Docker KBTG

- Training Day -

Jumpbox®

Get Slide

SCAN ME

Jumpbox®

Jumpbox®

Tech Passion | Sharing | Society

"เราเชื่อว่าการเรียนรู้ทำให้ชีวิตคุณดีขึ้น"

-Jumpbox Team-

Jumpbox®

JoJo

(Cloud Native Stylist)

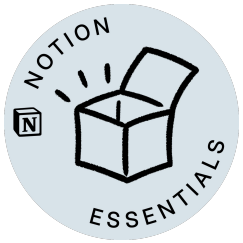
Pae

(Full-stack Developer)

เจ้าของ Page jitrak.dev

THAI PROGRAMMER
สมาคมโปรแกรมเมอร์ไทย

กรรมการ และประธานคณะทำงาน



Jumpbox®

Agenda

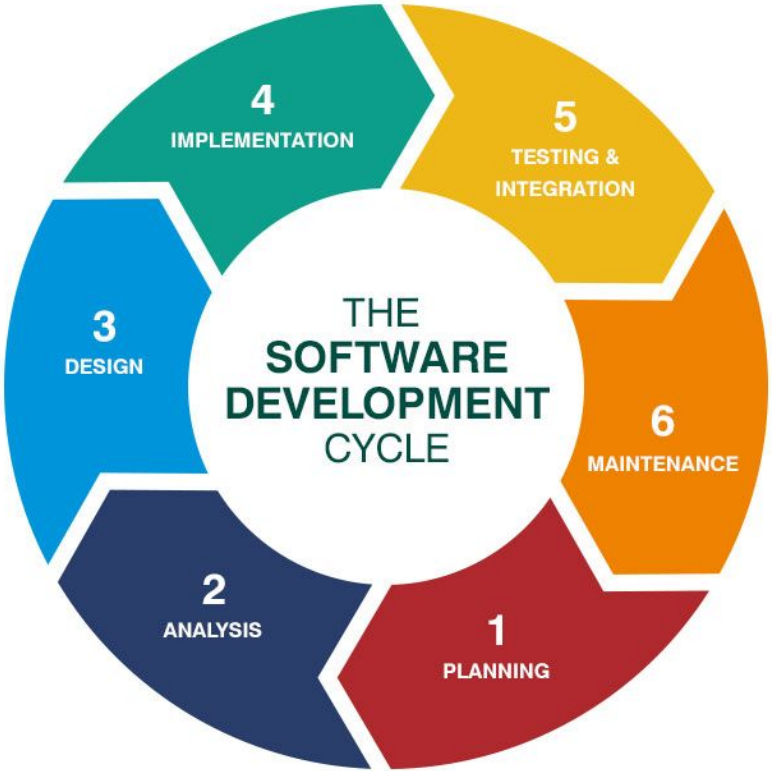
Day 1

- Software Development Life Cycle (SDLC)
- DevOps Evolution
- Get to Know Docker
- Hands-on Workshop: Running Docker Container
- Basic Docker Commands
- Docker Networking and Volumes
- Application Overview & Running Application in Docker

Day 2

- Advanced Docker Usage
- Docker Compose Deep Dive
- Hands-on Workshop: Real-world Application Deployment
- Next Steps & Advanced Tools
- Conclusion & Feedback Session

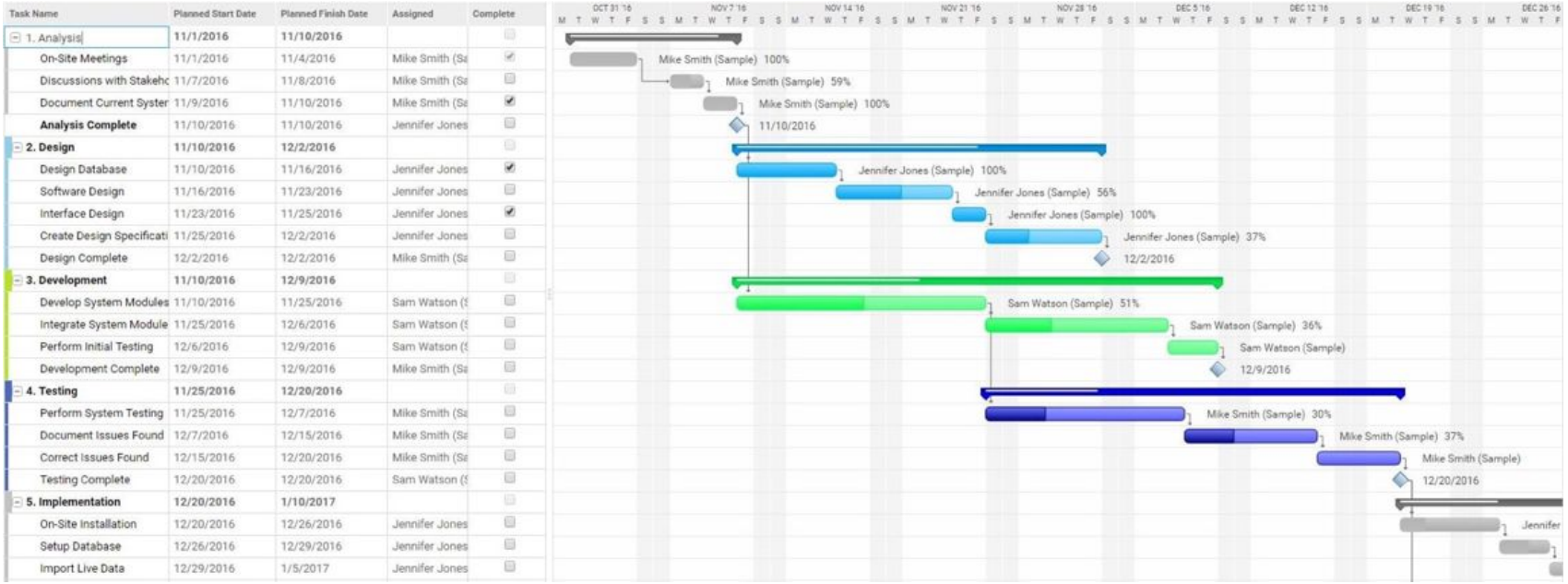
Software Development Life Cycle (SDLC)



Reference Pictures:

- [Software Development Life Cycle (SDLC) - Basics, Stages, Models]

Traditional software project - Gantt chart

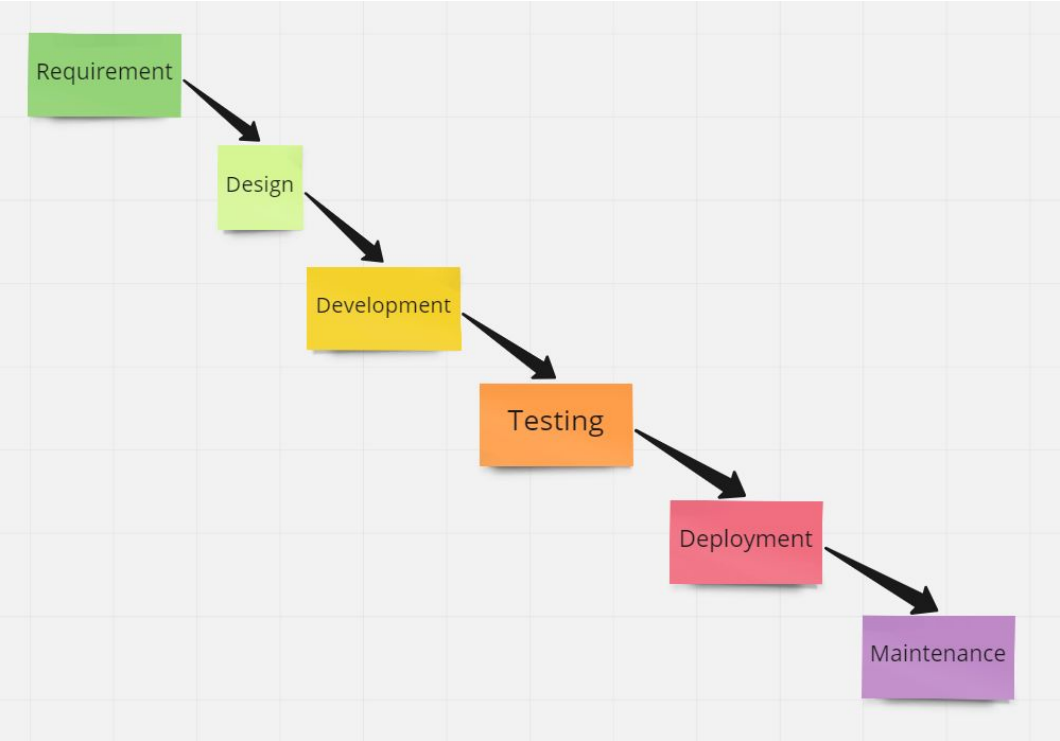


Task Name	Planned Start Date	Planned Finish Date	Assigned	Complete
1. Analysis	11/1/2016	11/10/2016		
On-Site Meetings	11/1/2016	11/4/2016	Mike Smith (Sa	✓
Discussions with Stakehc	11/7/2016	11/8/2016	Mike Smith (Sa	
Document Current Syster	11/9/2016	11/10/2016	Mike Smith (Sa	✓
Analysis Complete	11/10/2016	11/10/2016	Jennifer Jones	
2. Design	**11/10/2016**	**12/2/2016**		
Design Database	11/10/2016	11/16/2016	Jennifer Jones	✓
Software Design	11/16/2016	11/23/2016	Jennifer Jones	
Interface Design	11/23/2016	11/25/2016	Jennifer Jones	✓
Create Design Specificati	11/25/2016	12/2/2016	Jennifer Jones	
Design Complete	12/2/2016	12/2/2016	Mike Smith (Sa	
3. Development	**11/10/2016**	**12/9/2016**		
Develop System Modules	11/10/2016	11/25/2016	Sam Watson (S	
Integrate System Module	11/25/2016	12/6/2016	Sam Watson (S	
Perform Initial Testing	12/6/2016	12/9/2016	Sam Watson (S	
Development Complete	12/9/2016	12/9/2016	Mike Smith (Sa	
4. Testing	**11/25/2016**	**12/20/2016**		
Perform System Testing	11/25/2016	12/7/2016	Mike Smith (Sa	
Document Issues Found	12/7/2016	12/15/2016	Mike Smith (Sa	
Correct Issues Found	12/15/2016	12/20/2016	Mike Smith (Sa	
Testing Complete	12/20/2016	12/20/2016	Sam Watson (S	
5. Implementation	**12/20/2016**	**1/10/2017**		
On-Site Installation	12/20/2016	12/26/2016	Jennifer Jones	
Setup Database	12/26/2016	12/29/2016	Jennifer Jones	
Import Live Data	12/29/2016	1/5/2017	Jennifer Jones	

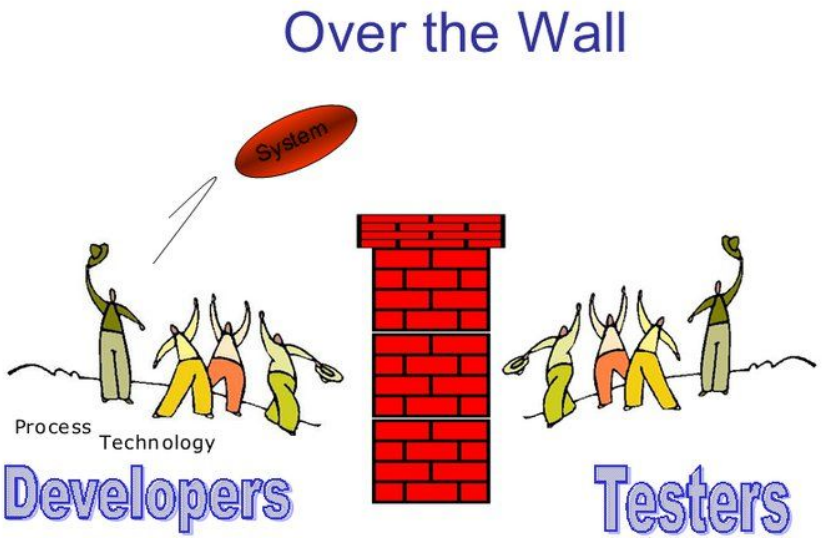
Reference Pictures:

- [Software Development Life Cycle (SDLC) - Basics, Stages, Models]

Waterfall model



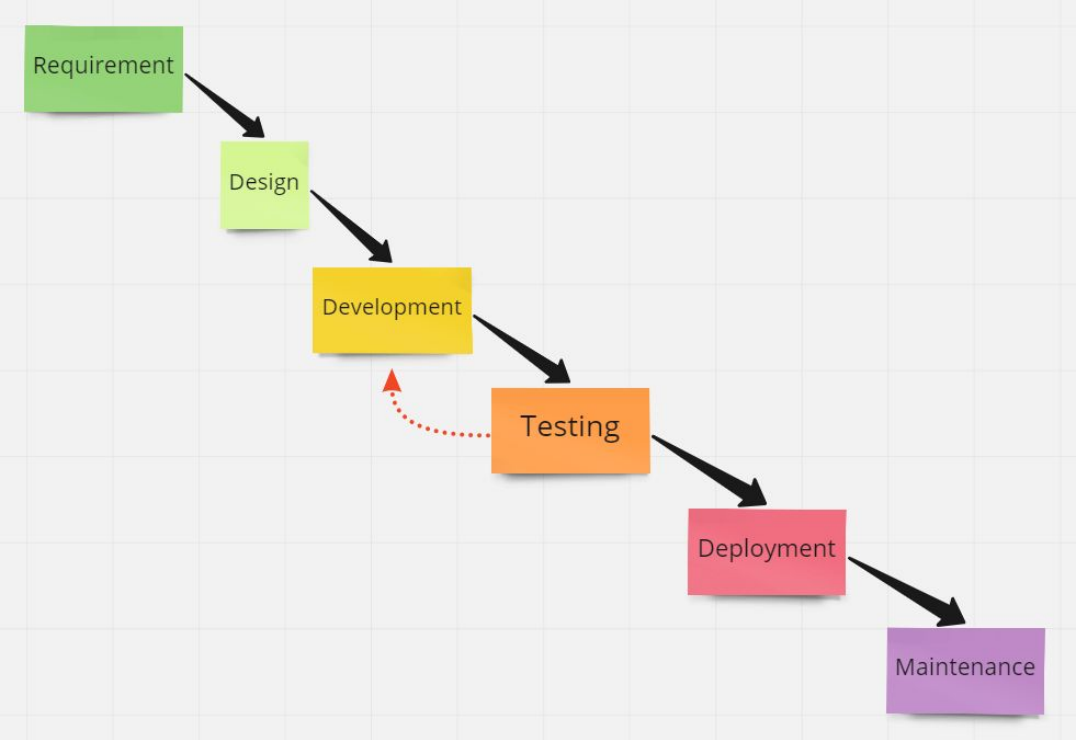
Hand Over



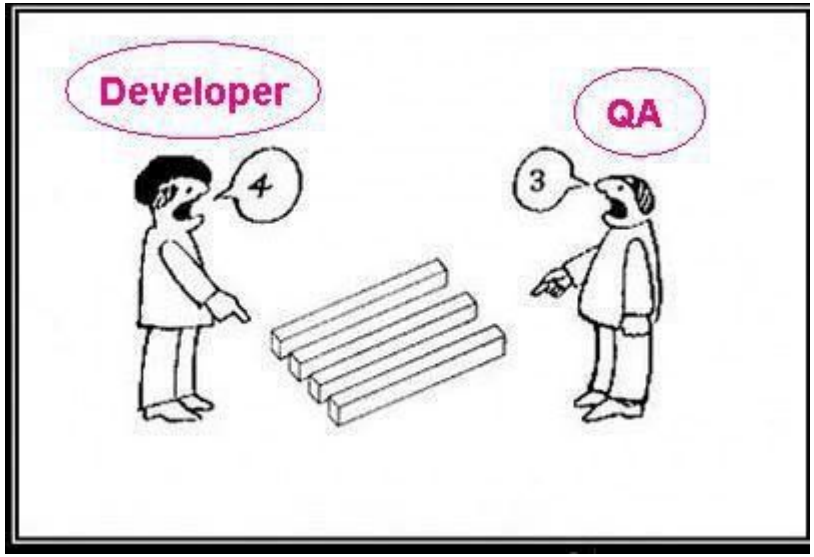
Reference Pictures:

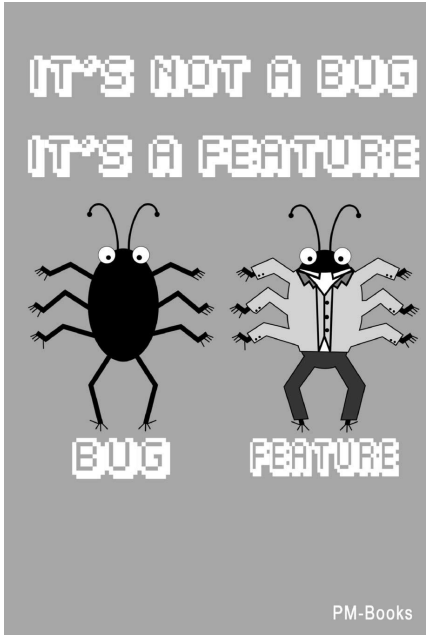
- Software Testing Or Software Development: Which Career Path To Follow?

Waterfall model - Feedback



Developer vs Tester



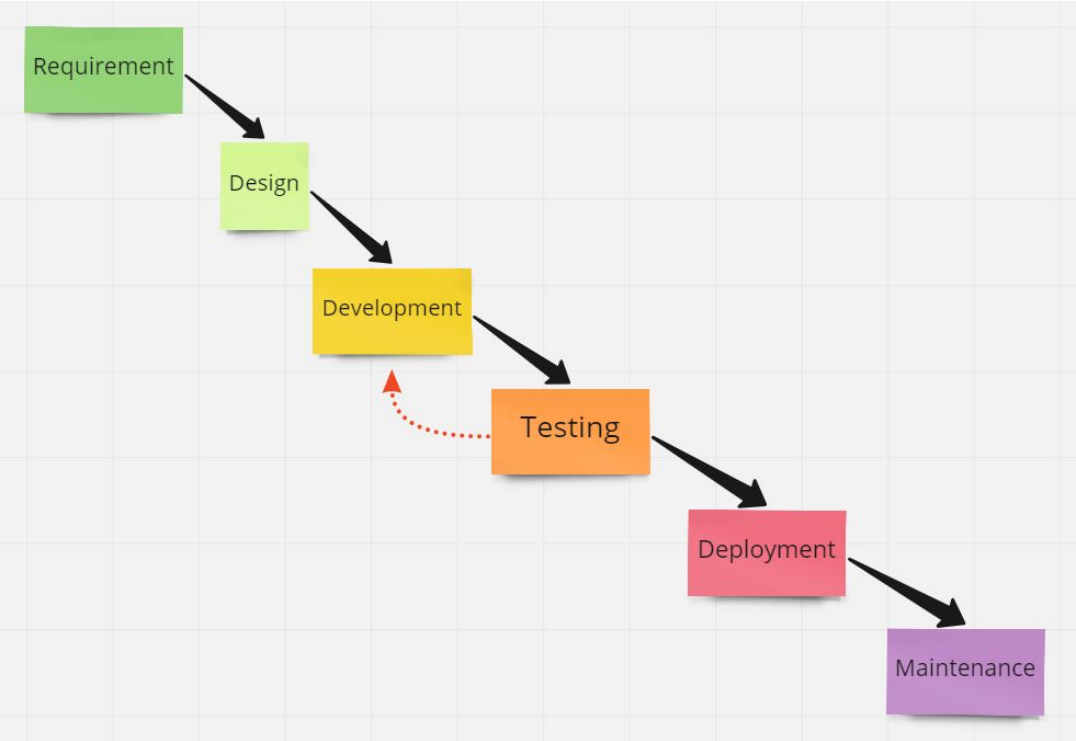


Reference Pictures:

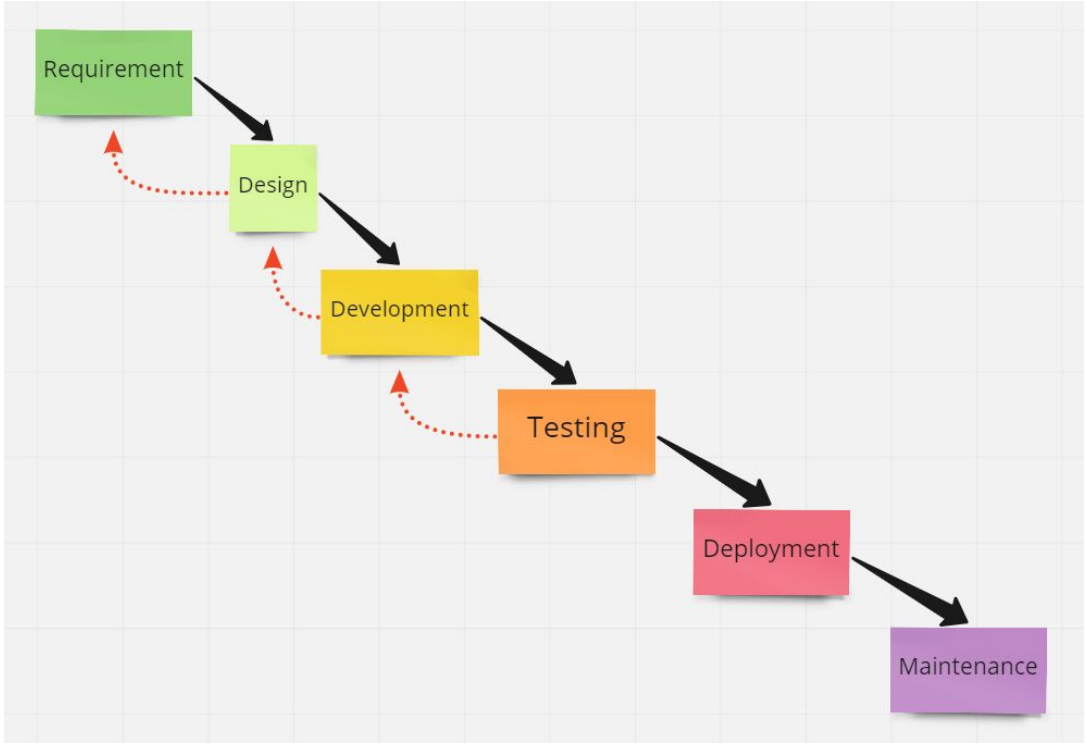
- Tester vs. Developer - part 2
- It´s not a bug it´s a feature: Lined notebook or paperback for pupils, students and adults Paperback

Jumpbox®

Waterfall model - Feedback (Cont.)



Waterfall model - Feedback (Cont.)

