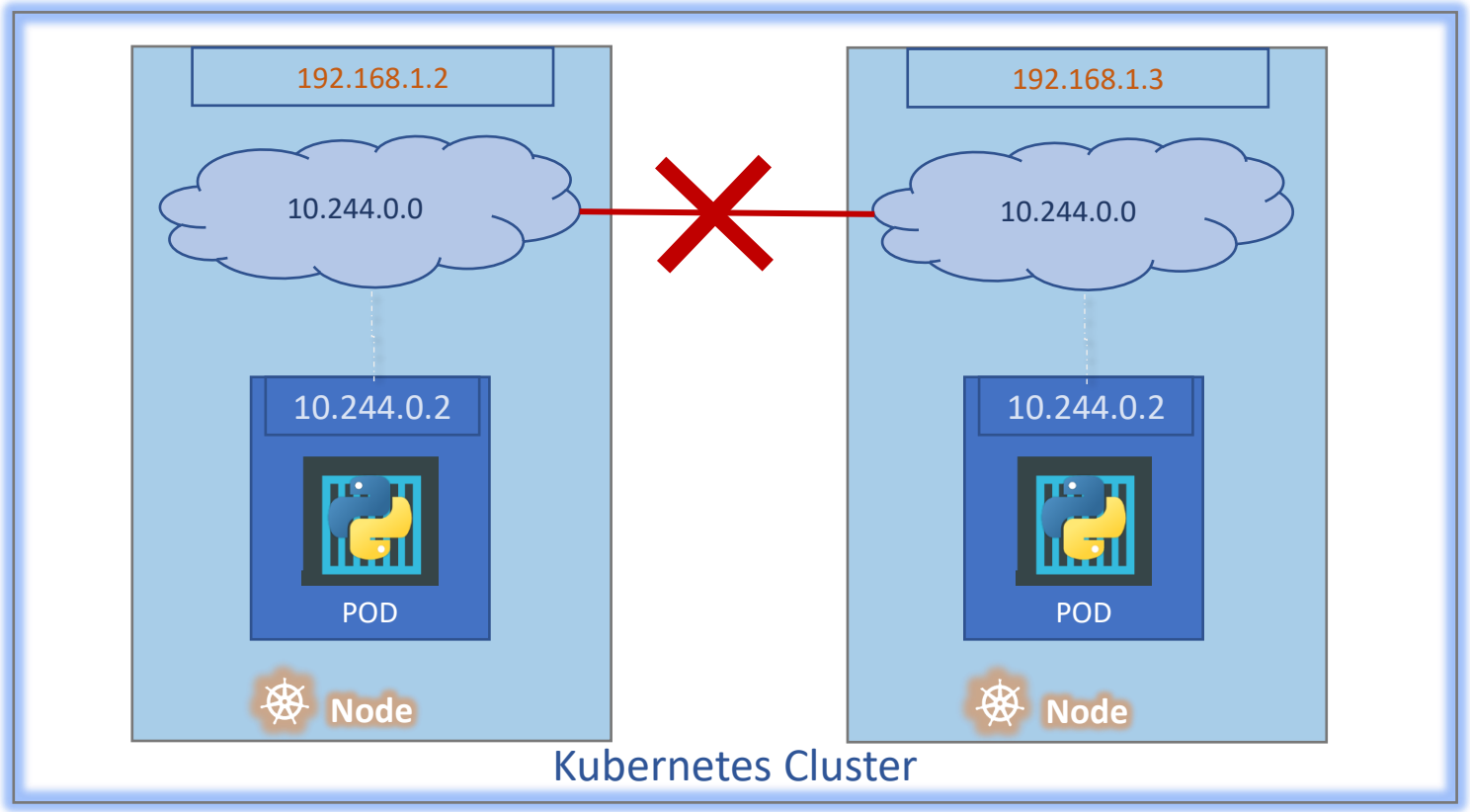
- IP Address is assigned to a POD





Cluster Networking

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

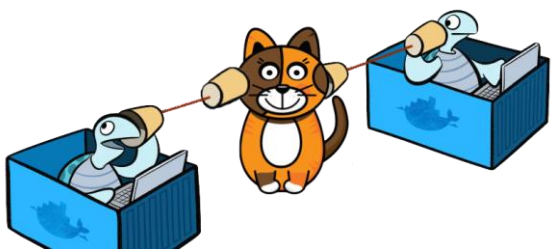



CISCO