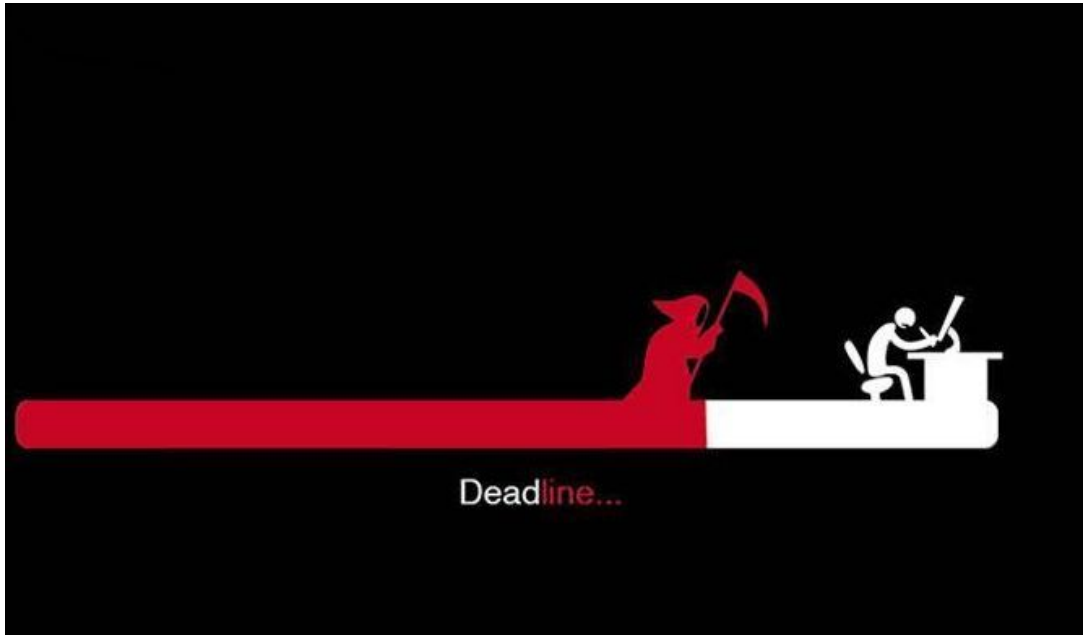


Legacy silos in the team

Reference Pictures:

- Tradition Silos

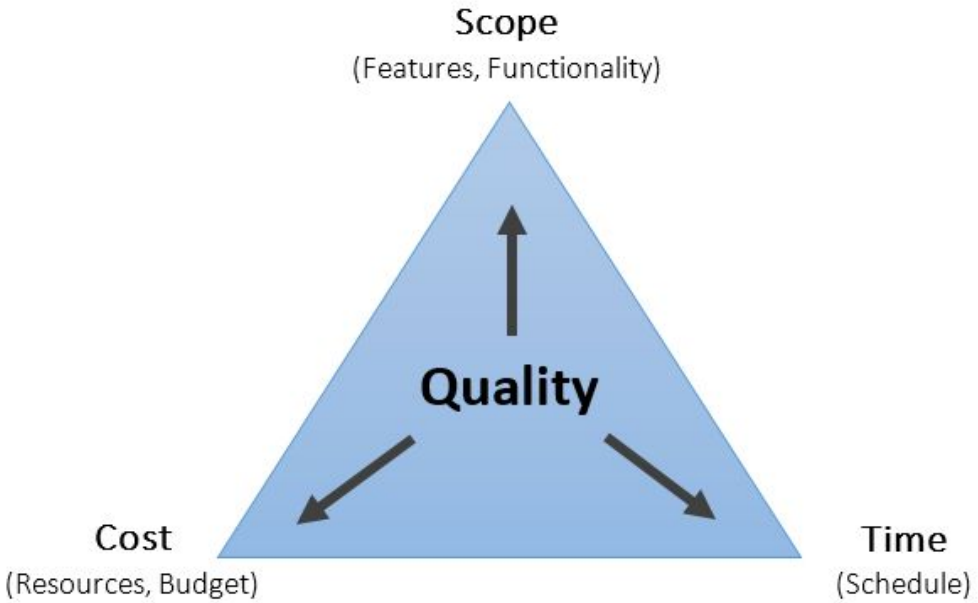
Deadline



Reference Pictures: [Funny Loading Bar]

Jumpbox®

Iron Triangle



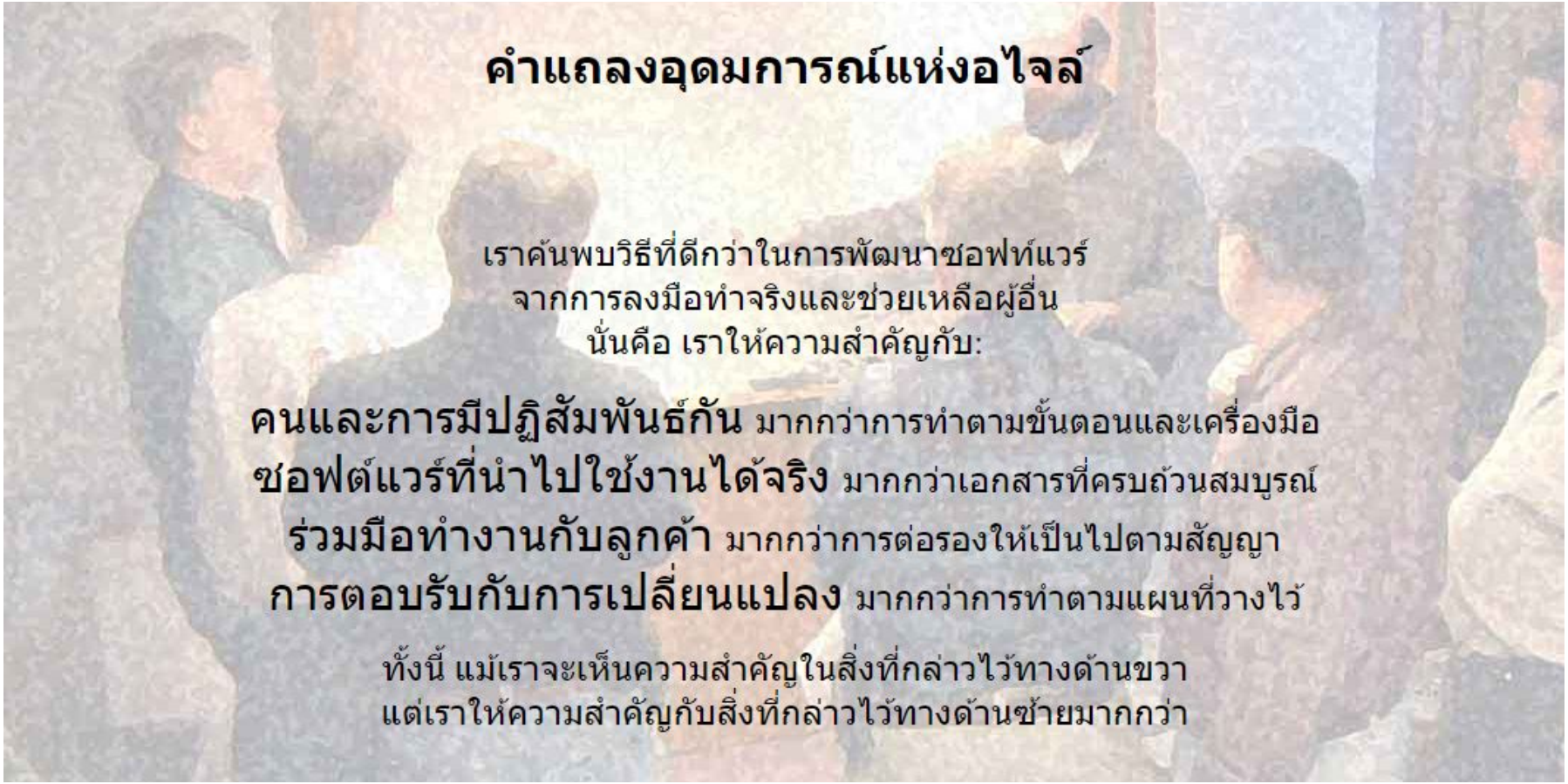
Reference Pictures:

- [Iron Triangle — Triple Constraints of Project Management]

Agile

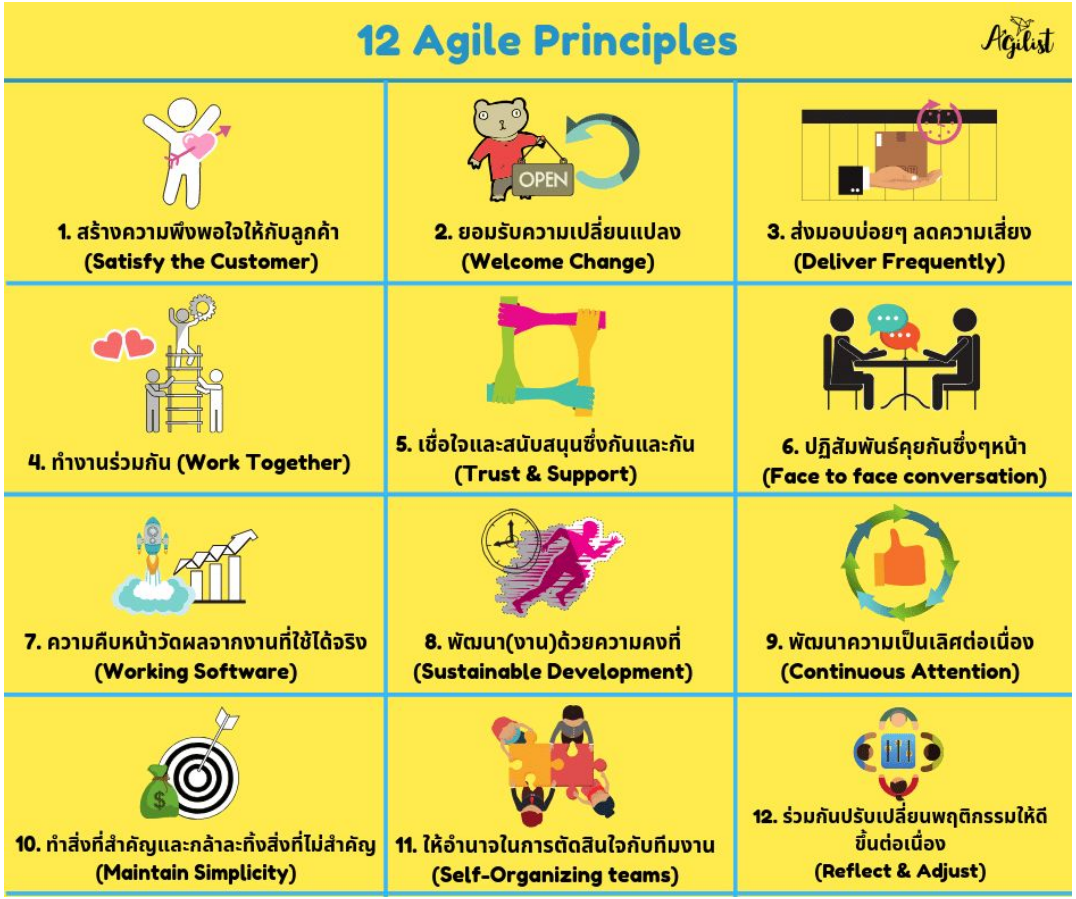
Reference Pictures:

- Manifesto for Agile Software Development



Reference Pictures:

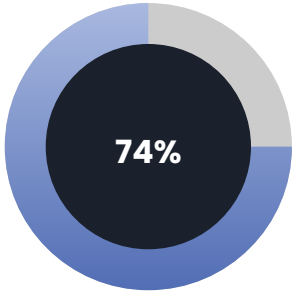
- [Manifesto for Agile Software Development]



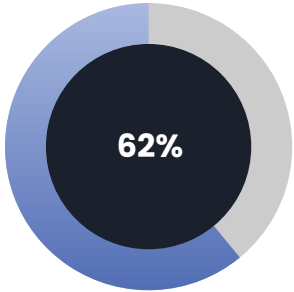
Reference Pictures:

- [หลักการอไจล์ 12 ข้อ (เวอร์ชั่น 1 หน้า) 12 Agile Principles]

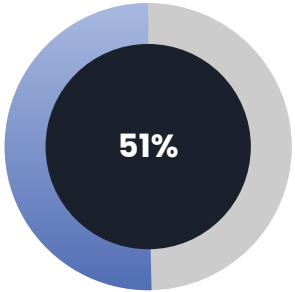
Top 3 reasons for adopting Agile



Accelerate software delivery



Enhance ability to manage changing priorities

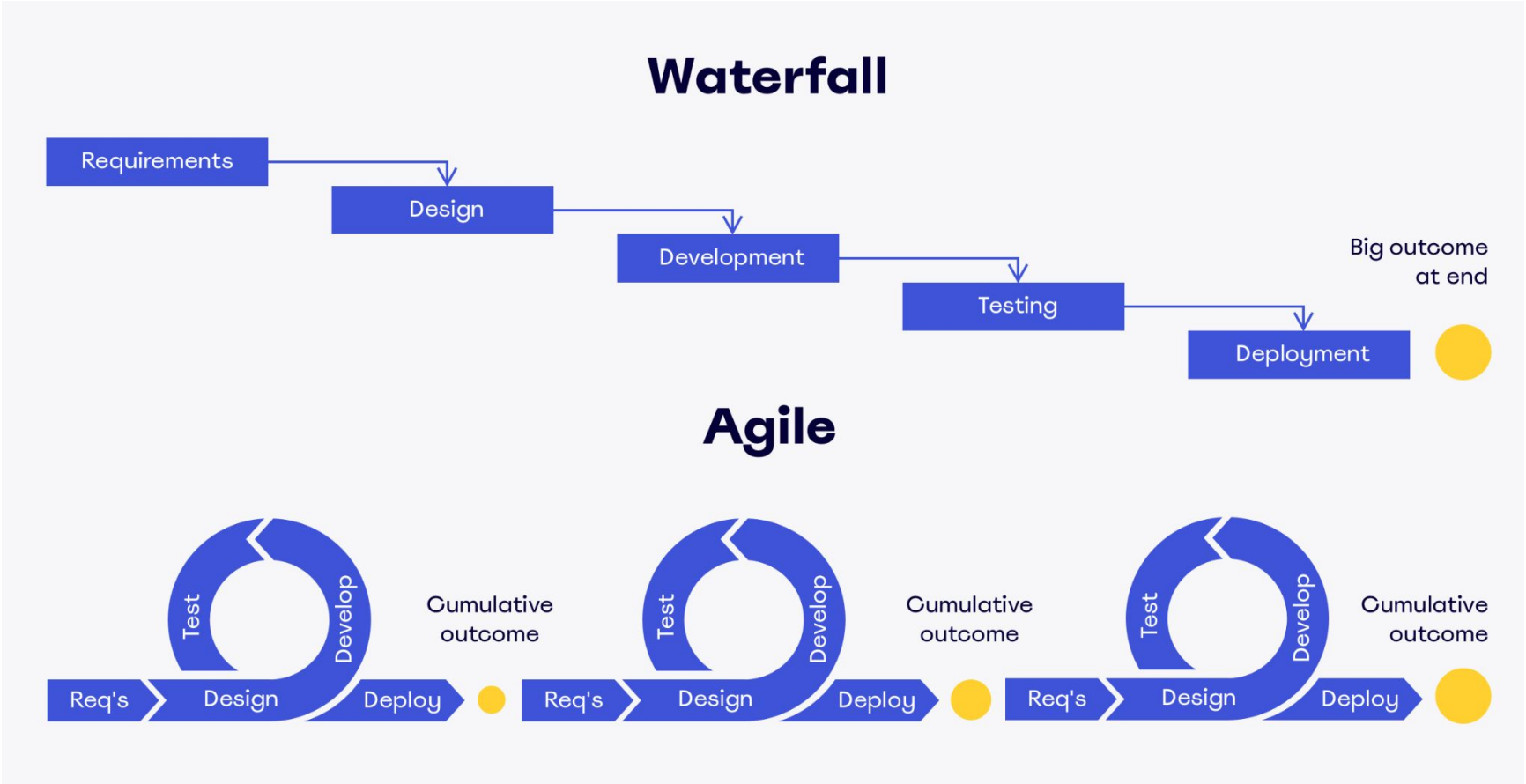


Increase productivity

Reference Pictures:

- Visual guide to Agile methodologies for modern product management

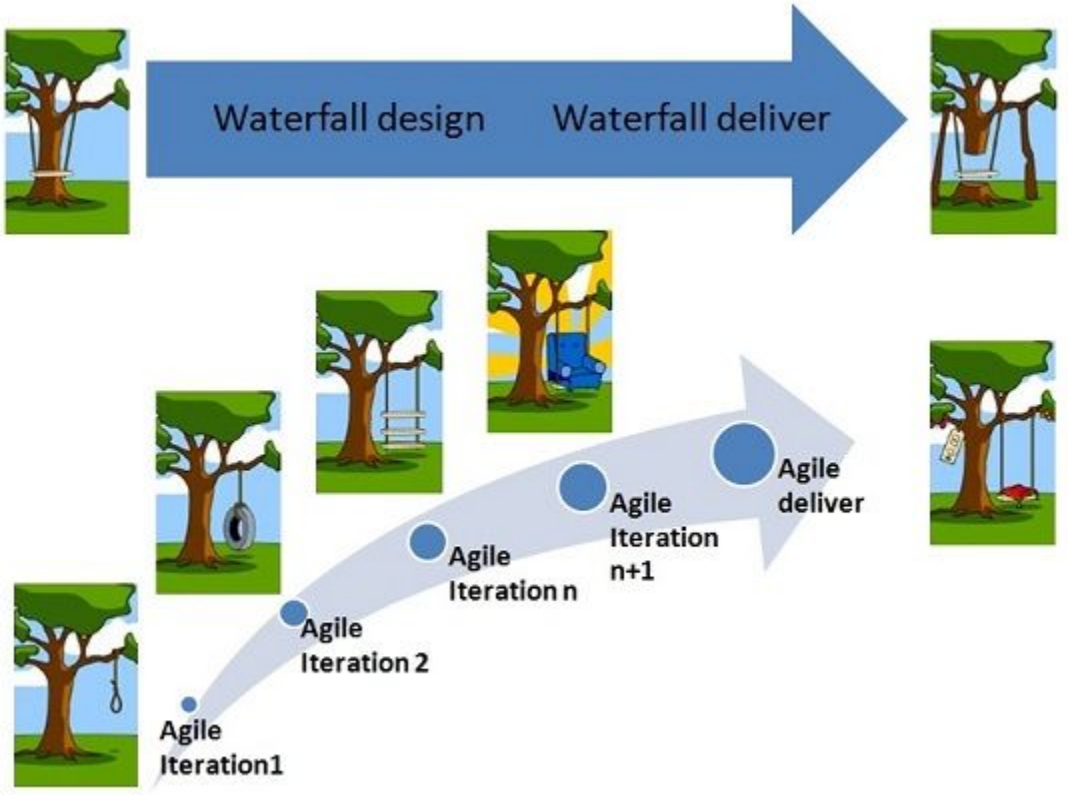
Jumpbox®



Reference Pictures:

- Visual guide to Agile methodologies for modern product management

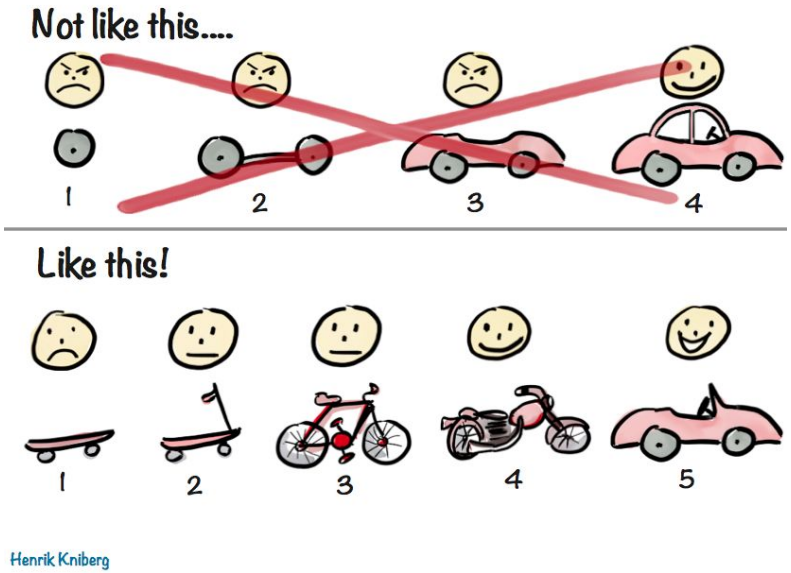
Jumpbox®



Reference Pictures:

- [Agile Software Development Basics and fundamentals]

Minimum Viable Product (MVP)



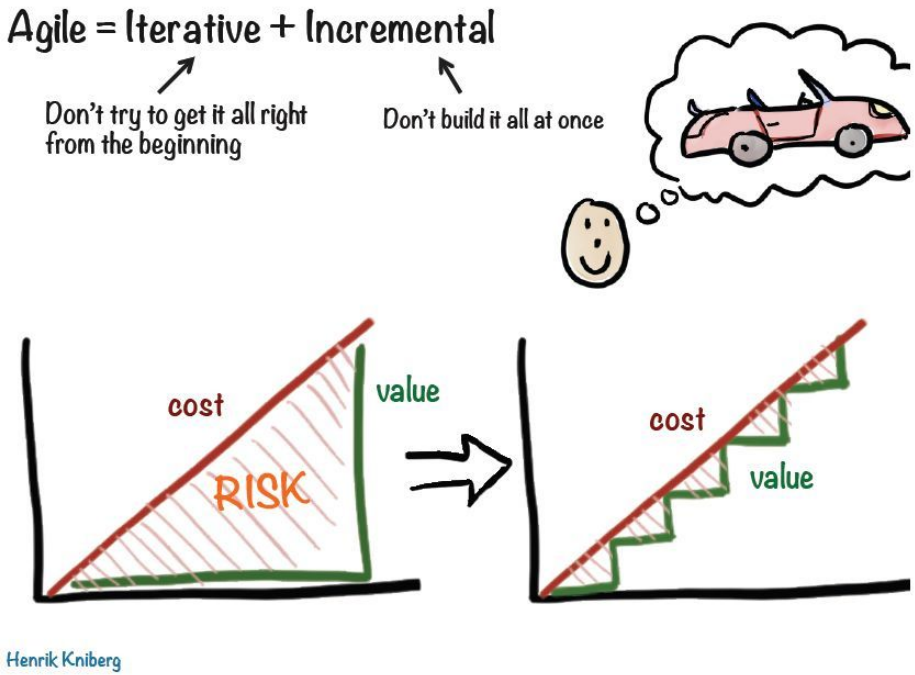
- Value is delivered at each stage of development.
- The solution is delivered and improved upon in an iterative model.
- At each stage, the solution can be evaluated and adjusted if needed.

Reference Pictures:

- The Agile Bicycle: A Better Analogy for Software Development

Jumpbox®

Risk



Reference Pictures:

- [Using Agile practices to manage project risk]

Waterfall

- Customers exactly know what they want.
- Developers exactly know what to do.
- Factors less change during the development.

Agile

- Customers gradually discover what they want.
- Developers gradually know what to do.
- Factors can change during development.