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

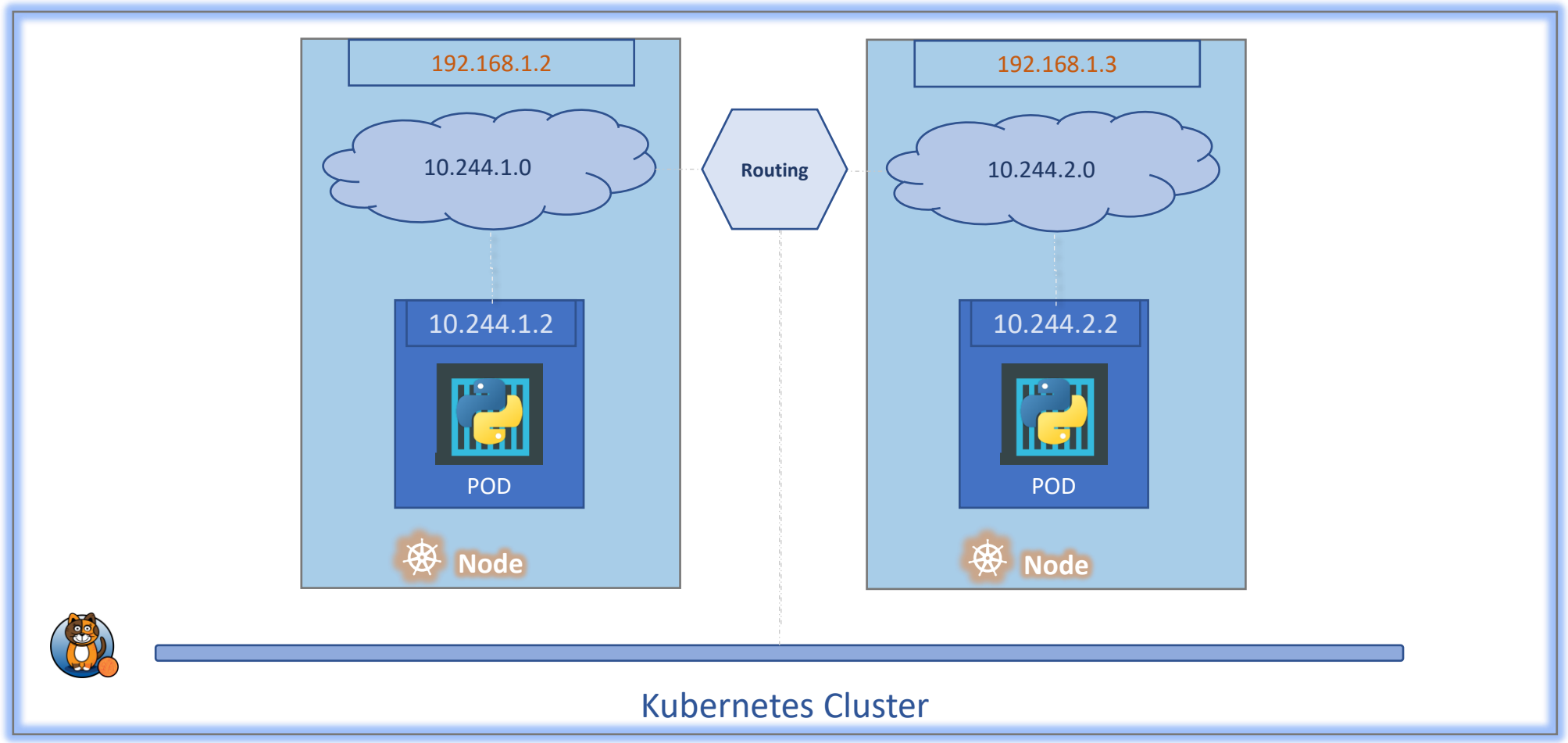
Refer to the Calico documentation for a kubeadm quickstart, a kubeadm installation guide, and other resources.

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
```

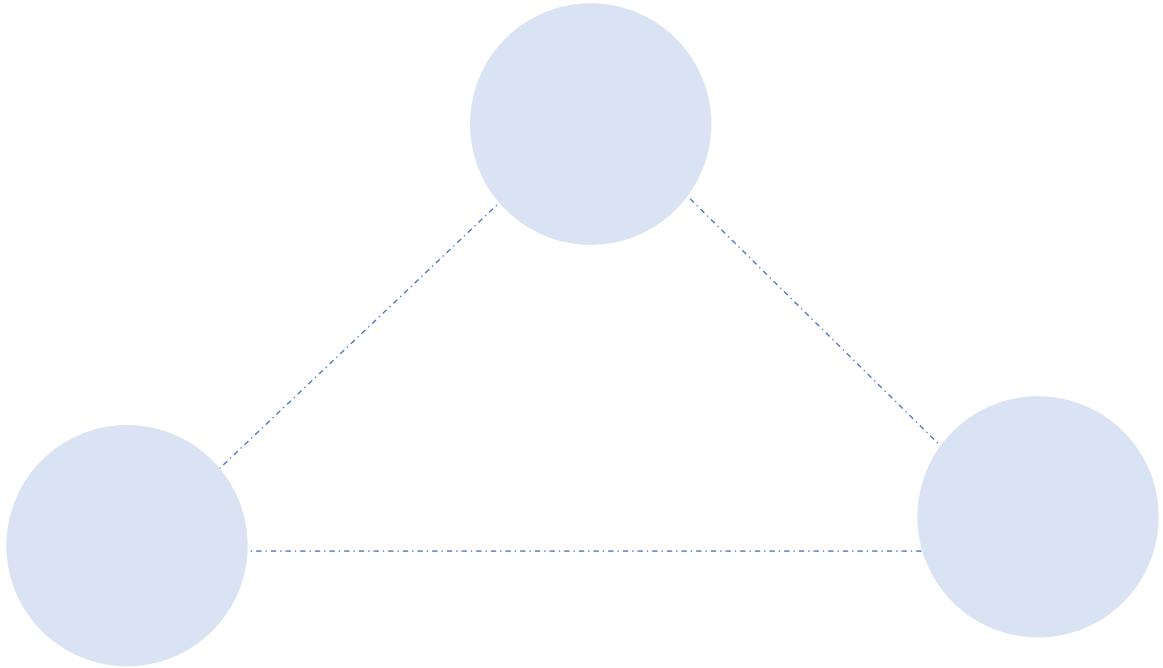
Cluster Networking



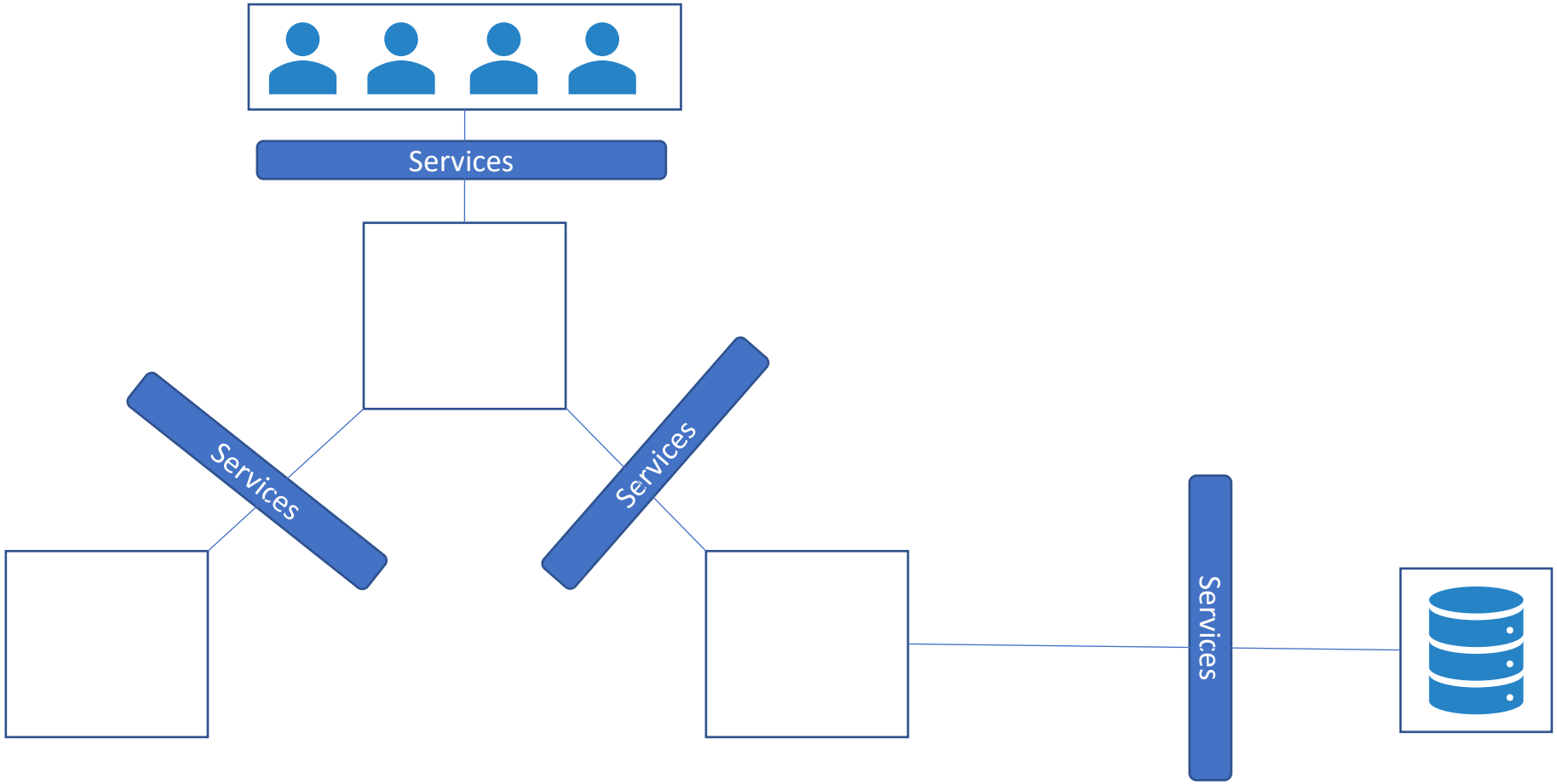
Demo

Networking

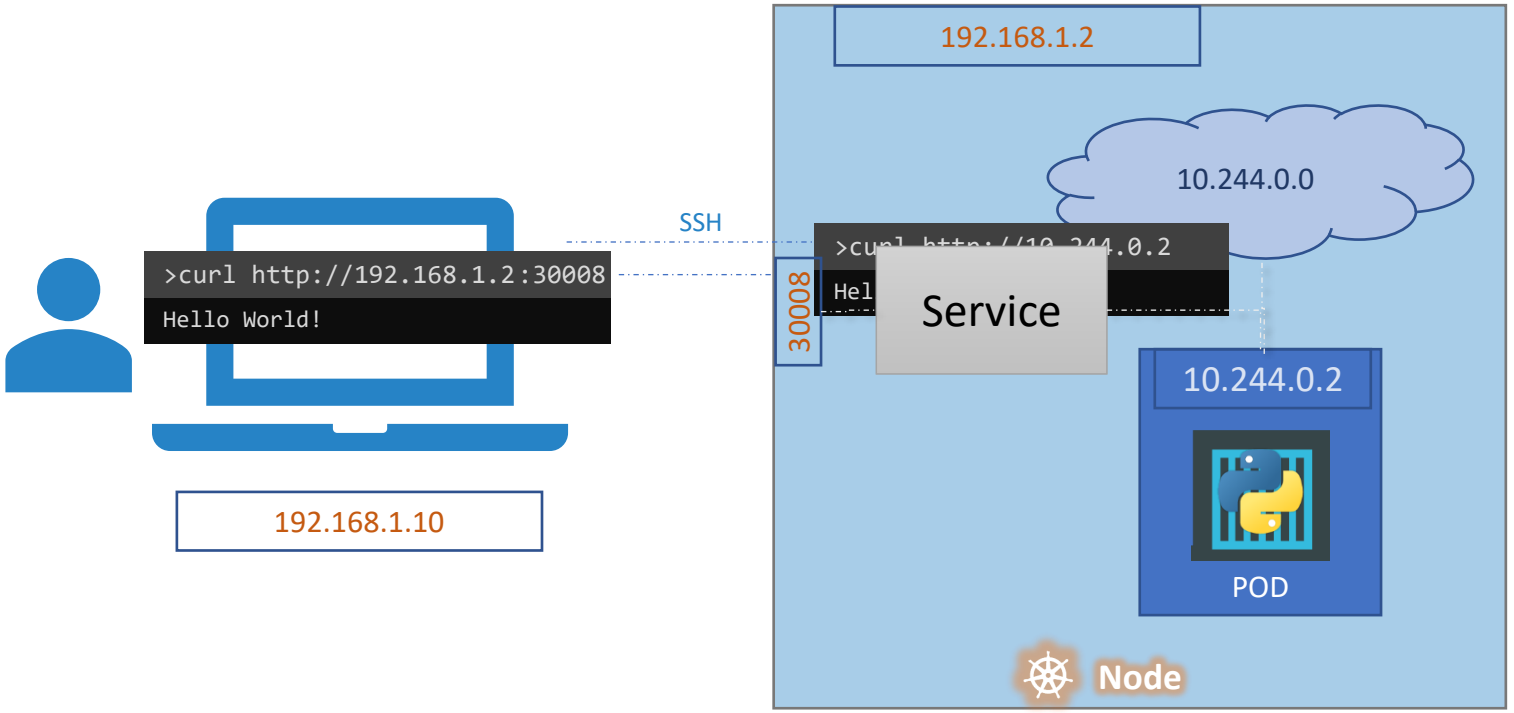