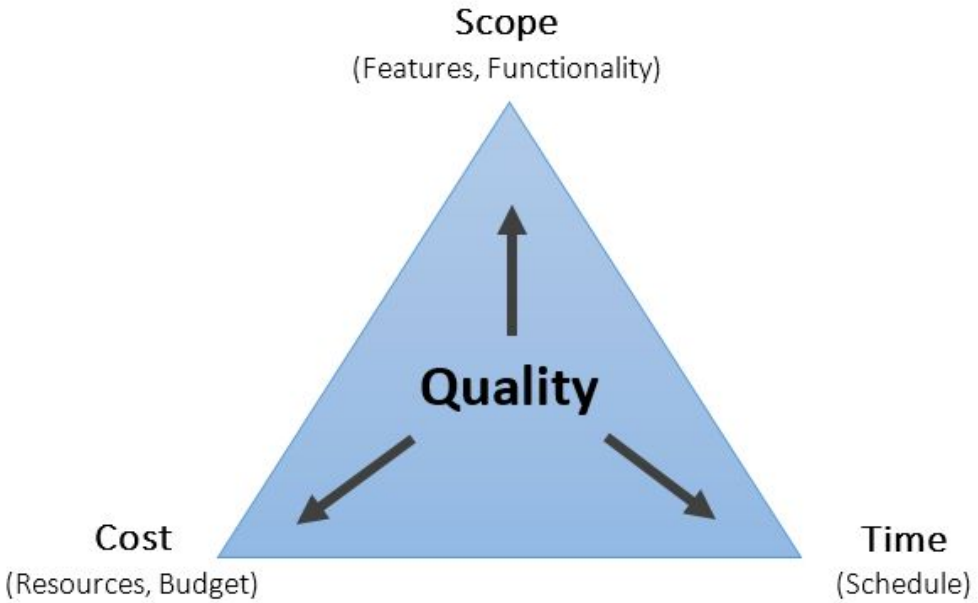
Reference Pictures:

- Lean Startup กับ Agile เหมือนหรือต่างกันอย่างไร

Embraces change

- Requirement change
- Goal change
- Situation change

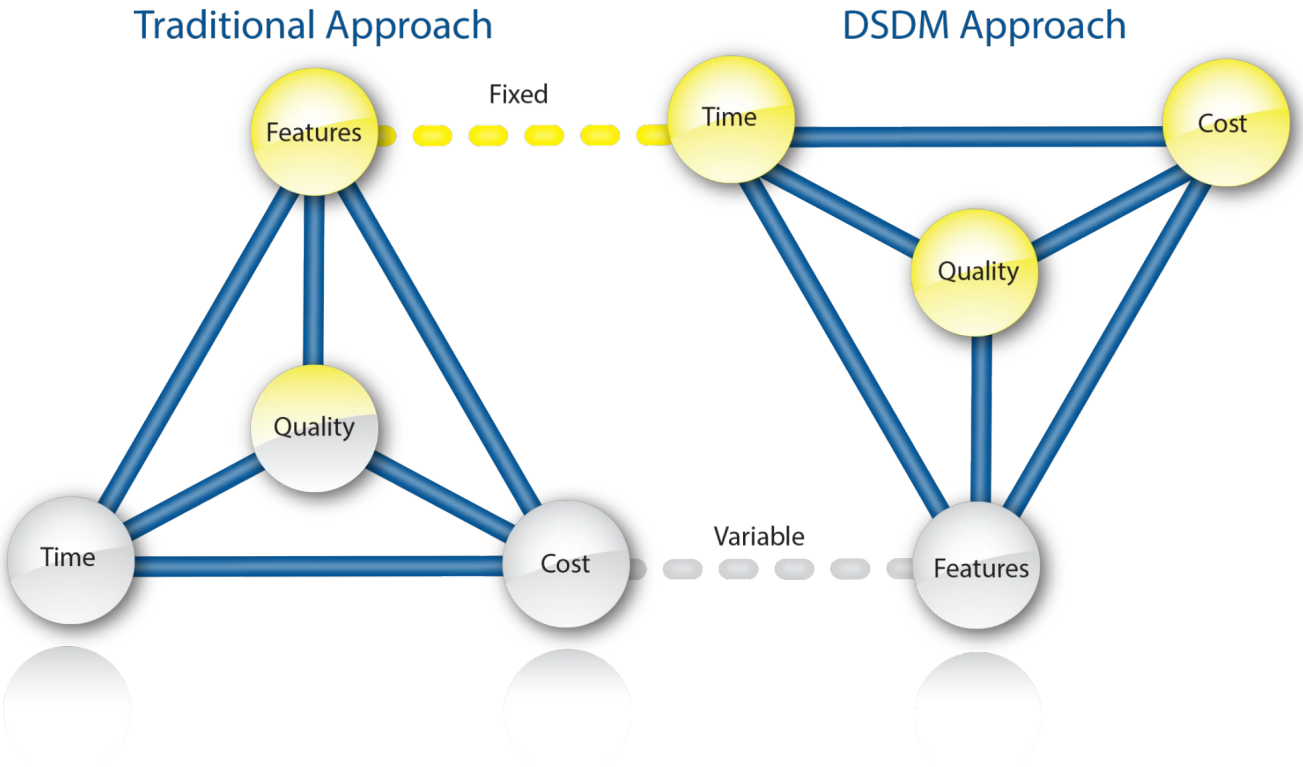
Iron Triangle



Reference Pictures:

- [Iron Triangle — Triple Constraints of Project Management]

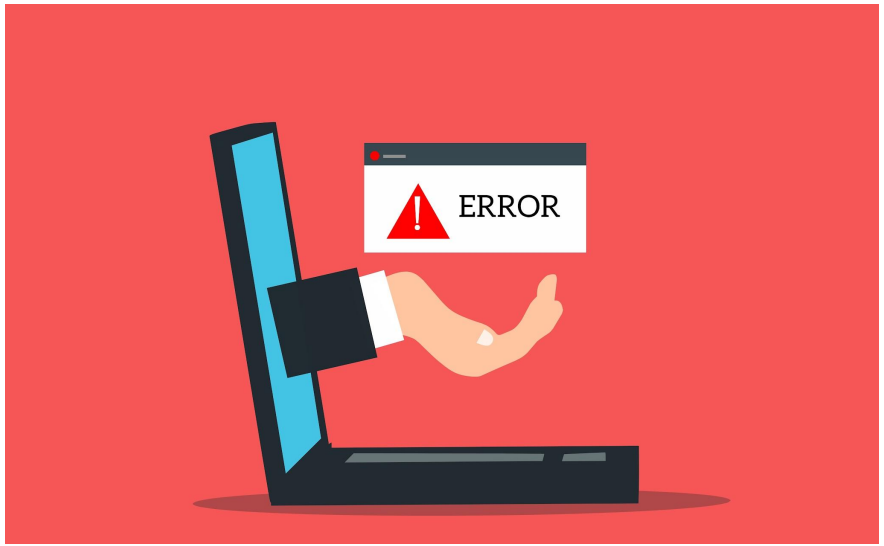
Traditional Approach and DSDM Approach



Reference Pictures:

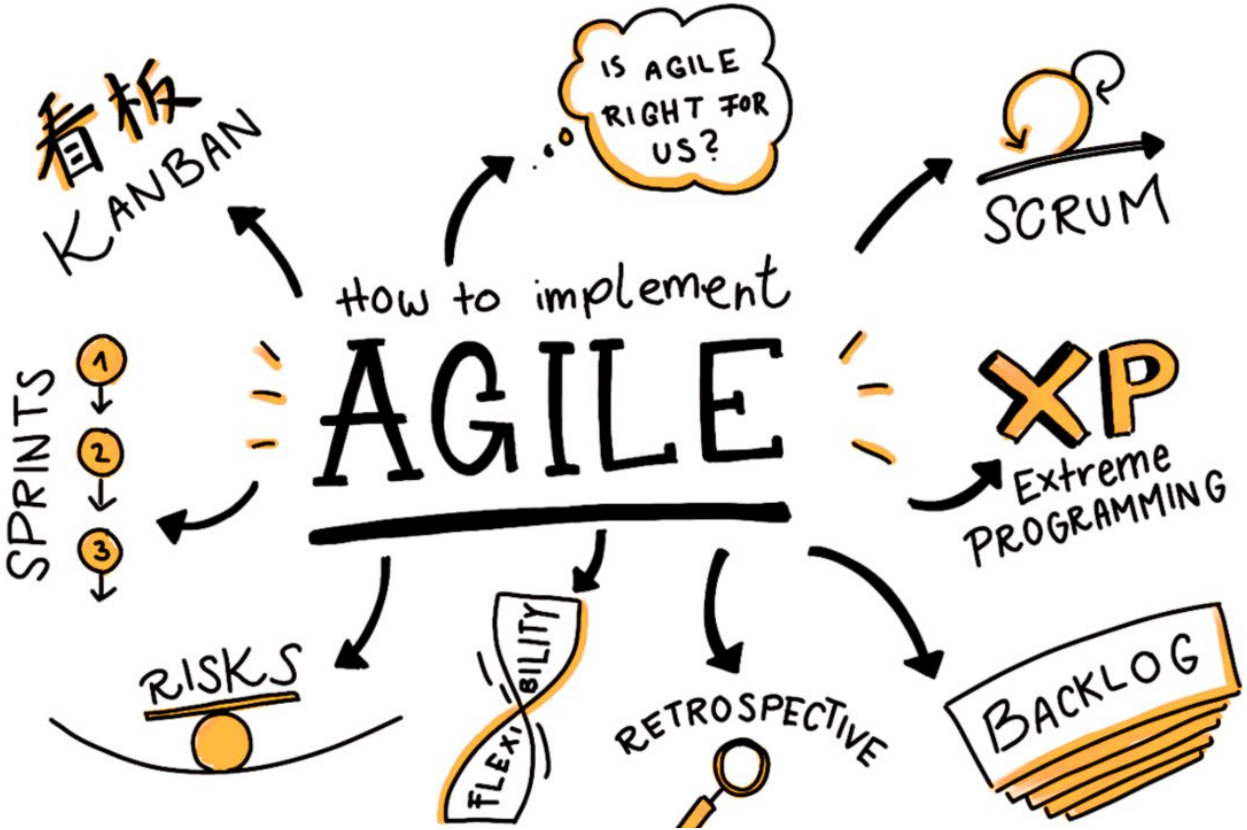
- Dynamic Systems Development Method (DSDM): Part 2 - Philosophy & Fundamentals

Broken software



Reference Pictures:

- https://pxhere.com/en/photo/1636749



Reference Pictures:

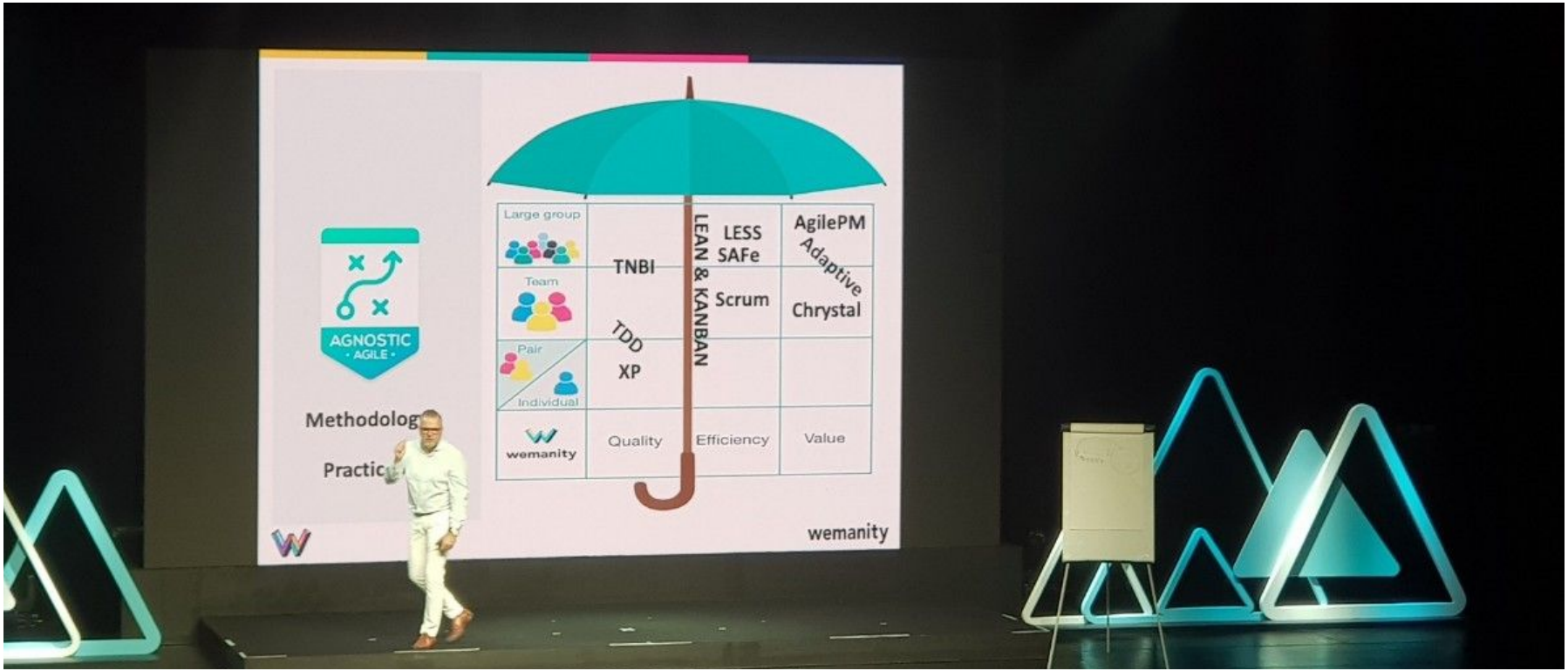
- [Utilizing Agile Methodologies in B2B]

Agile methodologies

Reference Pictures:

- Visual guide to Agile methodologies for modern product management

Agile Umbrella



Reference Pictures:

- Agile and the Future @ Beyond Agile by KBTG by Arie van Bennekum

Scrum

Scrum roles

- Product Owner (PO)
- Scrum Master
- Development Team

Product manager

- Owns the product road map
- Advocates for the product internally
- Represents the customer in meetings with development
- Is a job title

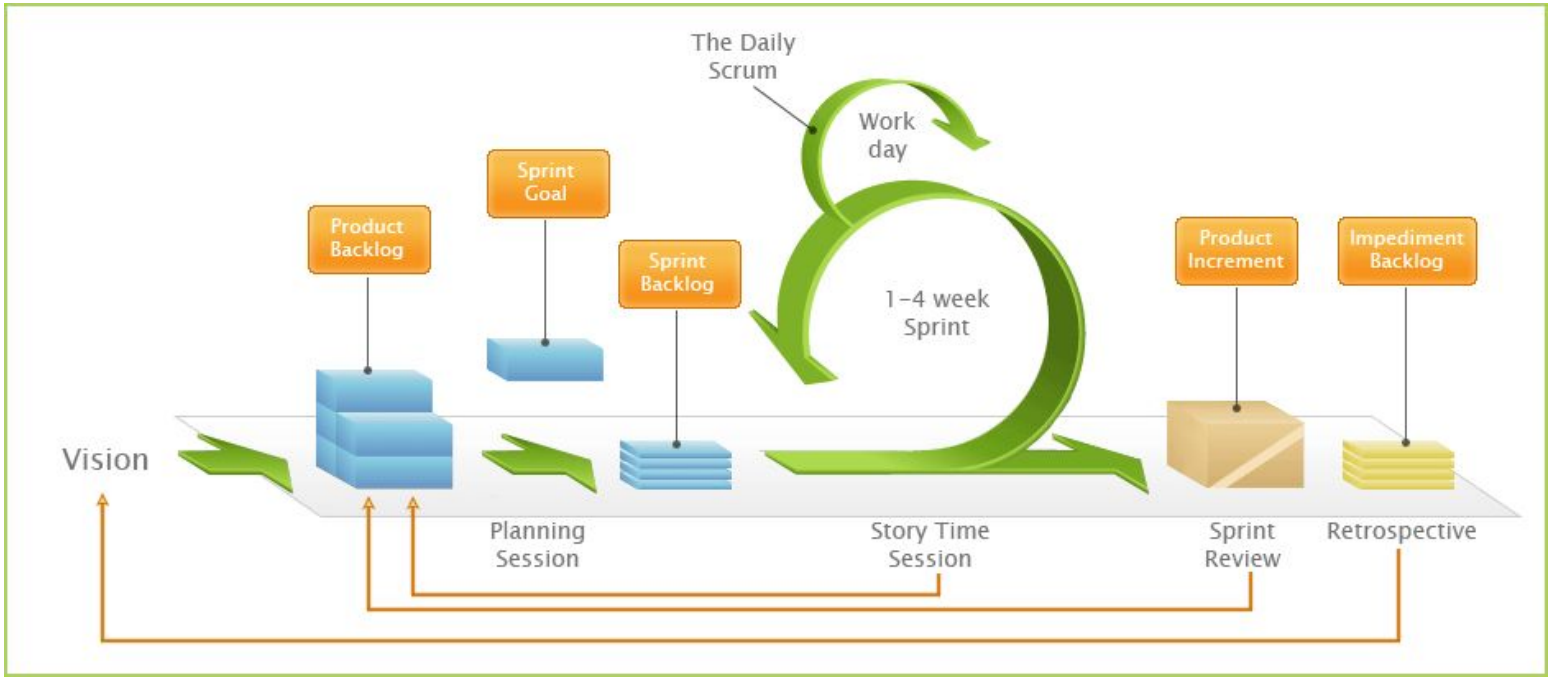
Product owner

- Grooms the team's backlog
- Answers the team's questions about requirements
- Has discussions with stakeholders
- Accepts content from the team
- Is the title of role, which could be performed by someone with any title

Reference Pictures:

- Lean Startup กับ Agile เหมือนหรือต่างกันอย่างไร

Scrum Life Cycle



Reference Pictures:

- Scrum ฉบับย่อ

Jumpbox®

Daily scrum

- What did I work on yesterday?
- What am I working on today?
- What issues are blocking me?

Kanban

Traffic Jam

Reference Pictures:

- Blockers and Traffic Jams

WIP (Work In Progress)

Reference Pictures:

- Kanban WIP limits - Agile Coach (2019)

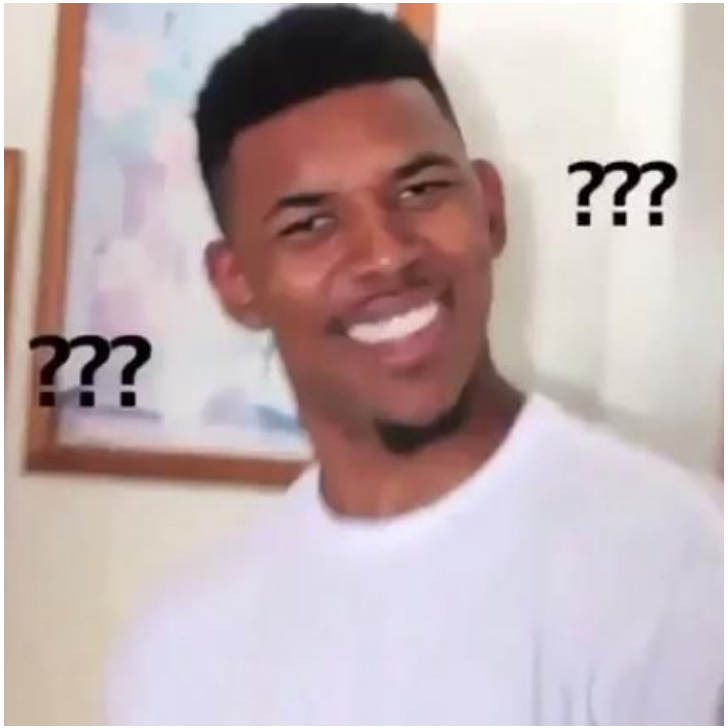
Jumpbox®

	Scrum	Kanban
Cadence	Regular fixed length sprints (ie, 2 weeks)	Continuous flow
Release methodology	At the end of each sprint	Continuous delivery
Roles	Product owner, scrum master, development team	No required roles
Key metrics	Velocity	Lead time, cycle time, WIP
Change philosophy	Teams should not make changes during the sprint.	Change can happen at any time

Reference Pictures:

- [Kanban vs. scrum: which agile are you?]

Scrumban



Scrum vs Kanban vs Scrumban

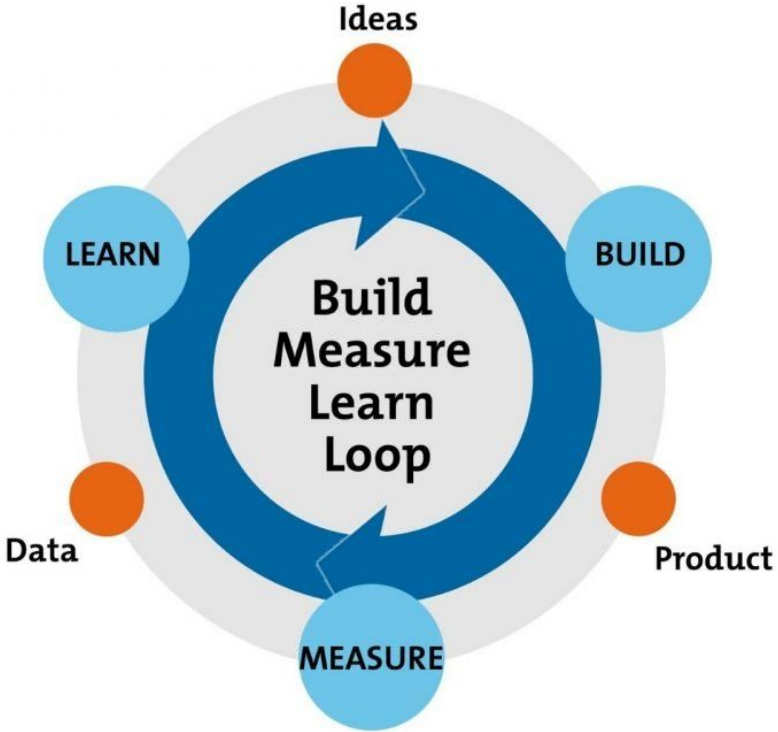
	Scrum	Kanban	Scrumban
Time base	1-4 weeks sprints	No time base - Kanban is event-driven	1-year, 6-months and 3-months buckets
Rules	Complete constrained process	few constraints mostly flexible process	Slightly restricted process
Roles	Product owner, Scrum master, scrum team and stakeholders	No specific roles required	No specific roles required
Event-Based	No - Once started sprints cannot be modified	Yes - On going work can react to the workflow	Yes - On going work can react to the workflow and cause On-Demand Planning
Board	Defined/resets each sprint	Persistent - the Kanban board	Persistent - the Scrumban board
Prioritization	Through backlog	Optional	Recommended on each planning
Work routines	The product owner manages tasks and assigns them to team members	Team members choose and pull tasks	The project manager push tasks in the To-Do column and team members choose and pull from there
Scope limits	Sprint limits the work amount	Work in progress limits current on going work amount	Work in progress limits and optional To-Do limit
Task size	What can be delivered in a single sprint	Any size	Any size
New items in an iteration	Not allowed	Allowed whenever the queue allows it (WIP limits)	Allowed whenever the queue allows it (To-Do & WIP limits)
Meetings	Sprint planning, daily stand-ups, sprint reviews and retrospectives	Avoidable	On-Demand Planning
Estimation	Has to be done before sprint has started	Optional	Optional
Planning routines	Sprint planning	Release/iteration planning, demand planning	Planning on demand for new tasks
Performance metrics	Burndown	Cumulative flow diagram, lead time cycle time	Average cycle time
Performance feedback	Sprint retrospective	Optional	Improvement events are an option

Reference Pictures:

- Cheatsheet Differences Scrum vs Scrumban vs Kanban

Lean Startup

Build-Measure-Learn - Feedback Loop



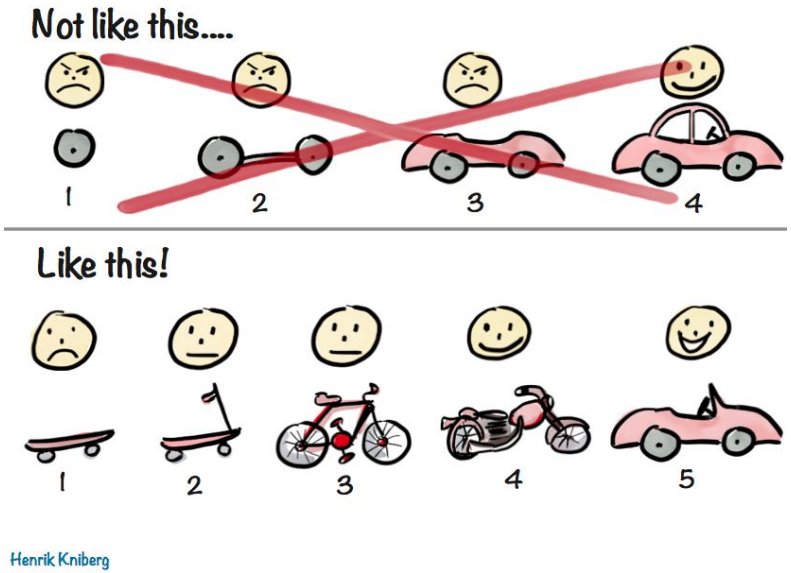
VISUALISE
LETS INNOVATE

Reference Pictures:

- Why the B-M-L Loop of Lean Startup is Misconstrued

Jumpbox®

Minimum Viable Product (MVP)



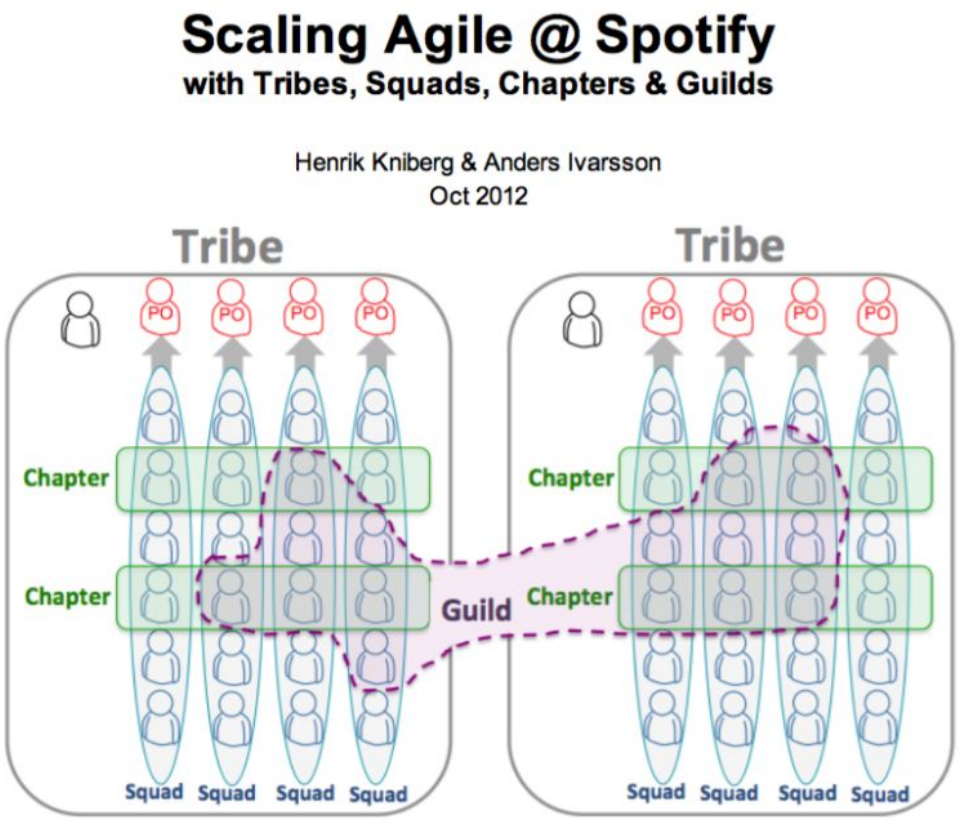
- Value is delivered at each stage of development.
- The solution is delivered and improved upon in an iterative model.
- At each stage, the solution can be evaluated and adjusted if needed.

Reference Pictures:

- The Agile Bicycle: A Better Analogy for Software Development

Jumpbox®

Examples of lean startup



Reference Pictures:

- [Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds]

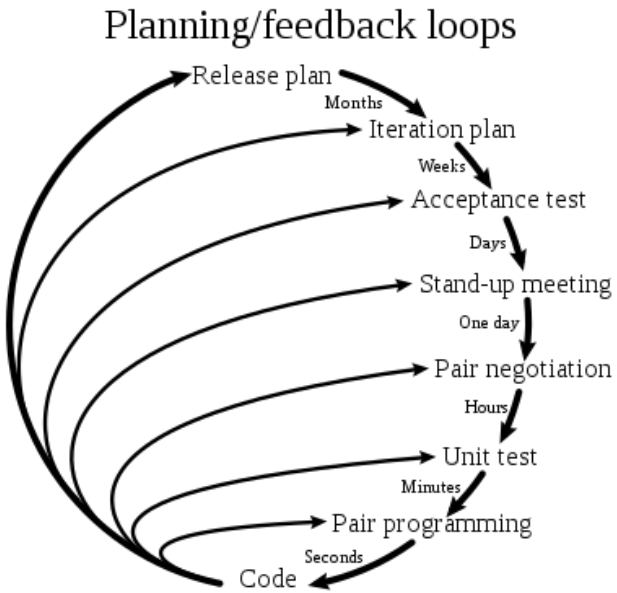
Extreme Programming (XP)

Tech Debt

Reference Pictures:

- Vincentdnl Drawings

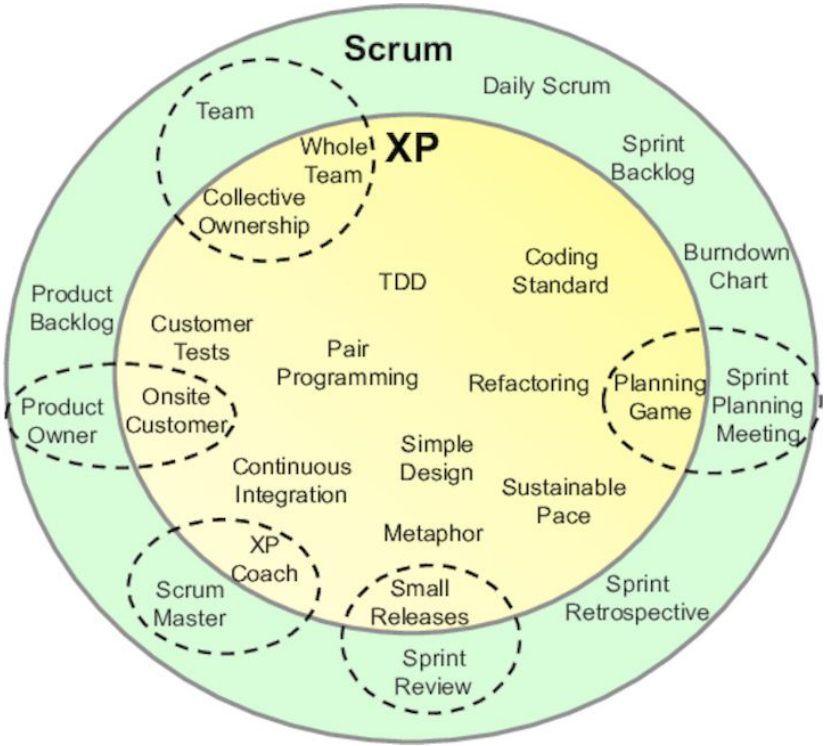
XP feedback loops



Reference Pictures:

- Extreme programming

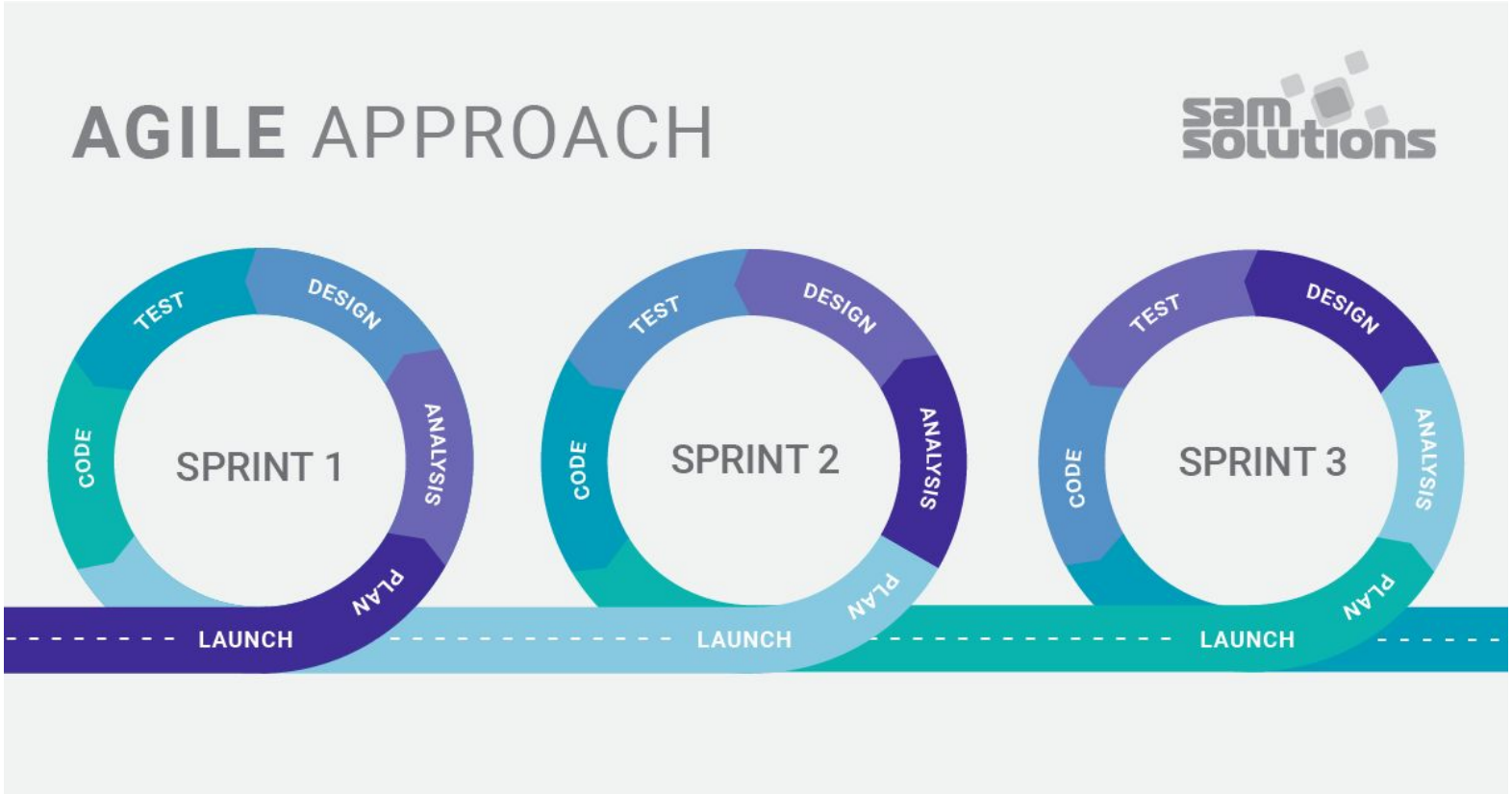
XP with Scrum



Reference Pictures:

- [XP in a Nutshell]

TDD

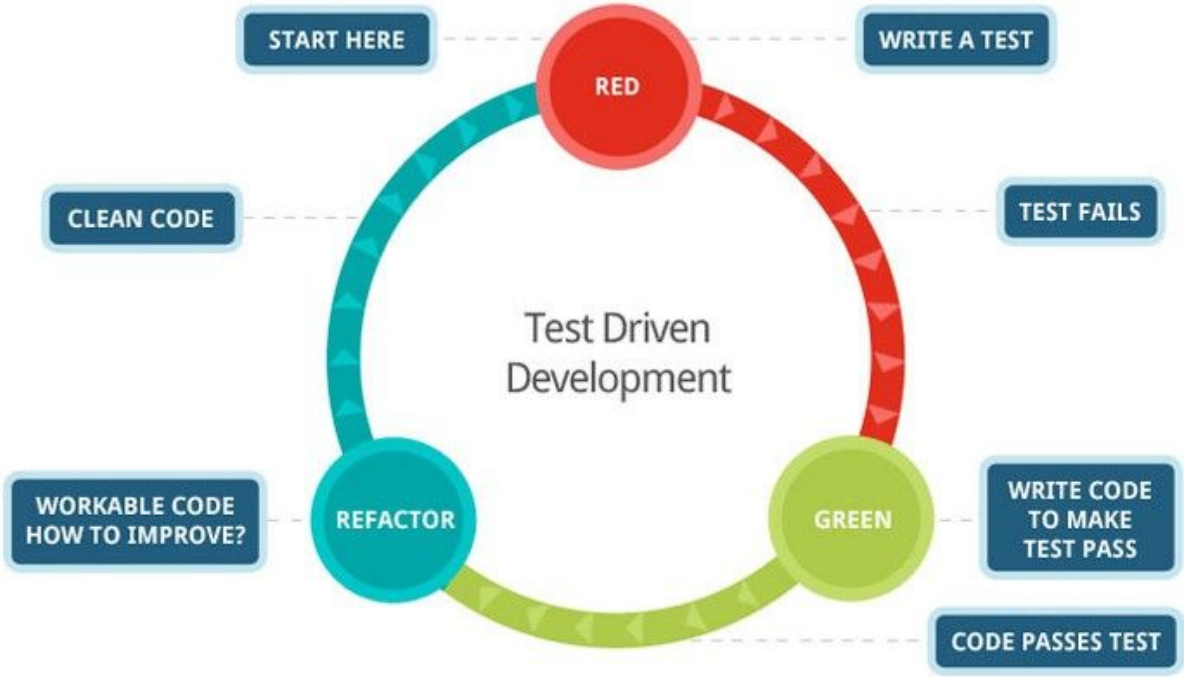


Reference Pictures:

- [Waterfall vs. Agile: A Comparison of Software Development Methodologies]

Reference Pictures:

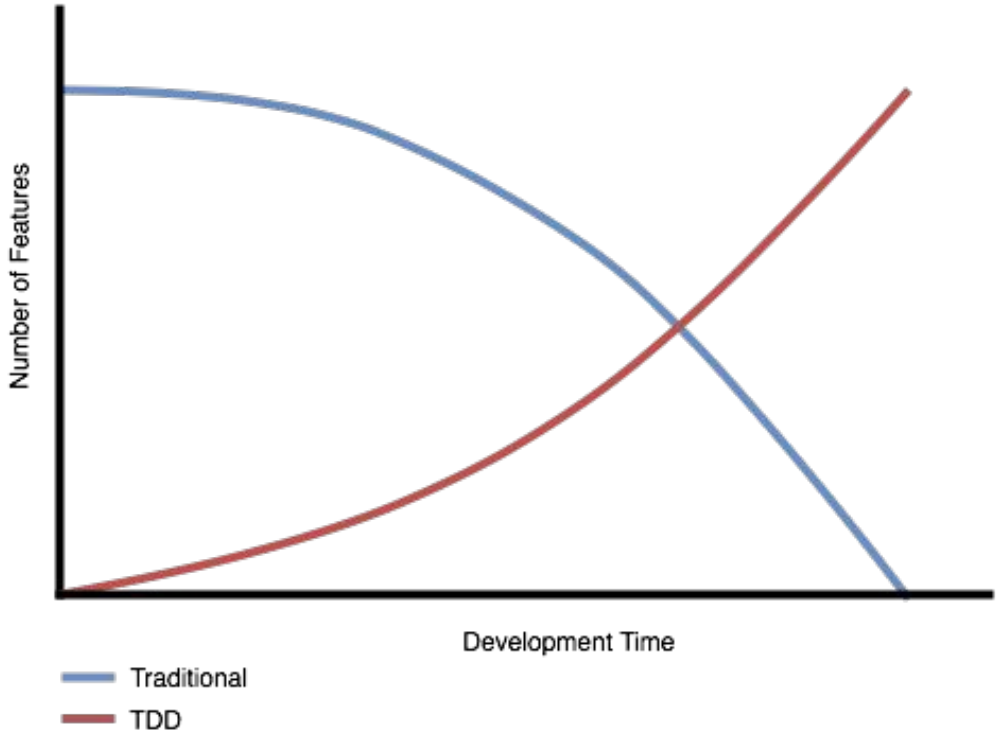
- [Where are the tests?]



Reference Pictures:

- [Test Driven Development — Understanding the business better]

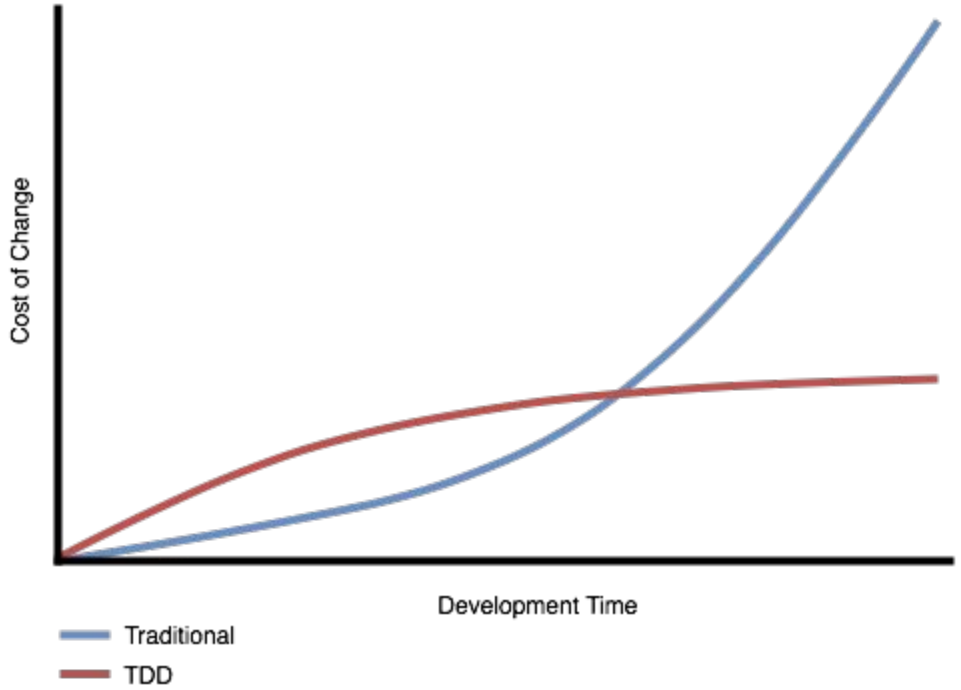
Cost of Change



Reference Pictures:

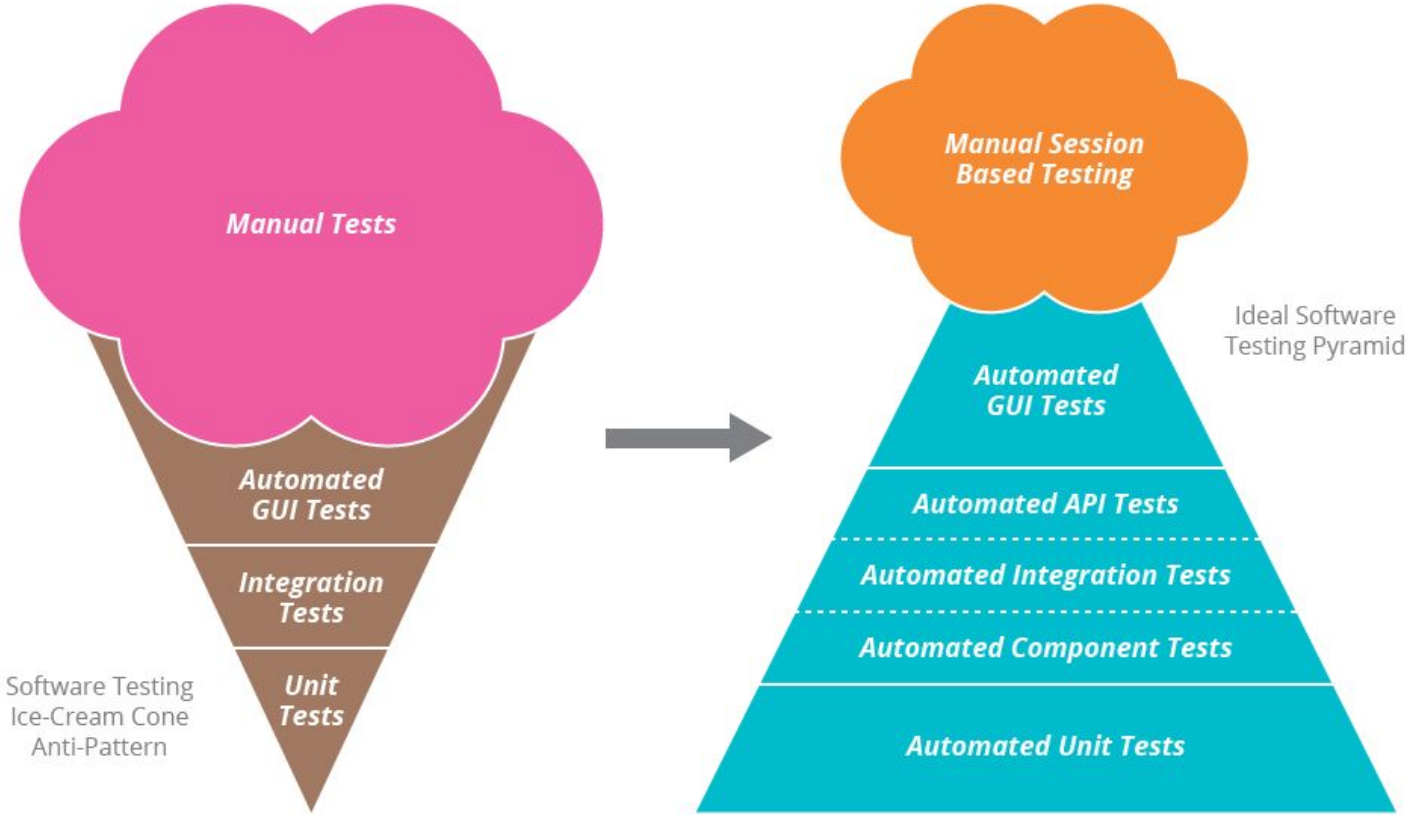
- The real reason why you don't like TDD

Number of Features



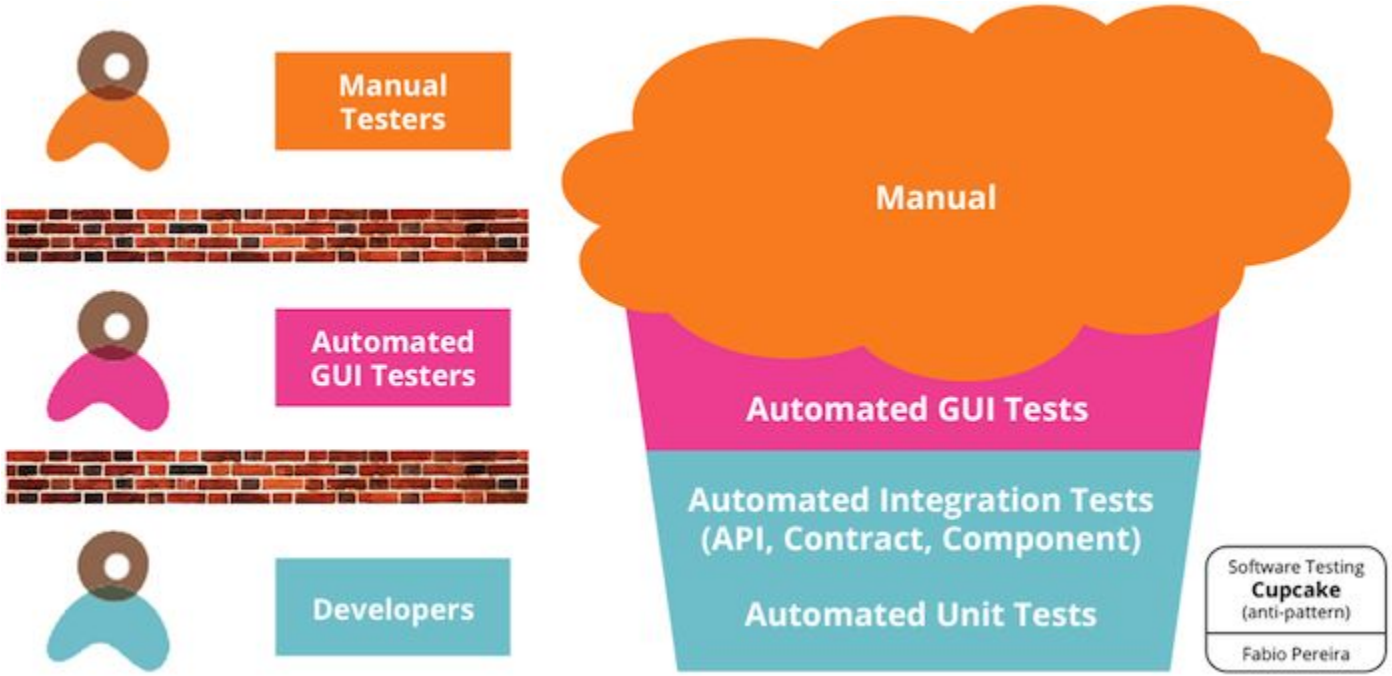
Reference Pictures:

- The real reason why you don't like TDD



Reference Pictures:

- [Is an inverted test pyramid really an anti-pattern?]