Services



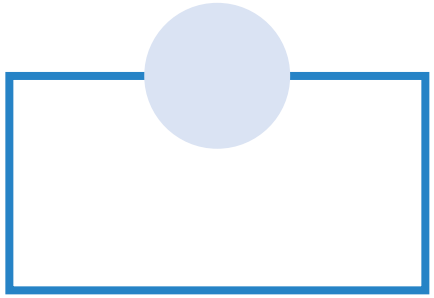
Services



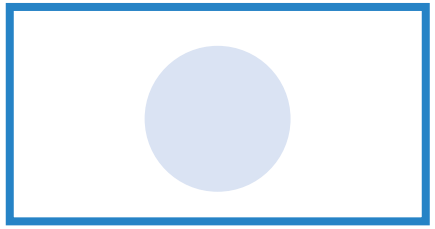
Service


>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

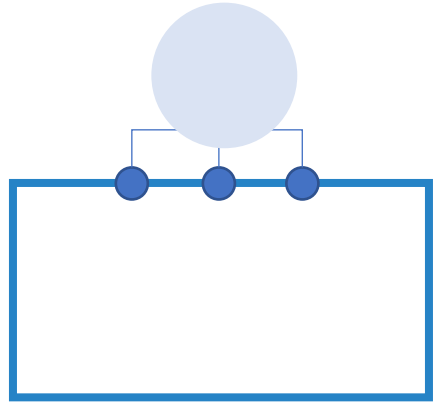