Reference Pictures:

- Introducing the Software Testing Cupcake (Anti-Pattern)

Pair programming

Reference Pictures:

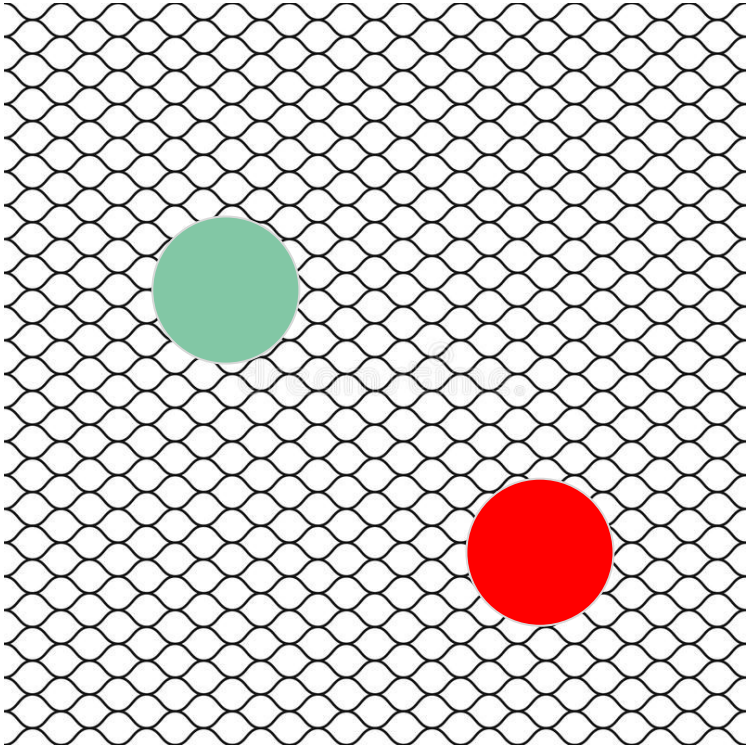
- Let's talk about Pair Programming

Defect

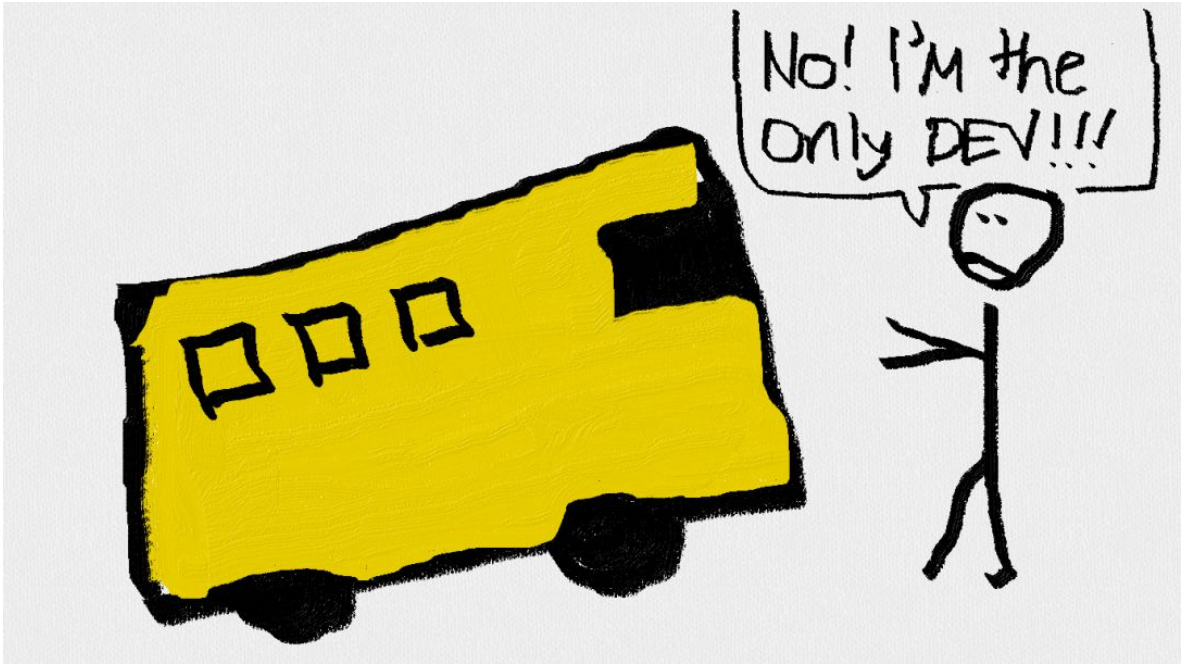
Reference Pictures:

- Net goal hole knots target grass

Fill the hole



Bus factor



Reference Pictures:

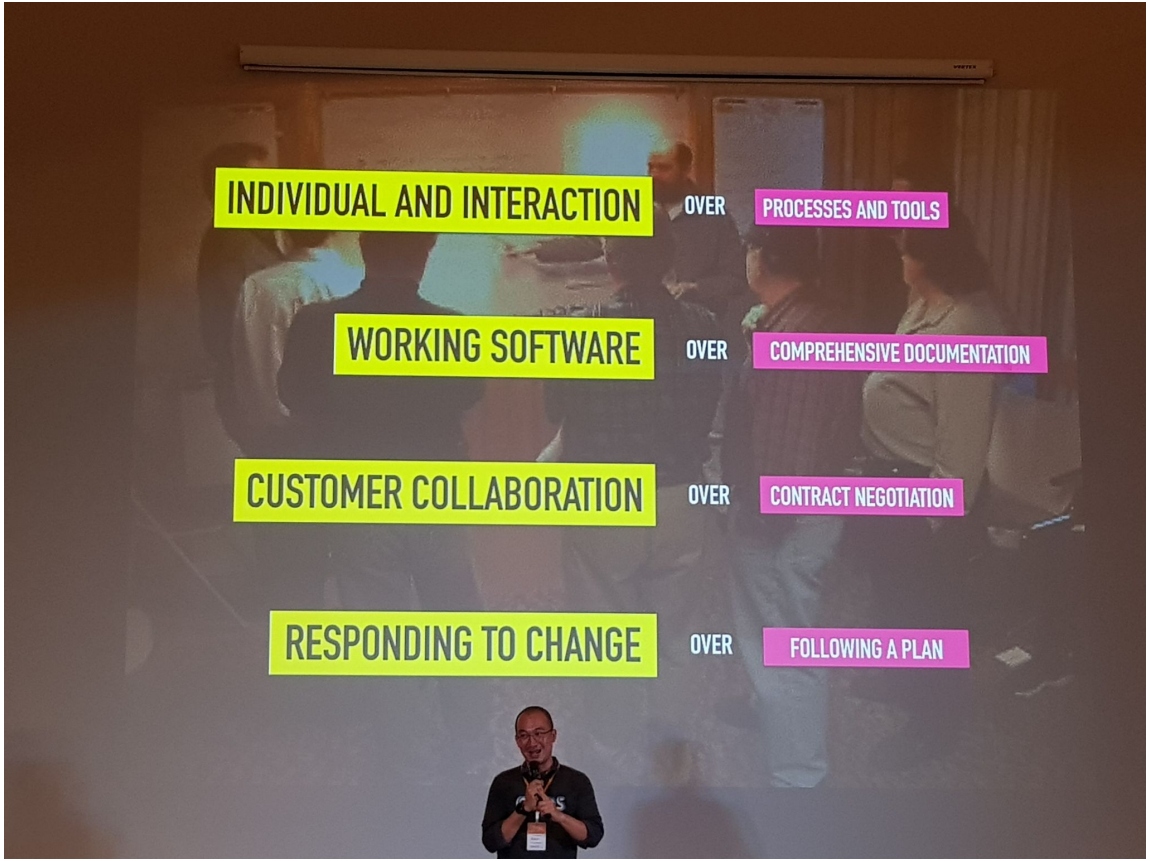
- The Bus Factor

Reference Pictures:

- When it comes to culture, one size does NOT fit all.

Reference Pictures:

- A little Friday Fun! Who can relate to this?



Reference:

- Twin Panitsombat @Beta Conference by Skooldio Session The Forgotten Manifesto

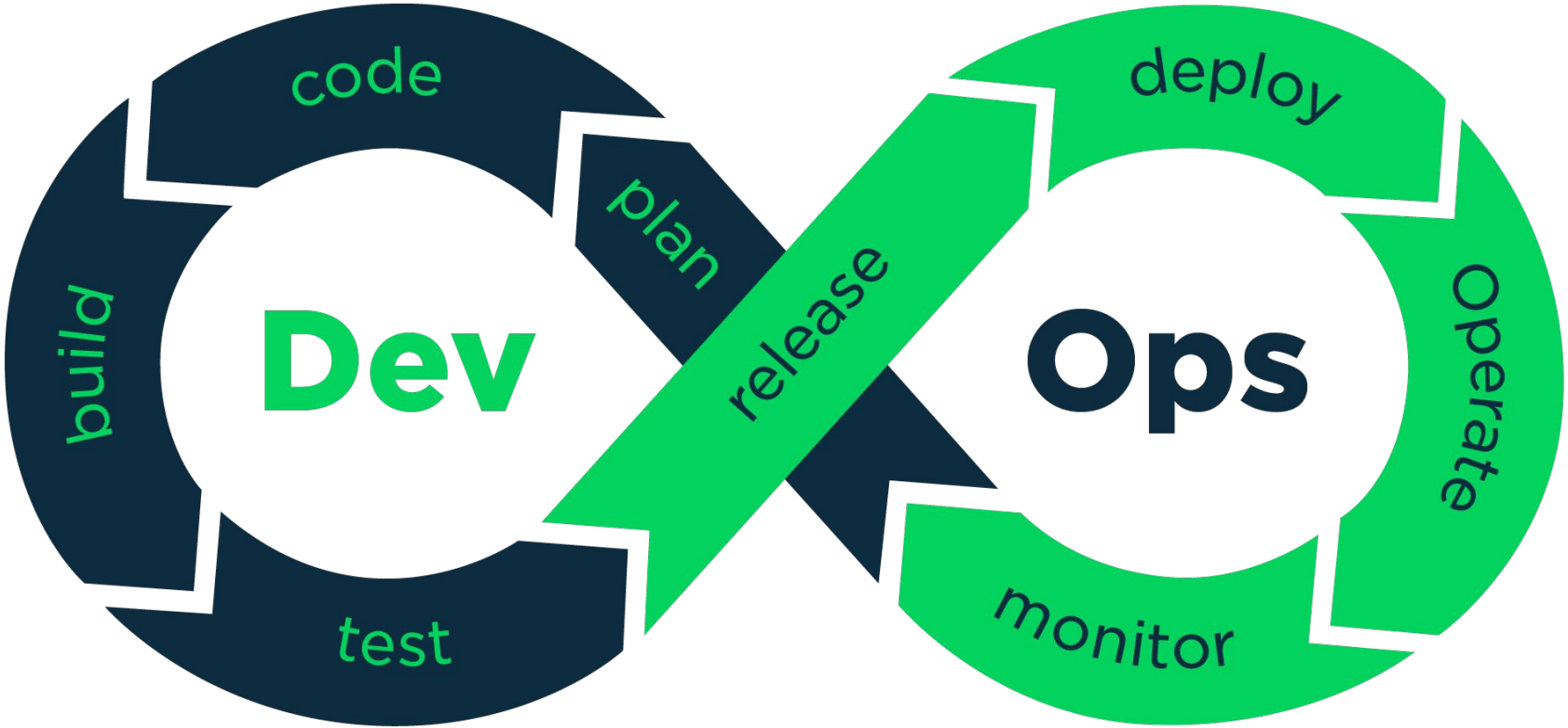
Jumpbox®

(working software)

Reference Pictures:

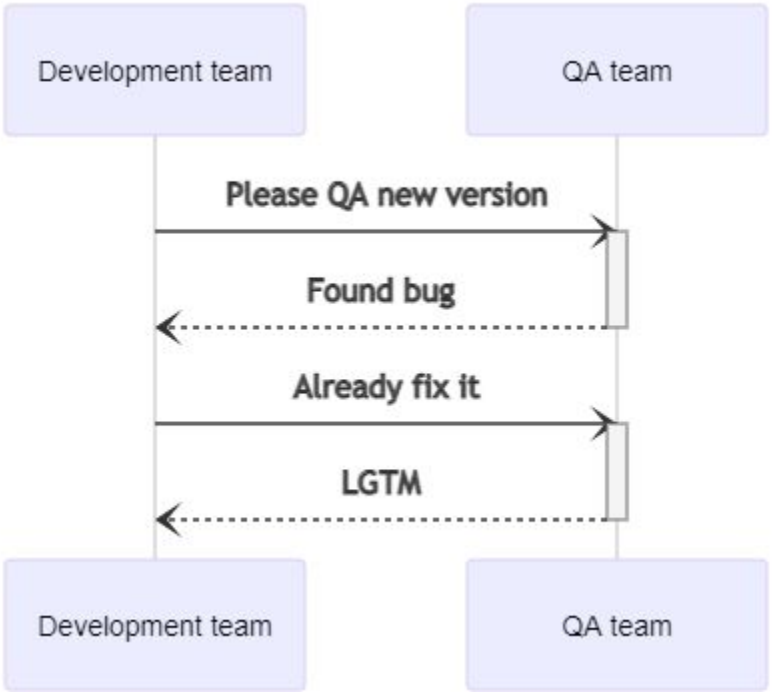
- Agile Software Development Basics and fundamentals

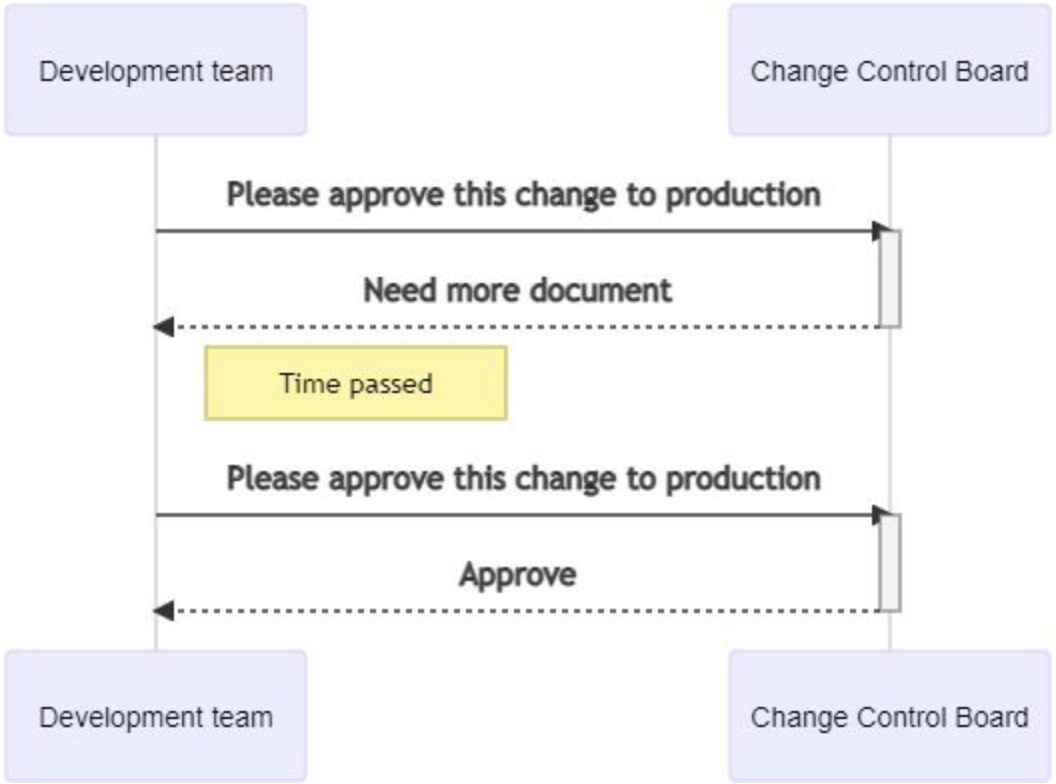
DevOps

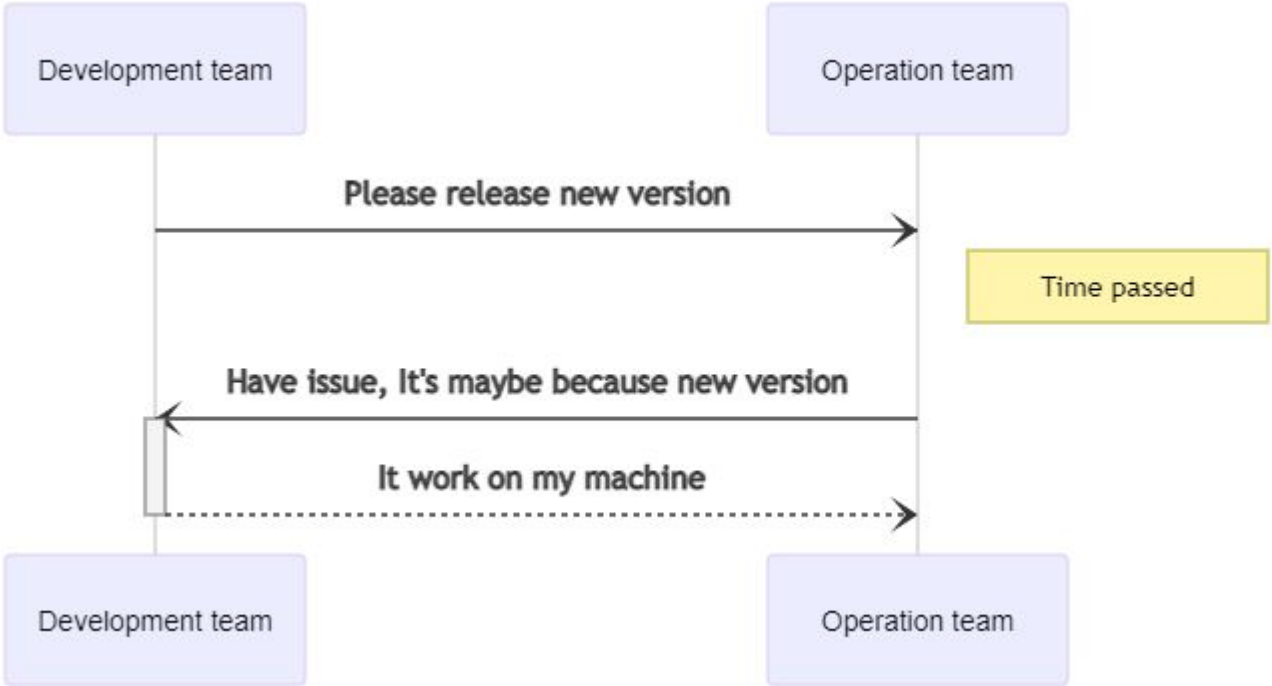


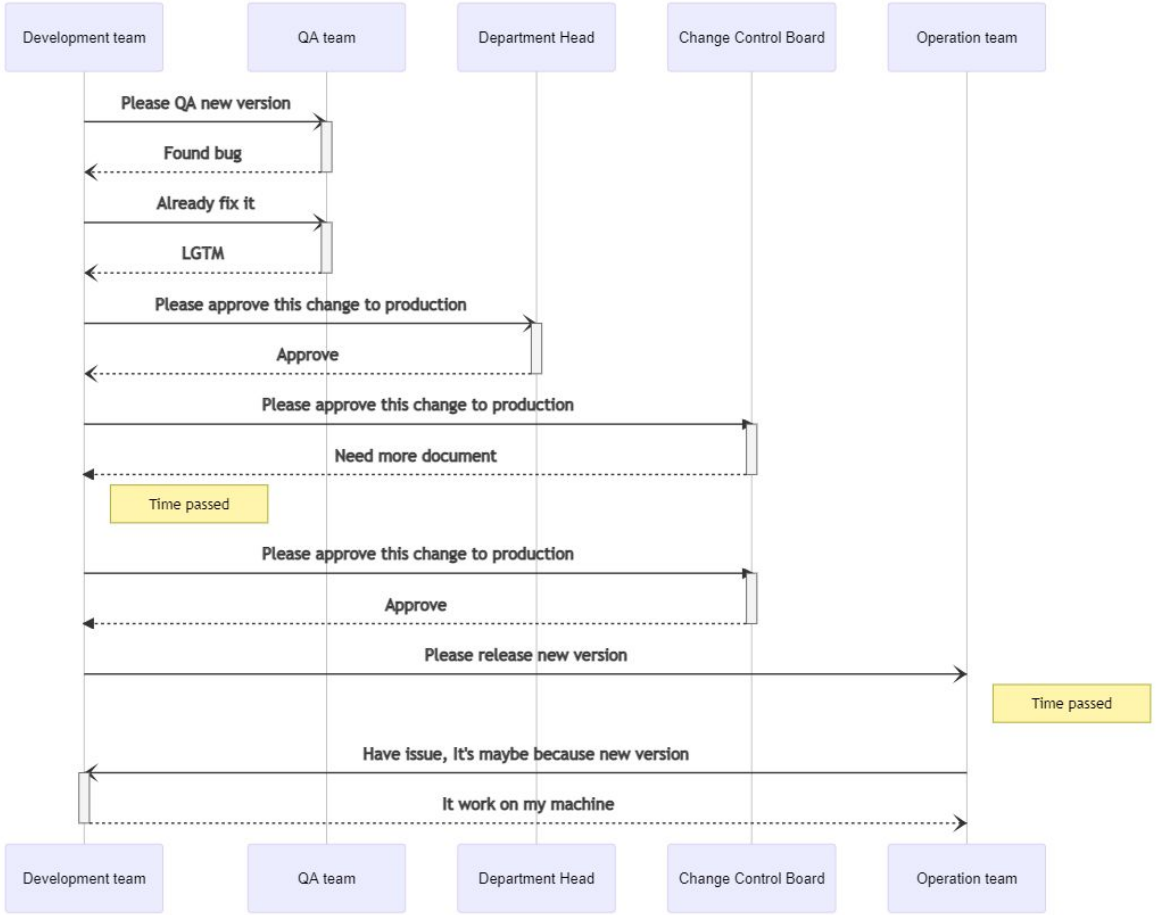
Reference Pictures:

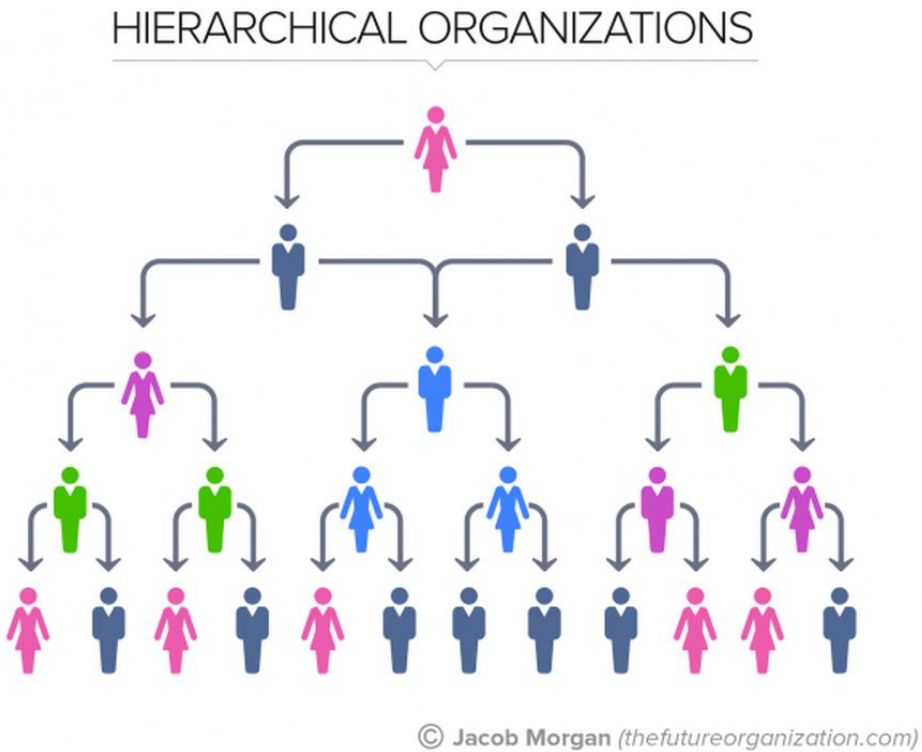
- ทำความรู้จักกับ DevOps







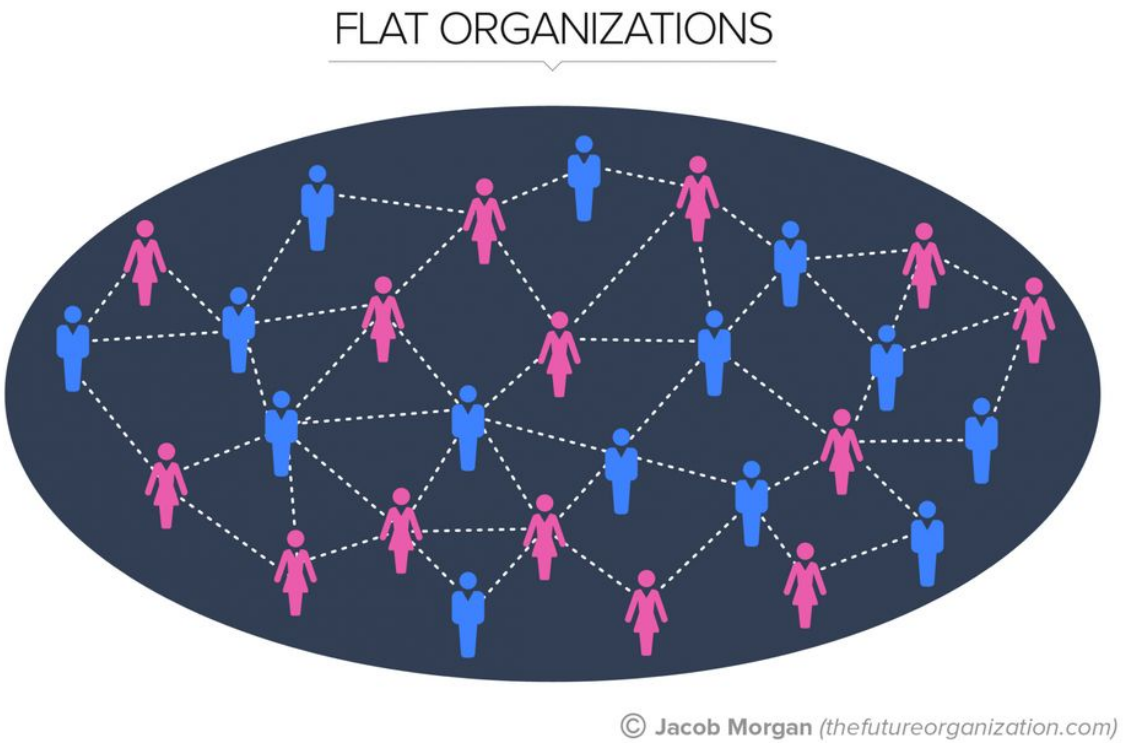




Reference Pictures:

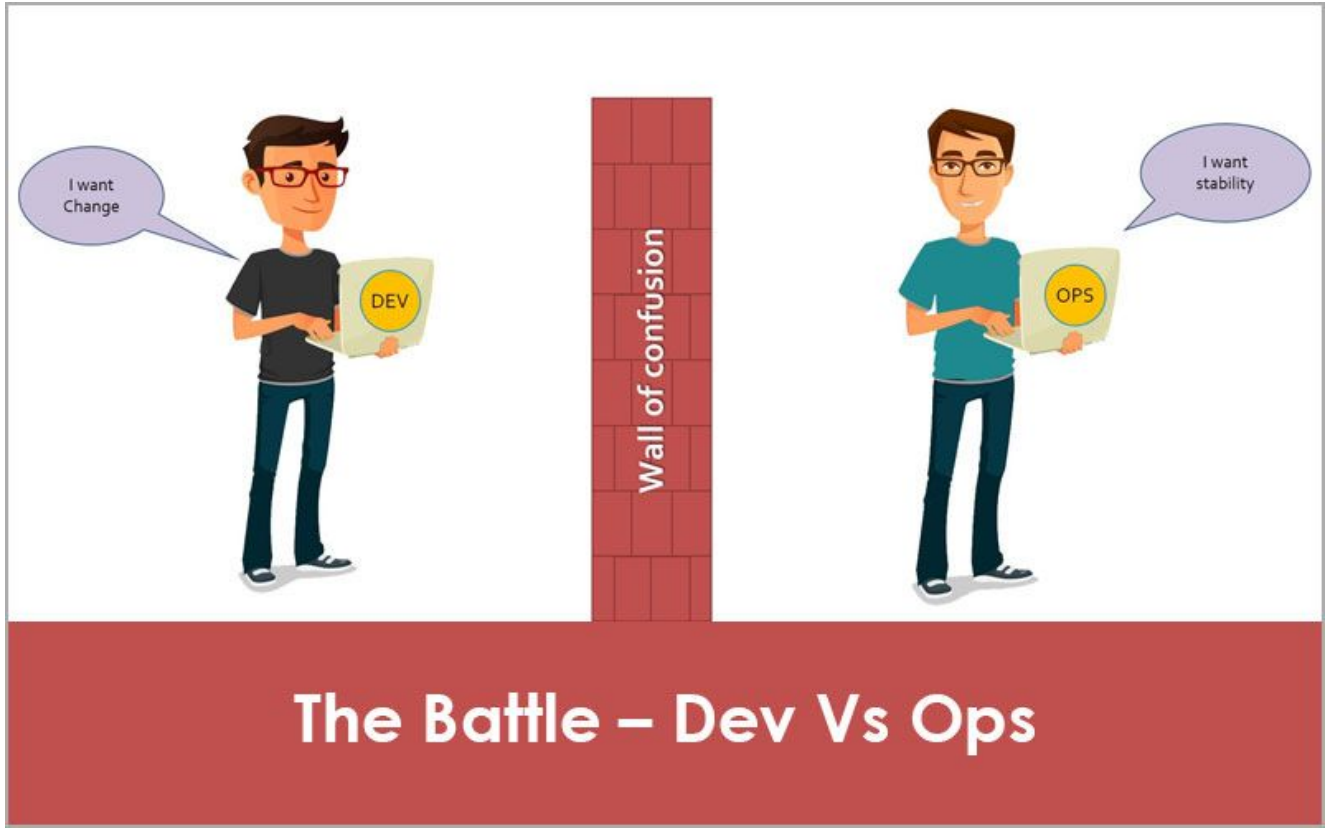
- The 5 Types Of Organizational Structures: Part 1, The Hierarchy

Jumpbox®



Reference Pictures:

- The 5 Types Of Organizational Structures: Part 3, Flat Organizations



Reference Pictures:

- [The Battle – Dev vs Ops]

The culture clash (Cont.)

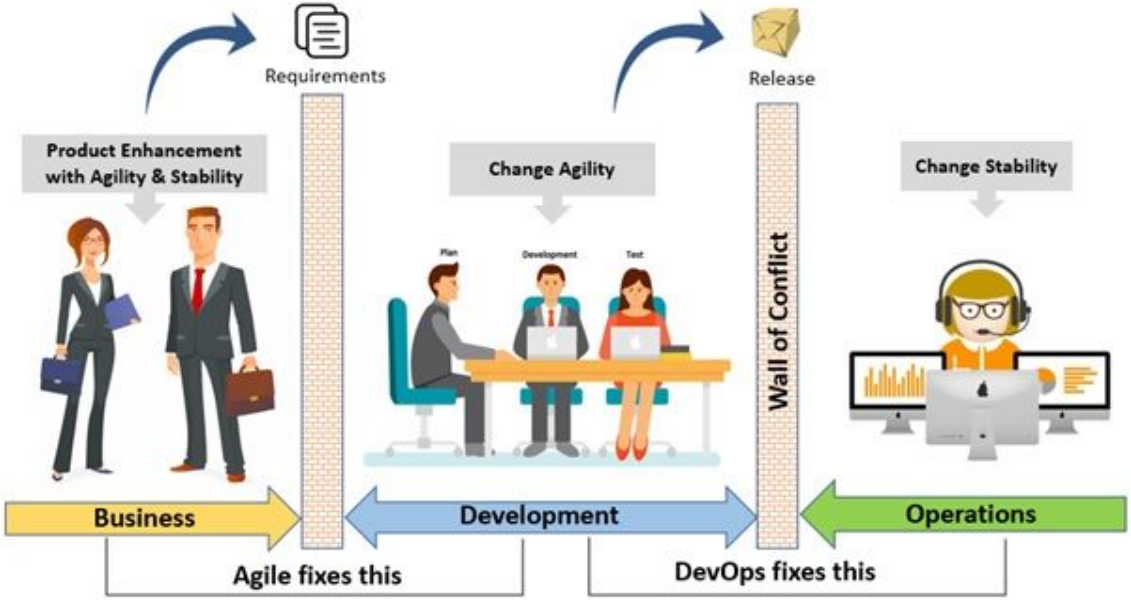
Developers

- Change
- Add or modify features
- Fix bugs

Operations

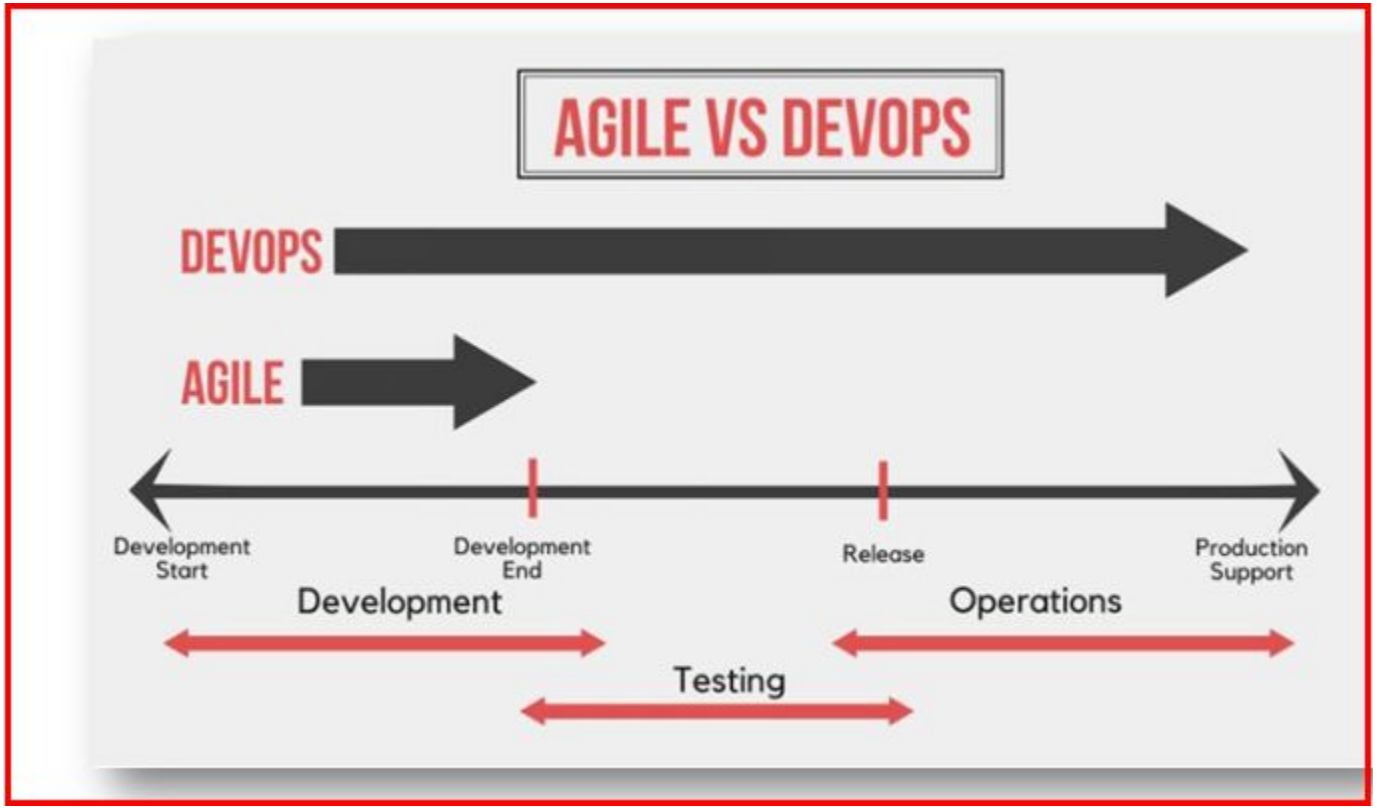
- Stability
- Create or enhance services
- Motivation to resist change

The culture clash (Cont.)



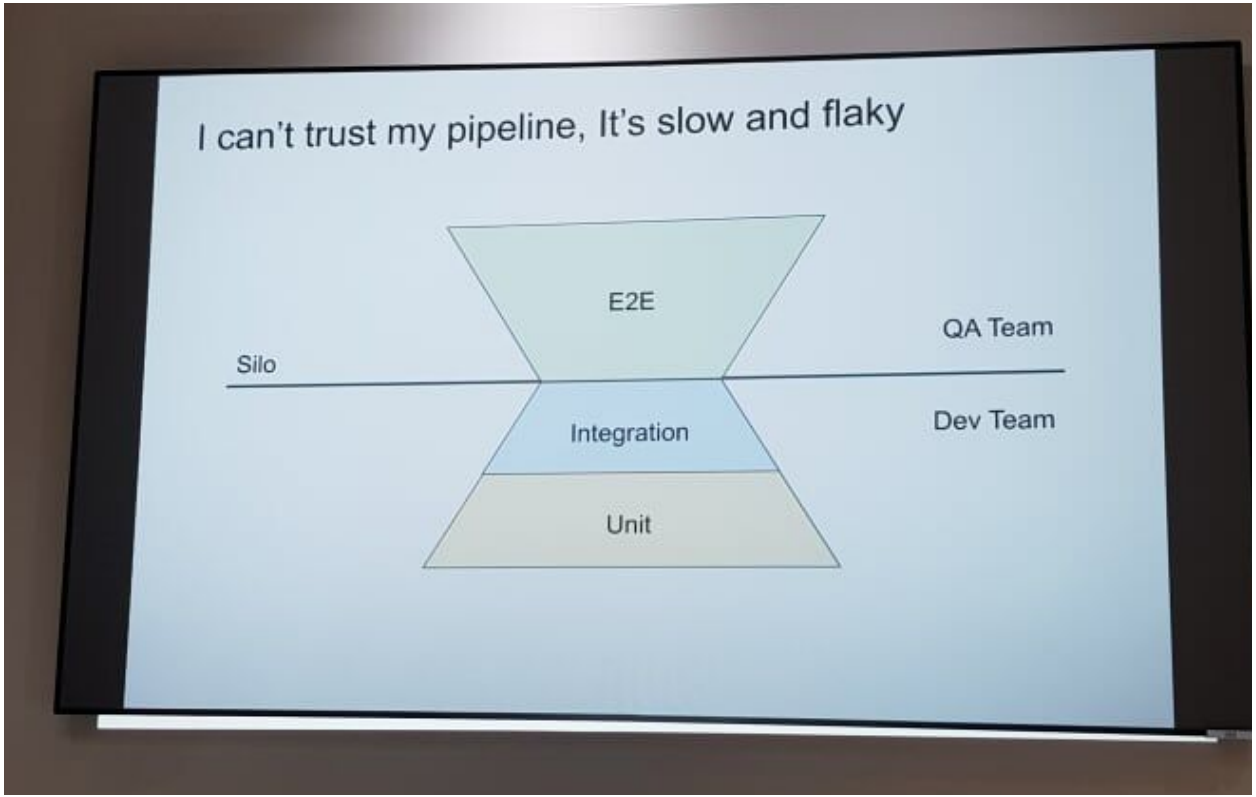
Reference Pictures:

- What does mean collaboration in DevOps culture?



Reference Pictures:

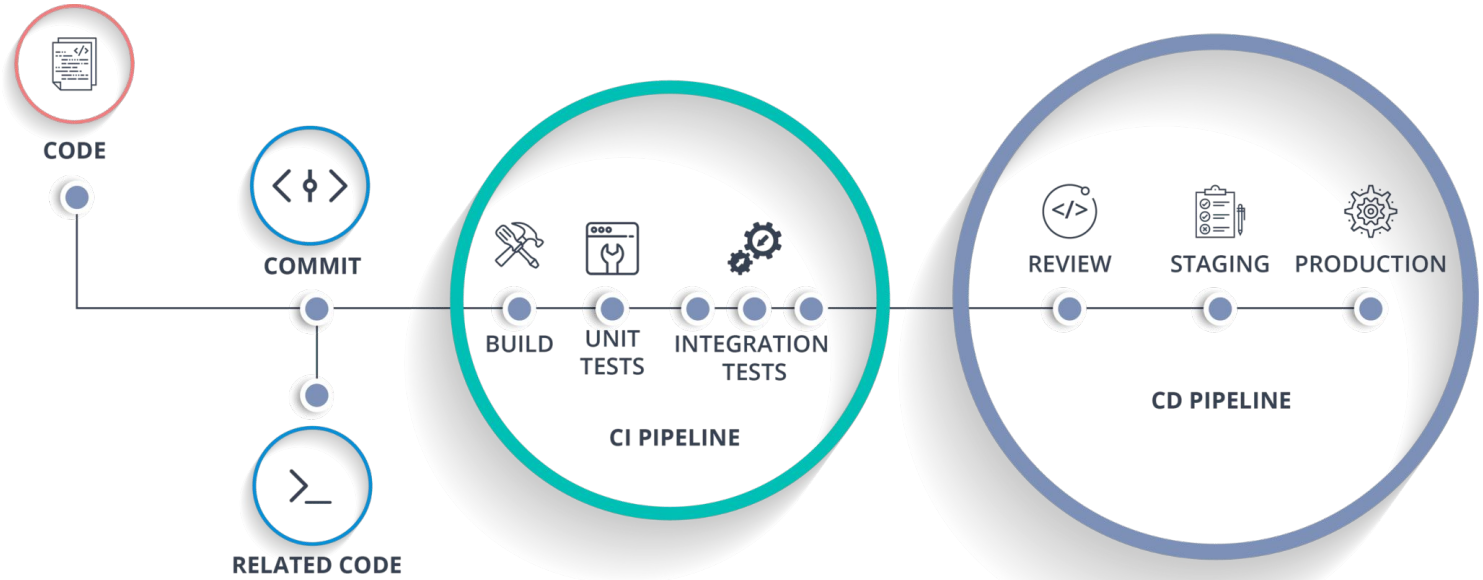
- Agile vs DevOps



Reference Pictures:

- CI/CD pipelines : The Good and Bad 190326 Thoughtworks

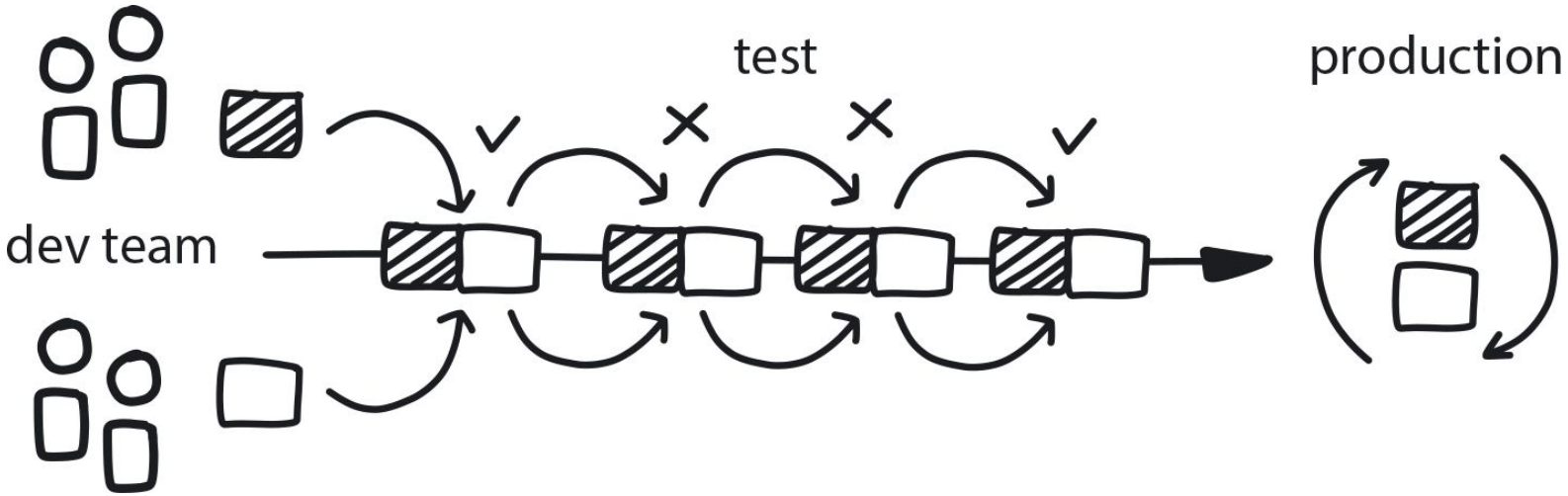
CI/CD



Reference Pictures:

- What is CI/CD Pipeline?