Services Types



NodePort

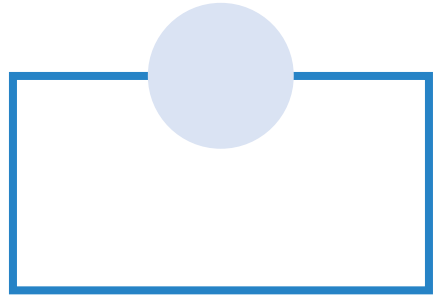


ClusterIP

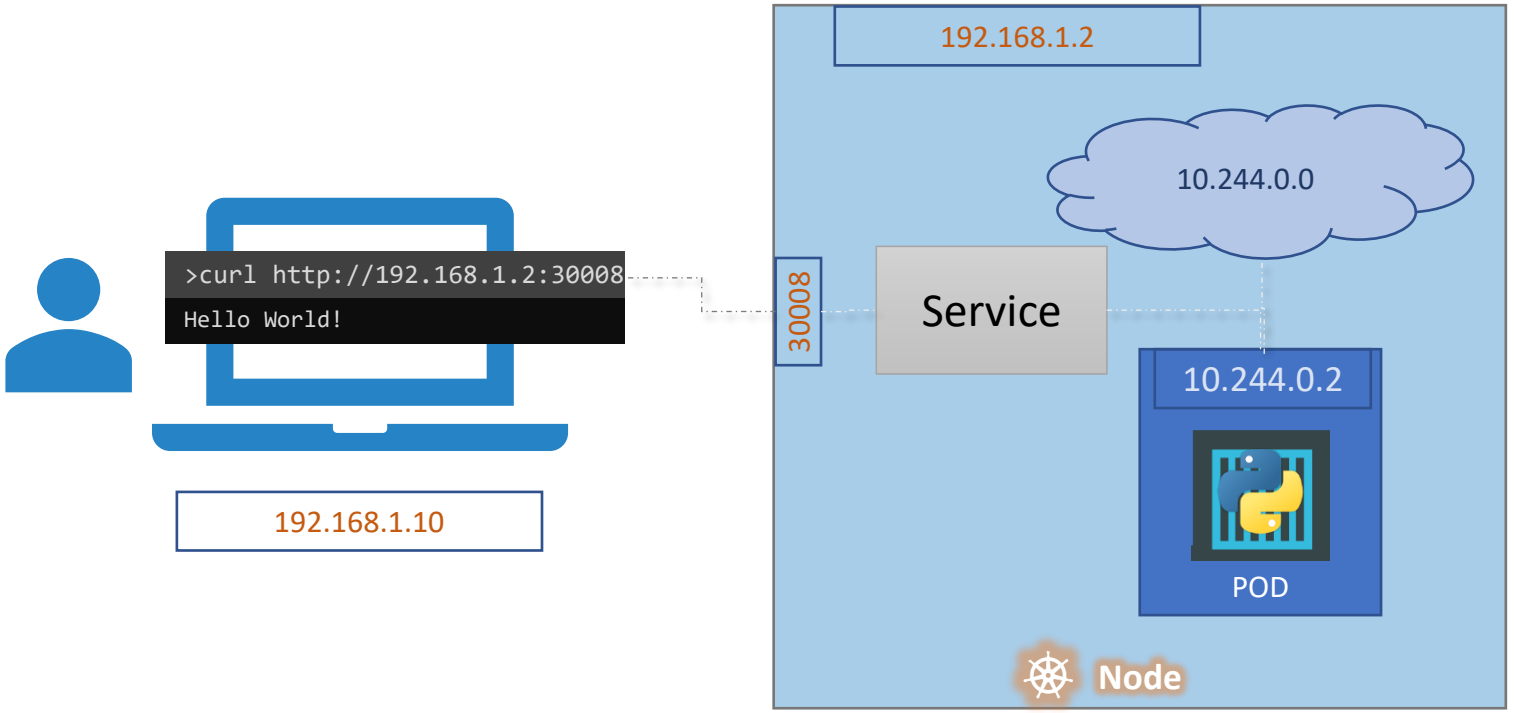


LoadBalancer

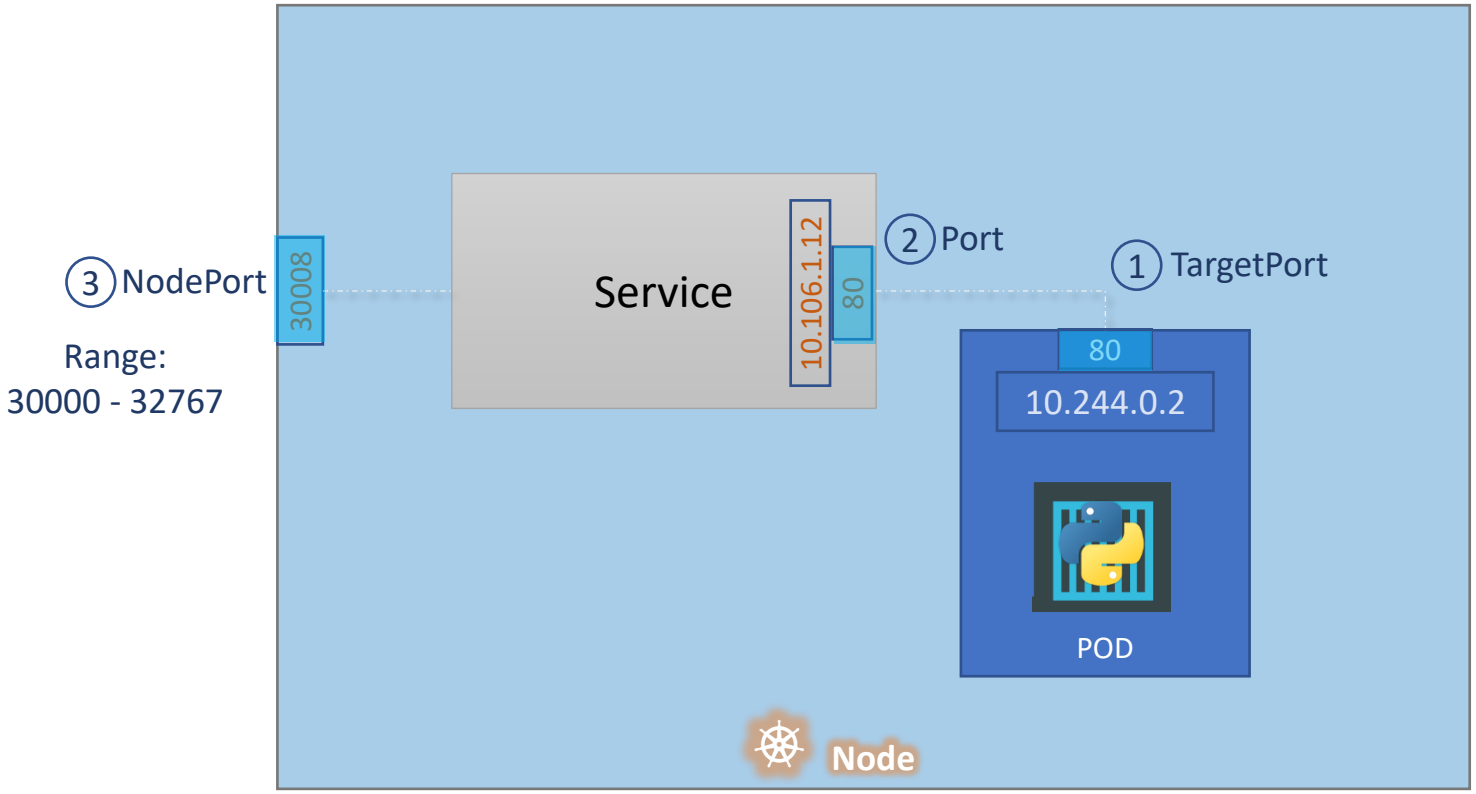
NodePort



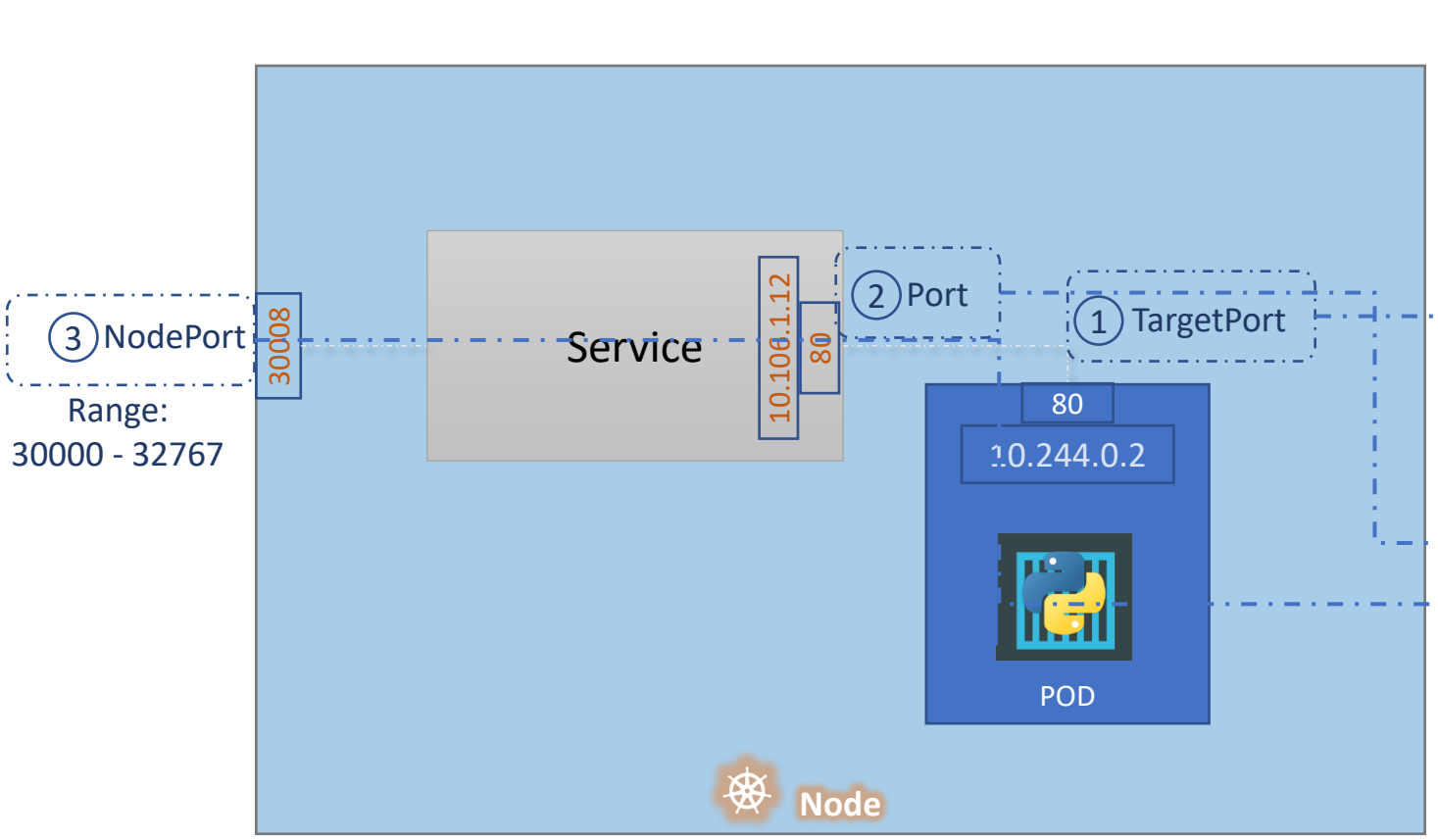
Service - NodePort



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

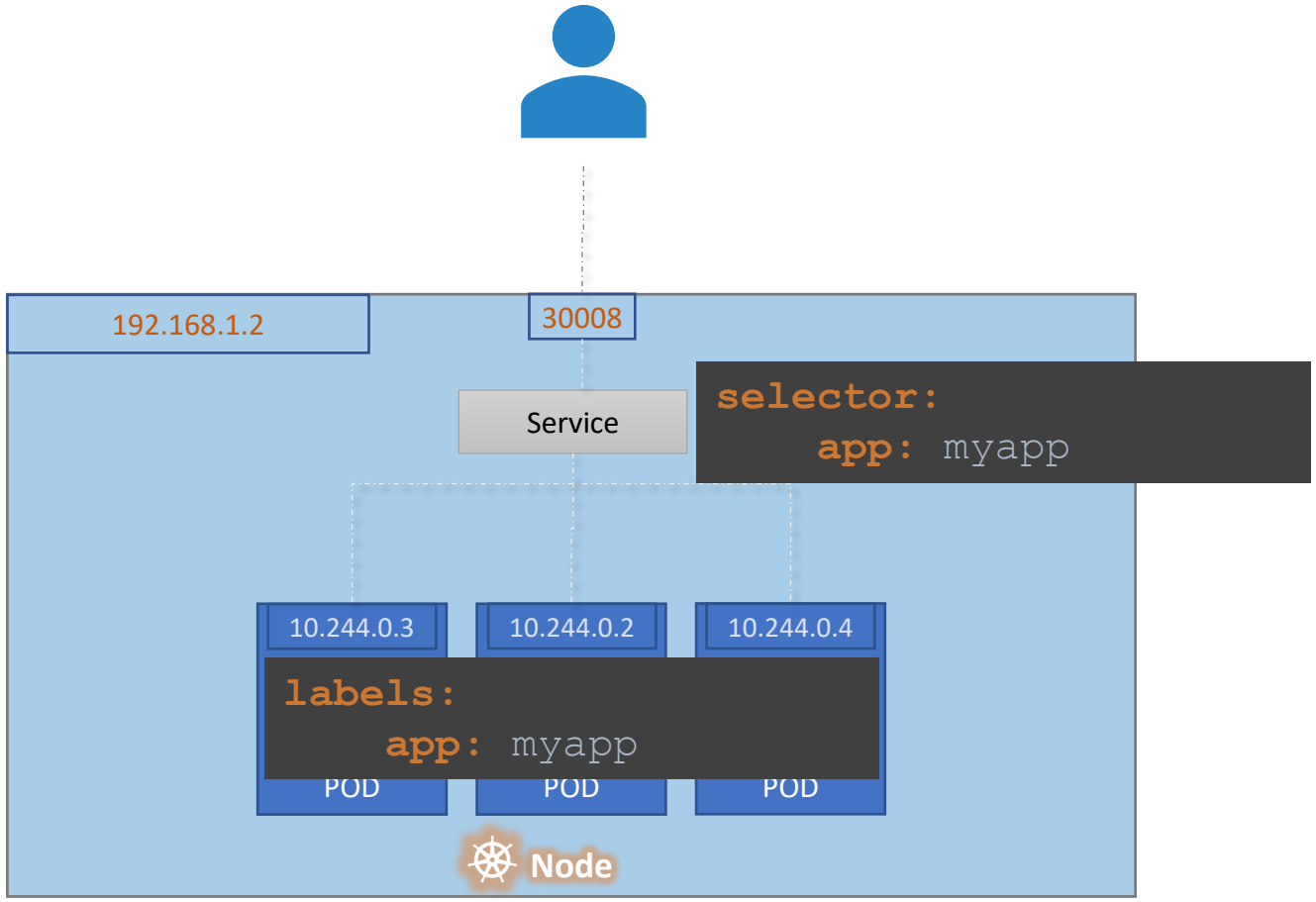
```
> kubectl create -f service-definition.yml