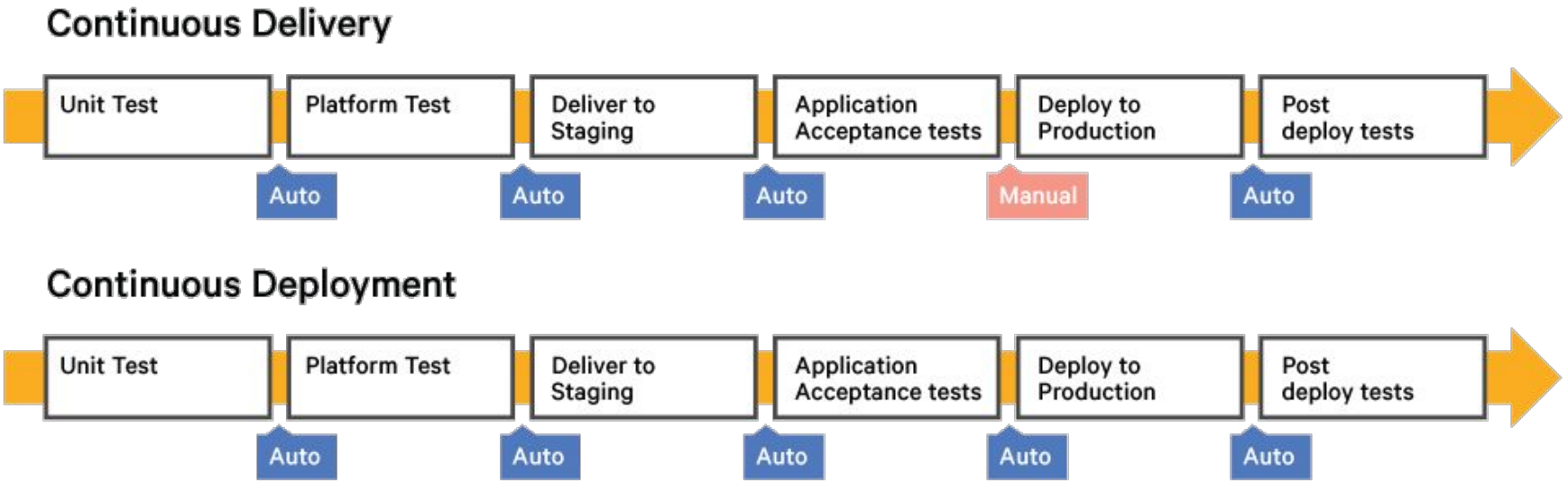
Continuous Integration (CI)



Reference Pictures:

- Cloud Native Transformation

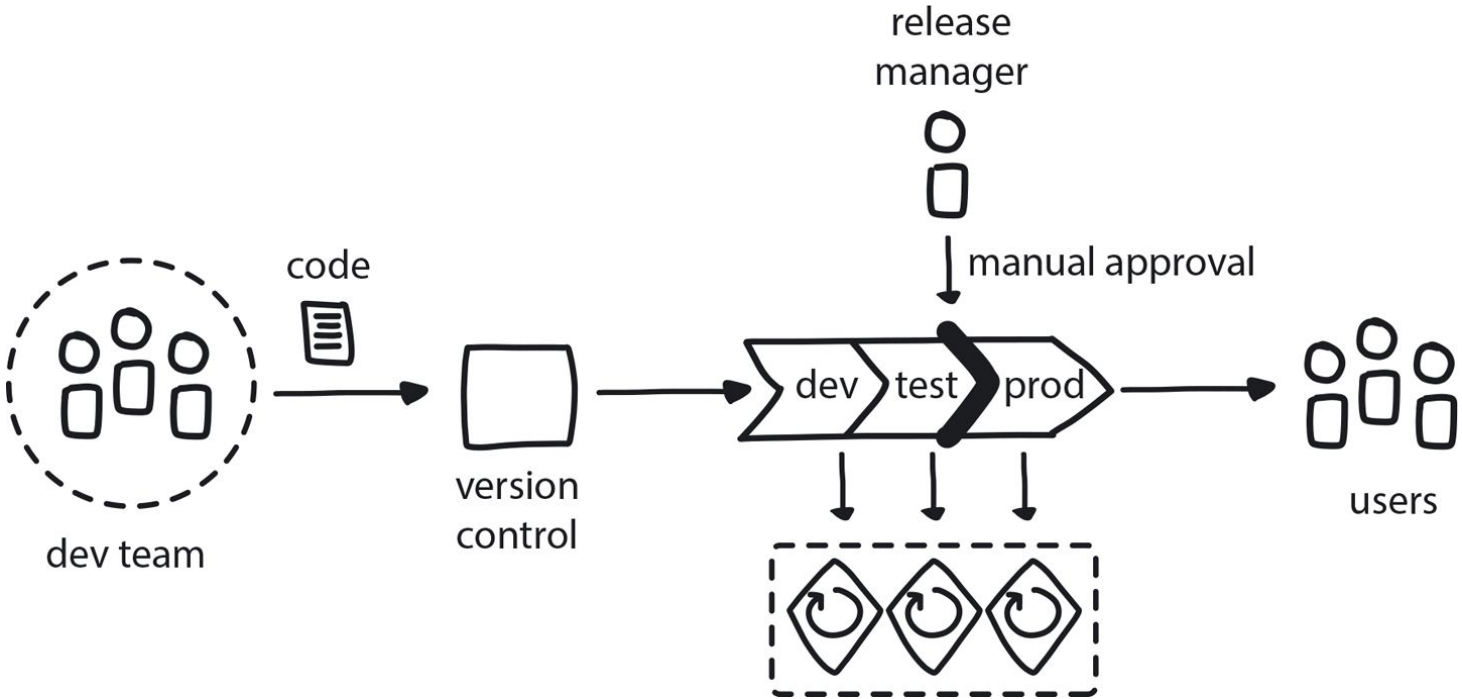
CD



Reference Pictures:

- Continuous Delivery vs. Continuous Deployment: An Overview

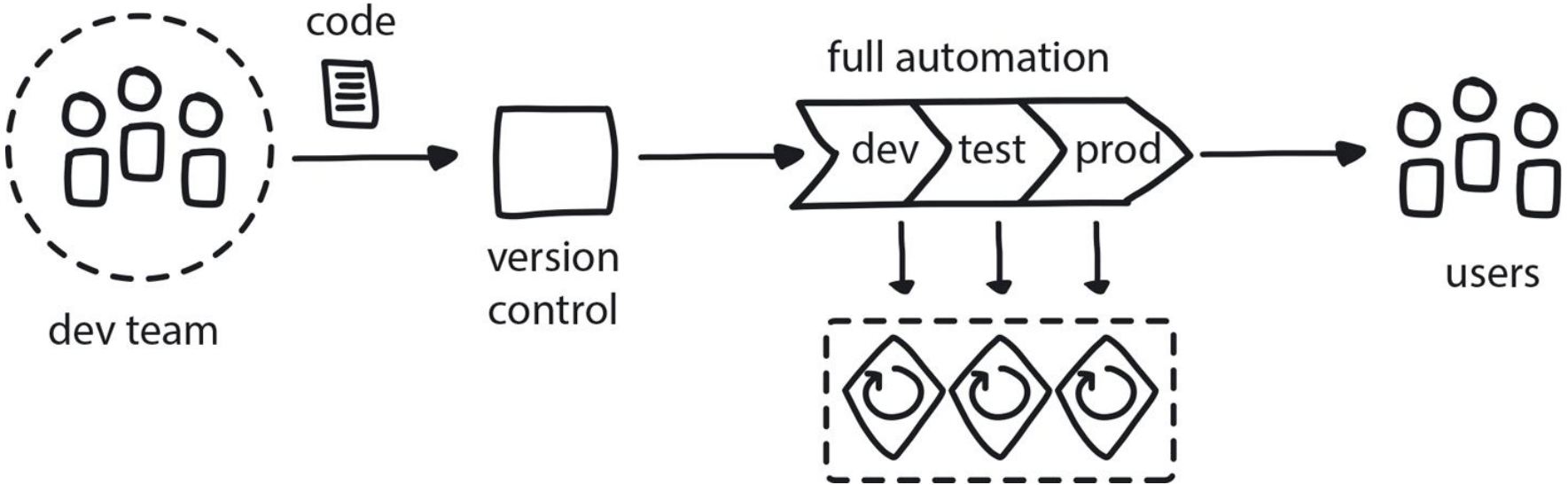
Continuous Delivery (CD)



Reference Pictures:

- Cloud Native Transformation

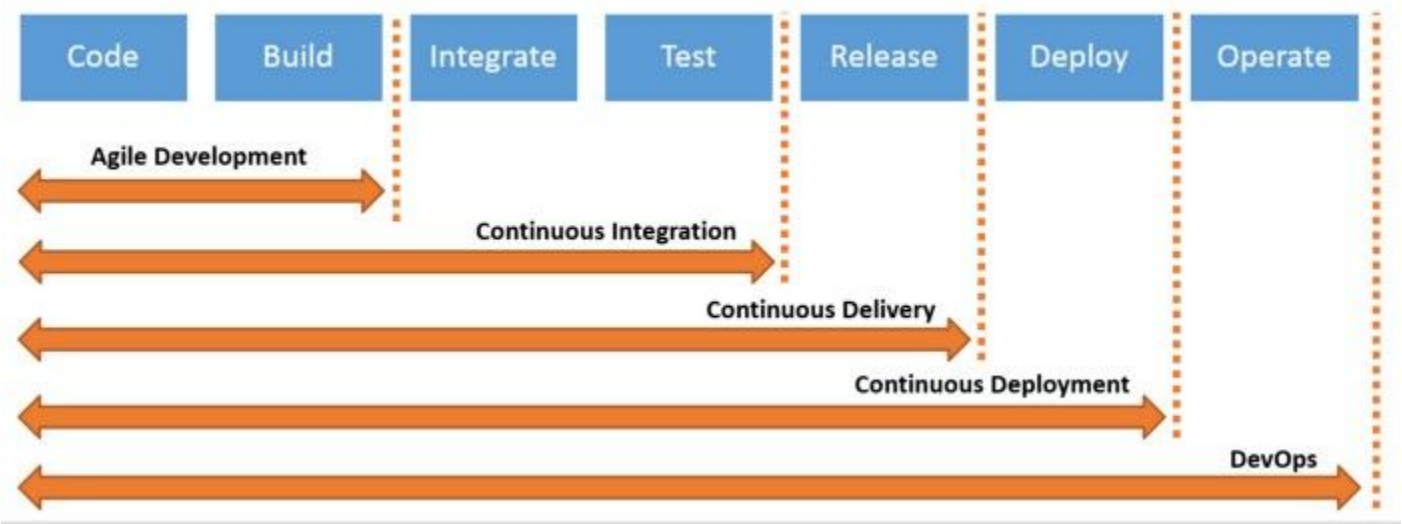
Continuous Deployment (CD)



Reference Pictures:

- Cloud Native Transformation

DevOps



Reference Pictures:

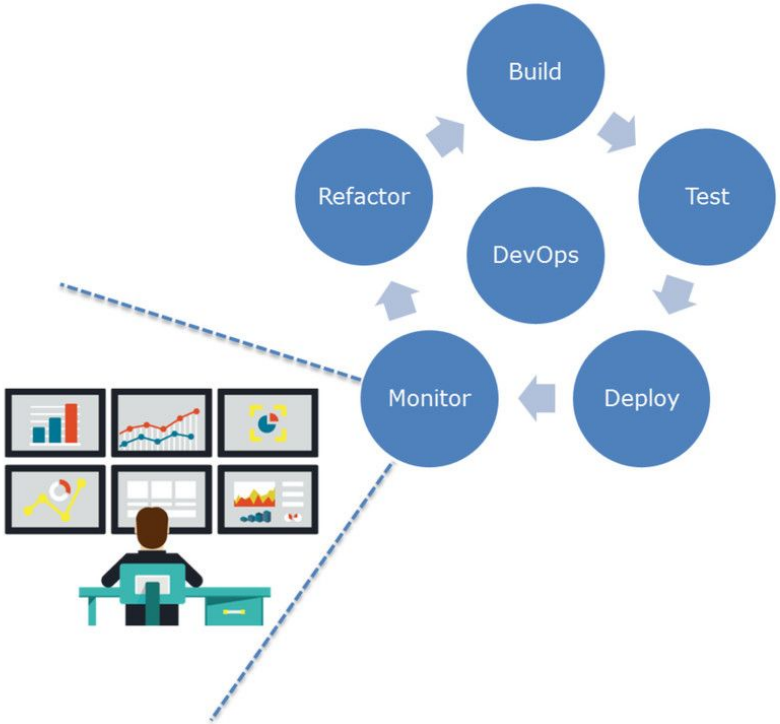
- Continuous Integration / Delivery / Deployment (CI / CD)

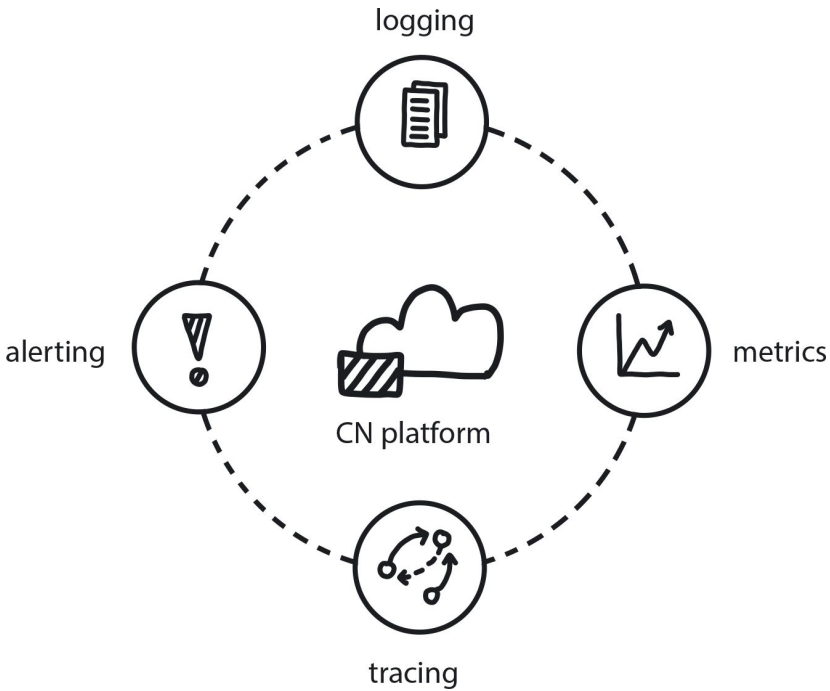
DevOps Culture

Reference Pictures:

- [Continuous Monitoring: The Role of DevOps and APM]

Monitor and Observability

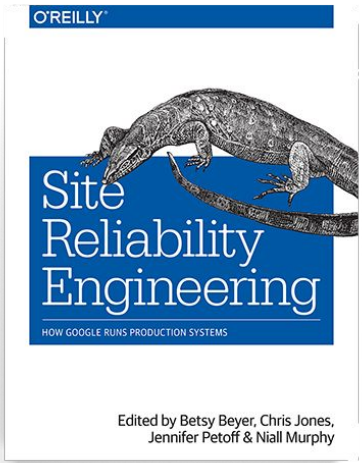




Reference Pictures:

- Enterprise Monitoring in DevOps
- Cloud Native Transformation

Site Reliability Engineering (SRE)



SLIs drive **SLOs** which inform **SLAs**

Reference Pictures:

- [Principles of Site Reliability Engineering at Google]
- [SLIs, SLOs, SLAs, oh my! (class SRE implements DevOps)]

Jumpbox®

Example SLA

SLA	Allowed Monthly Downtime	Allowed Annual Downtime
99.9% (Classic VPNs)	43m 49.7s	8h 45m 57.0s
99.95% (Other cloud providers)	21m 54.9s	4h 22m 58.5s
99.99% (Google Cloud HA VPN)	**4m 23.0s**	**52m 35.7s**

Reference Pictures:

- Google Cloud networking in depth: Faster, more reliable connectivity with HA VPN and 100 Gbps Dedicated Interconnect

Uptime and downtime with 99.9 % SLA

[*simple* / flexible / reverse / @uptisbot 🤖]

Agreed SLA level: 99.9 **%** (enter SLA level and hit the <enter> key)

SLA level of 99.9 % uptime/availability results in the following periods of allowed downtime/unavailability:

- **Daily:** 1m 26s
- **Weekly:** 10m 4s
- **Monthly:** 43m 49s
- **Quarterly:** 2h 11m 29s
- **Yearly:** 8h 45m 56s

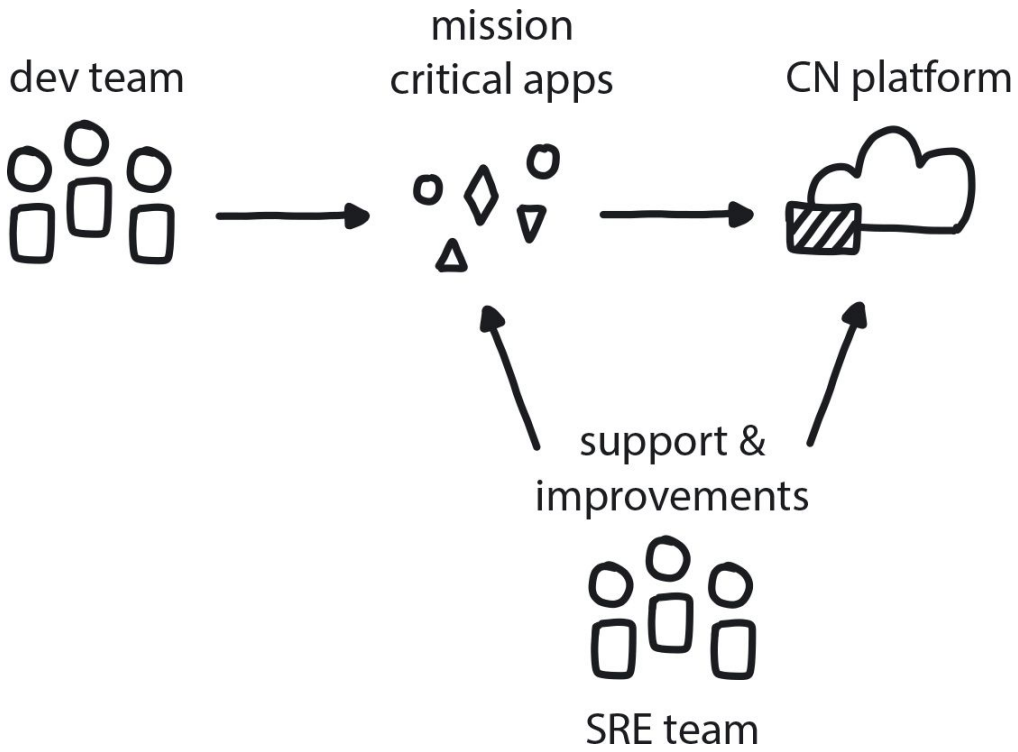
Direct link to page with these results: **uptime.is/99.9** (or uptime.is/three-nines)

The SLA calculations assume a requirement of continuous uptime (i.e. 24/7 all year long) with additional approximations as described in the source.

For convenience, there are special CEO and SEO friendly links for *N nines*: three nines, four nines, five nines, six nines etc.

Link: https://uptime.is/

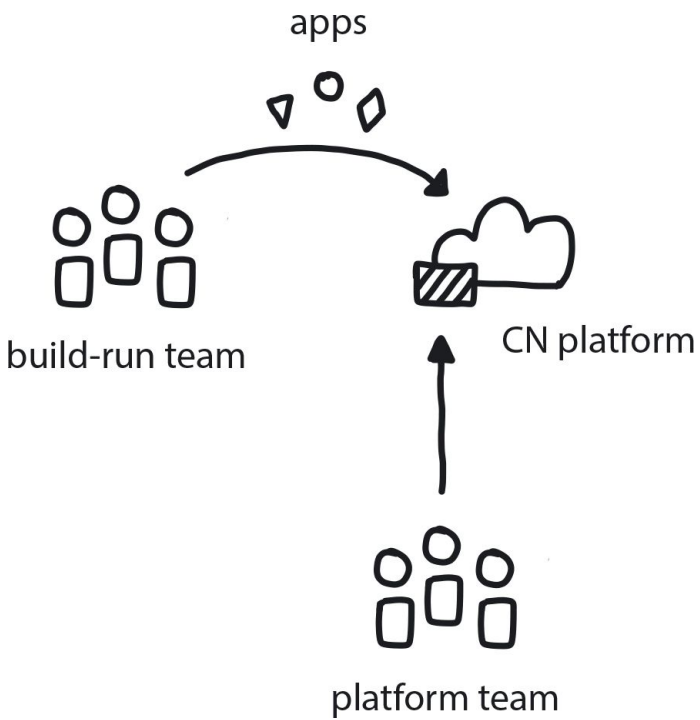
Site Reliability Engineering (SRE) (Cont.)

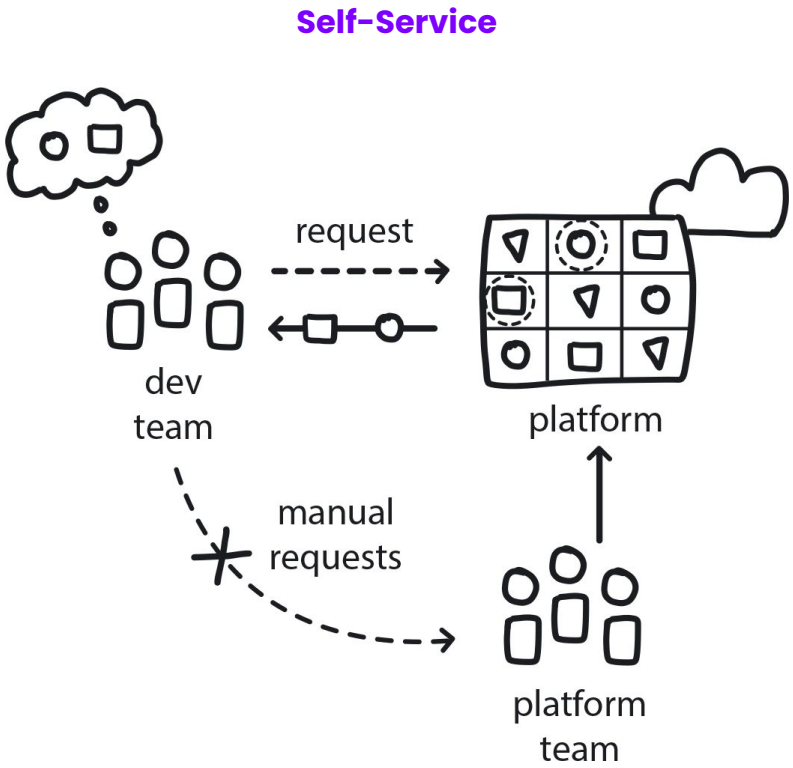


Reference Pictures:

- Cloud Native Transformation

Platform Engineer (PE) (Cont.)



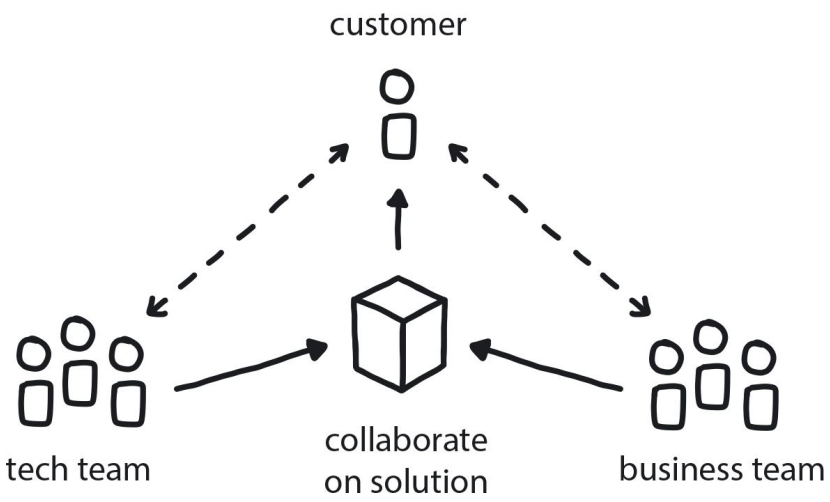


Reference Pictures:

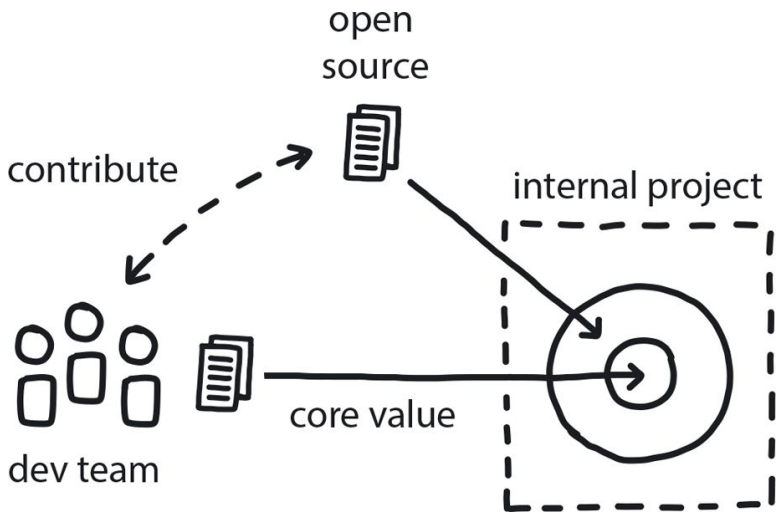
- [Cloud Native Transformation]

Internal Supply Chains

Involve the Business



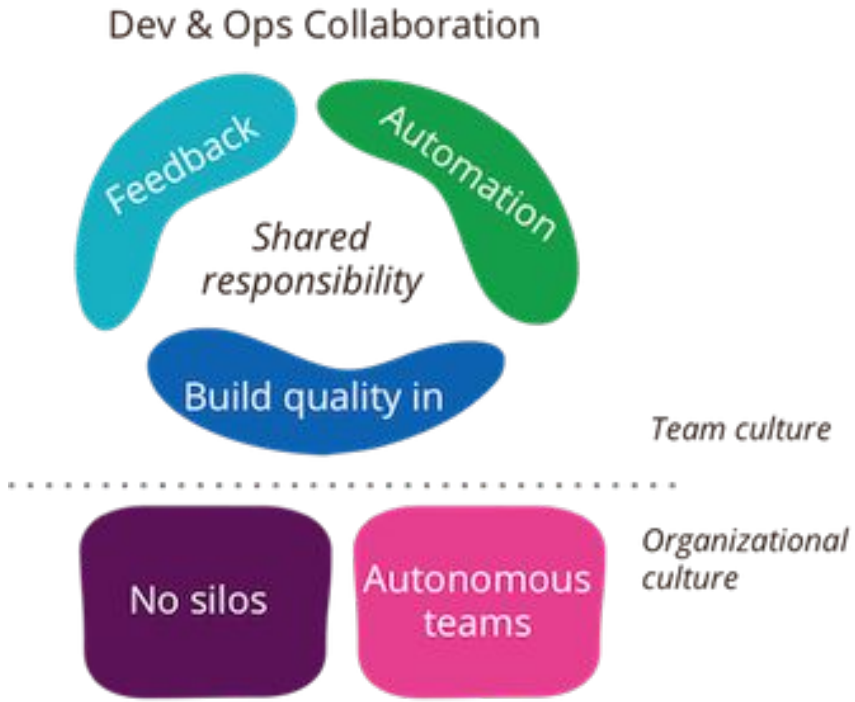
Open Source Internal Projects



Reference Pictures:

- Cloud Native Transformation

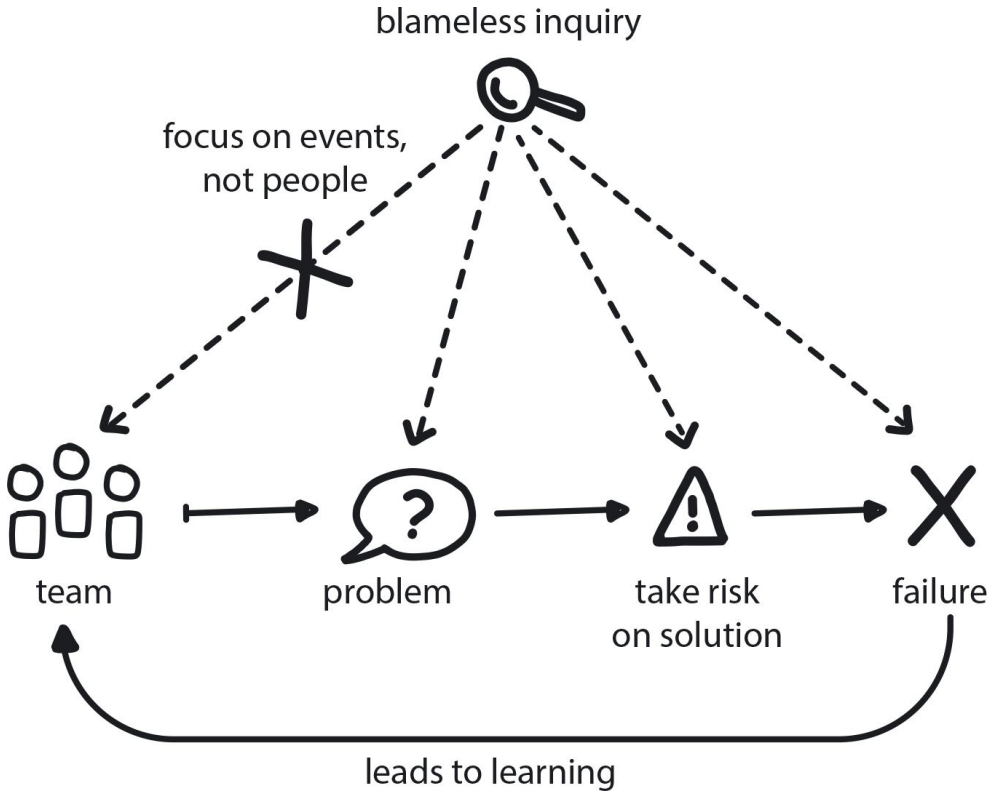
What is DevOps?