service "myapp-service" created

> 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
```

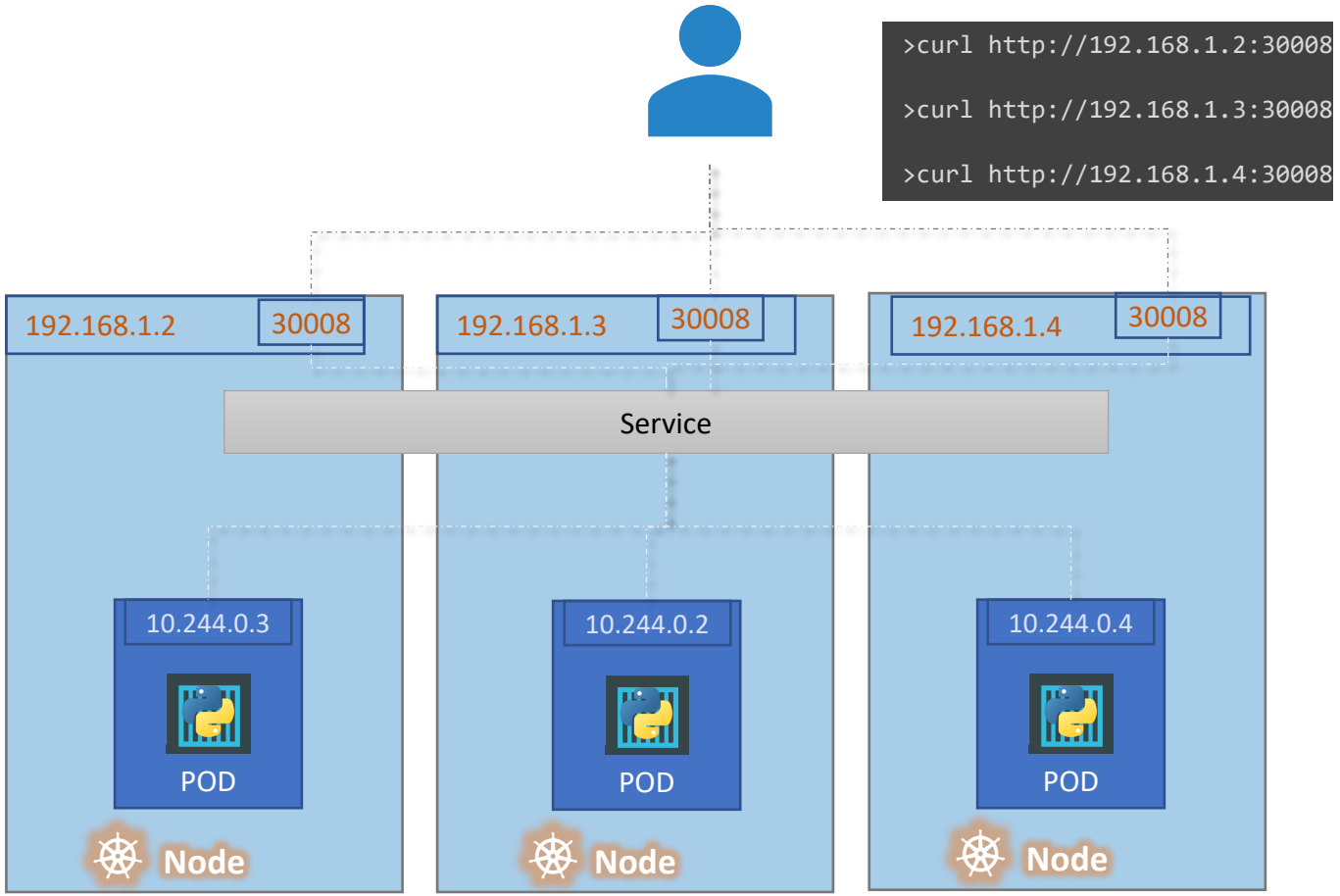
```
> curl http://192.168.1.2:30008
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
```

Service - NodePort

Algorithm: Random
SessionAffinity: Yes



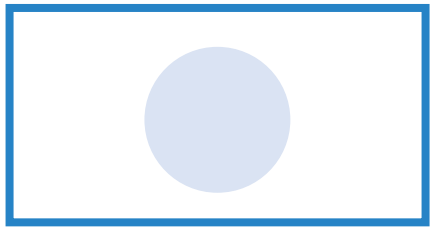
Service - NodePort



Demo

Service - NodePort

ClusterIP



ClusterIP

front-end

back-end

redis