Reference Pictures:

- Dev Ops Culture

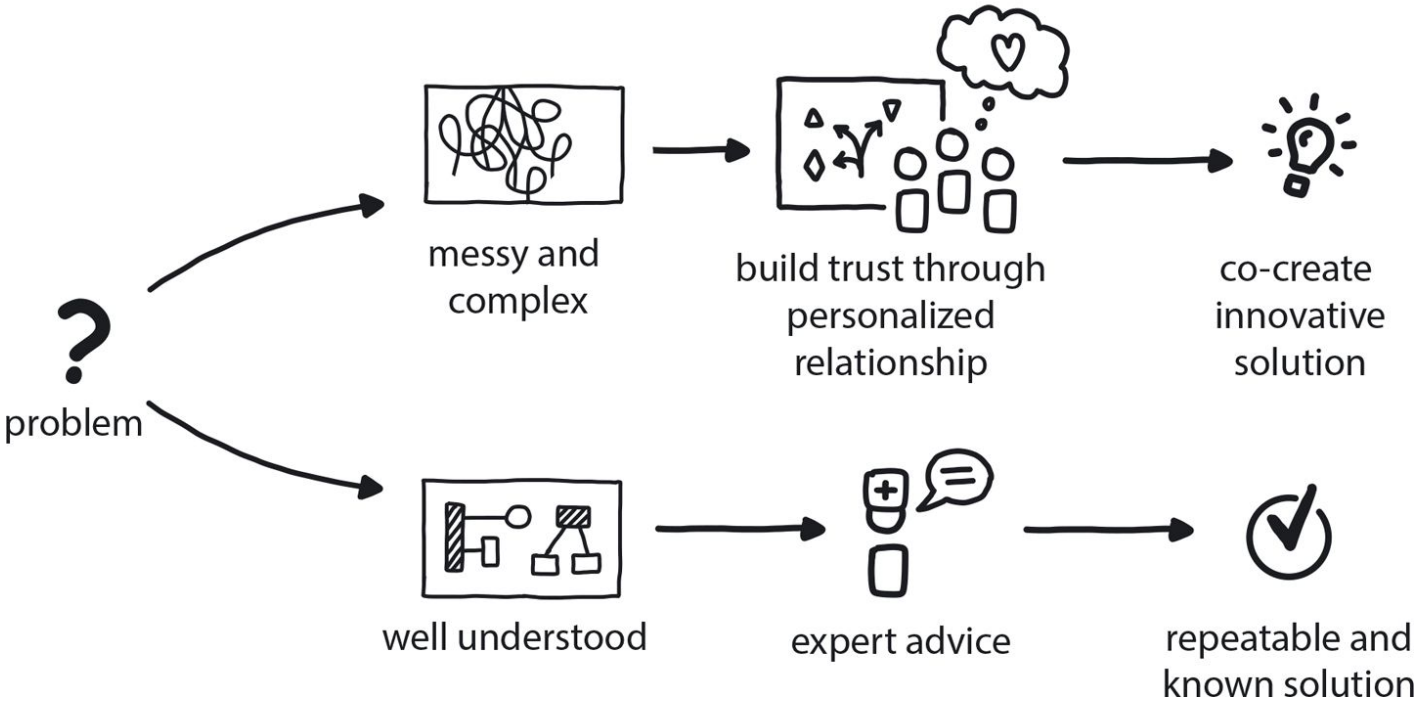
Blameless Culture



Reference Pictures:

- Cloud Native Transformation

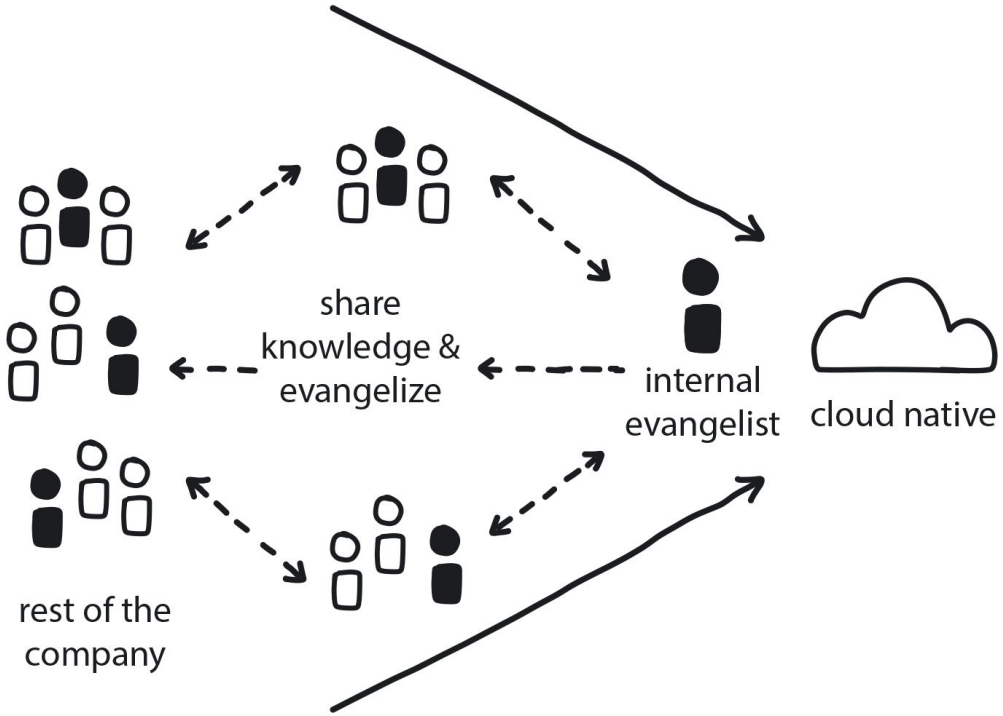
Personalized Relationships for Co-Creation



Reference Pictures:

- Cloud Native Transformation

Internal Evangelism



Reference Pictures:

- Cloud Native Transformation

DevOps Environment

The four key metrics

2017	High Performers	Medium Performers	Low Performers
Deployment Frequency	On demand (multiple deploys per day)	Between once per week and once per month	Between once per week and once per month*
Lead Time for Changes	Less than one hour	Between one week and one month	Between one week and one month*
MTTR	Less than one hour	Less than one day	Between one day and one week
Change Failure Rate	0–15%	0–15%	31–45%

* Low performers were lower on average (at a statistically significant level) but had the same median as the medium performers.

Reference Pictures:

- [Learning from the Accelerate "Four Key Metrics"]

Feedback Loops

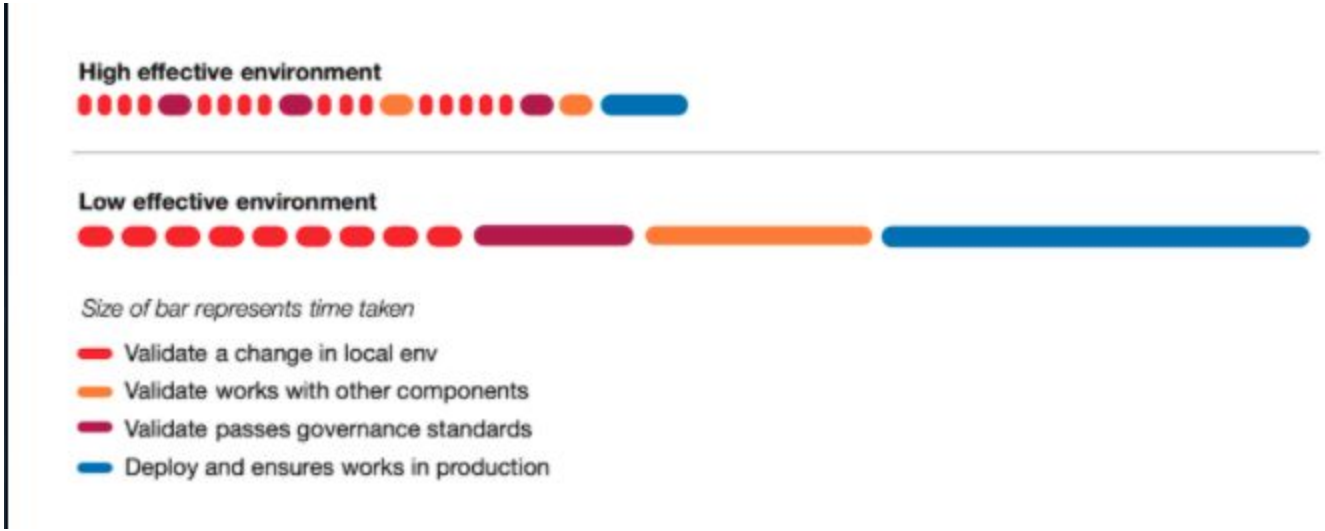
The key loops I have identified are:

Feedback Loop	Low Effectiveness	High Effectiveness
Validate a local code change works	2 mins	5-15 seconds (depending on tech choice)
Find root cause for defect	4-7 days	1 day
Validate component integrates with other components	3 days - 2 weeks	2 hours
Validate a change meets non-functional requirements	3 months	1 day - 1 week (depending on scope of change)
Become productive on new team	2 months	4 weeks
Get answers to an internal technical query	1-2 weeks	30 mins
Launch a new service in production	2-4 months	3 days
Validate a change was useful to the customer	6 months or never	1 - 4 weeks (depending on scope of change)

Reference Pictures:

- Maximizing Developer Effectiveness

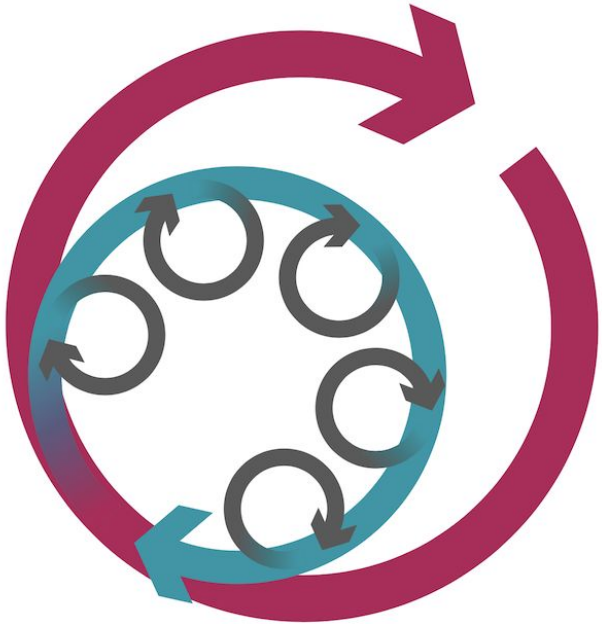
Time taken



Reference Pictures:

- Maximizing Developer Effectiveness

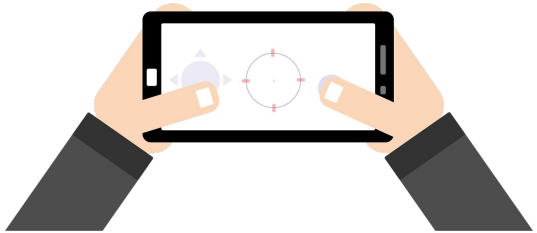
Micro feedback loops



Reference Pictures:

- Maximizing Developer Effectiveness

Distract



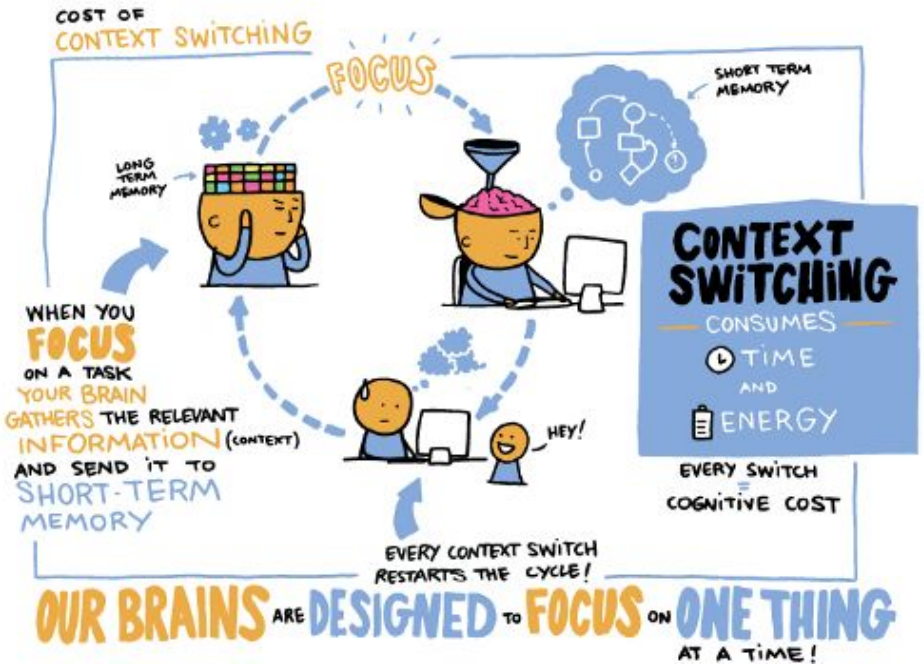
Multitasking

Reference Pictures:

- Walk the Week - Embrace Manic Mondays

Jumpbox®

Context switching



Reference Pictures:

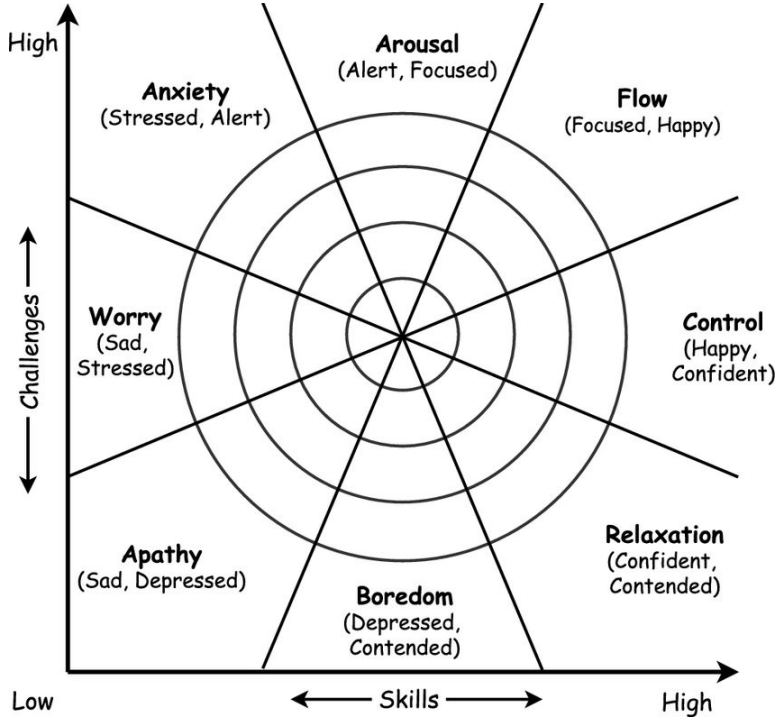
- [Why you should limit work in progress and stop multitasking]

Get back into the state of flow

Reference Pictures:

- The Cost of Interrupted Work: More Speed and Stress

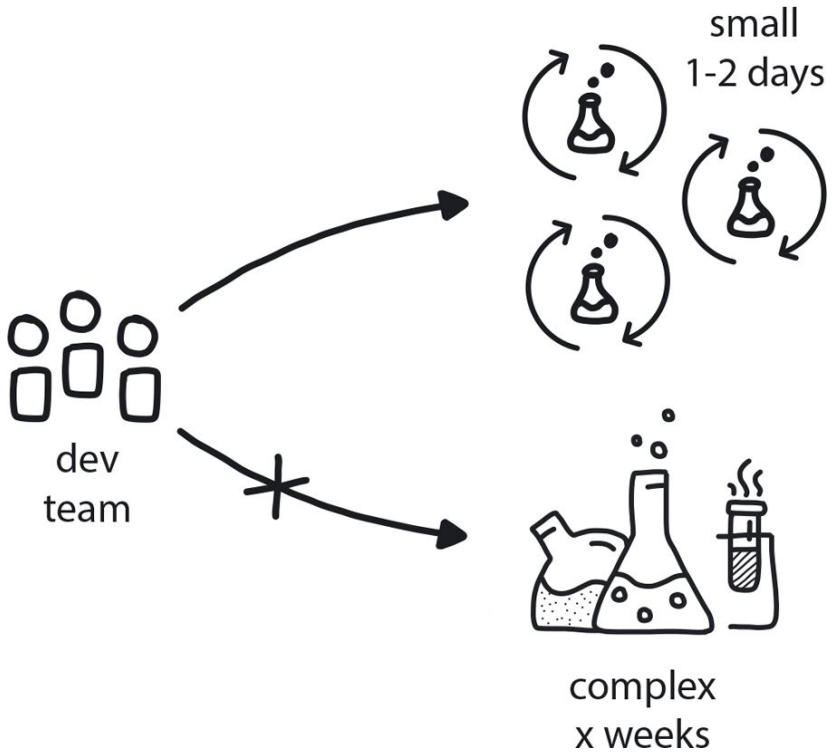
Flow (psychology)



Reference Pictures:

- [Balanced difficulty task finder: an adaptive recommendation method for learning tasks based on the concept of state of flow]

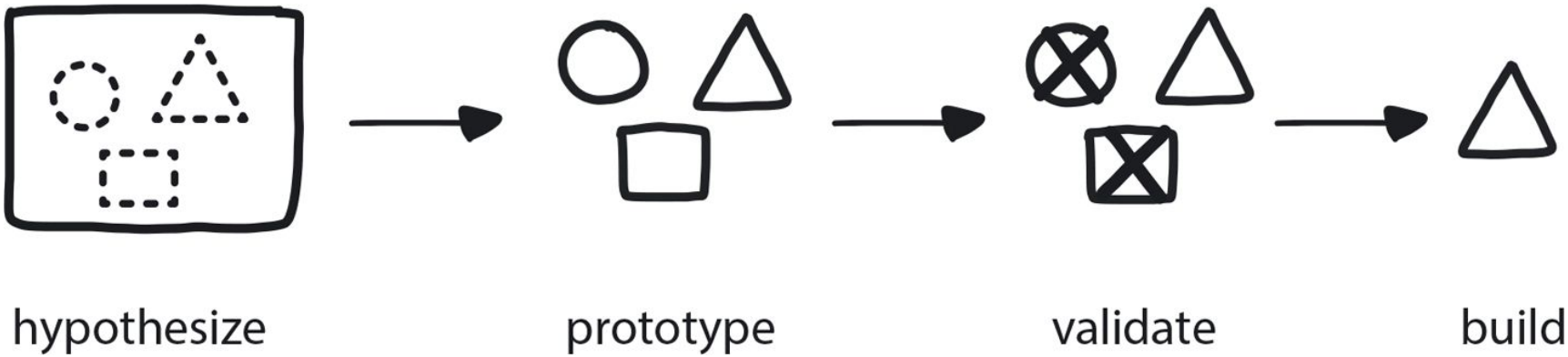
Reduce Cost of Experimentation



Reference Pictures:

- Cloud Native Transformation

Proof of Concept (PoC)



Reference Pictures:

- Cloud Native Transformation

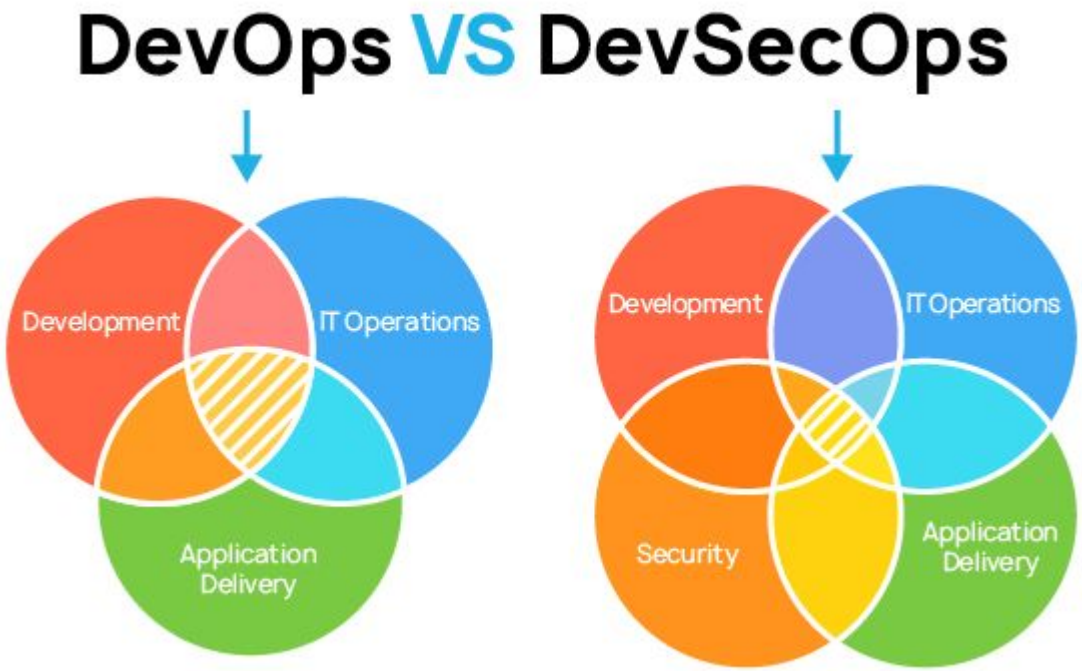
DevSecOps?

DevSecOps?

Team

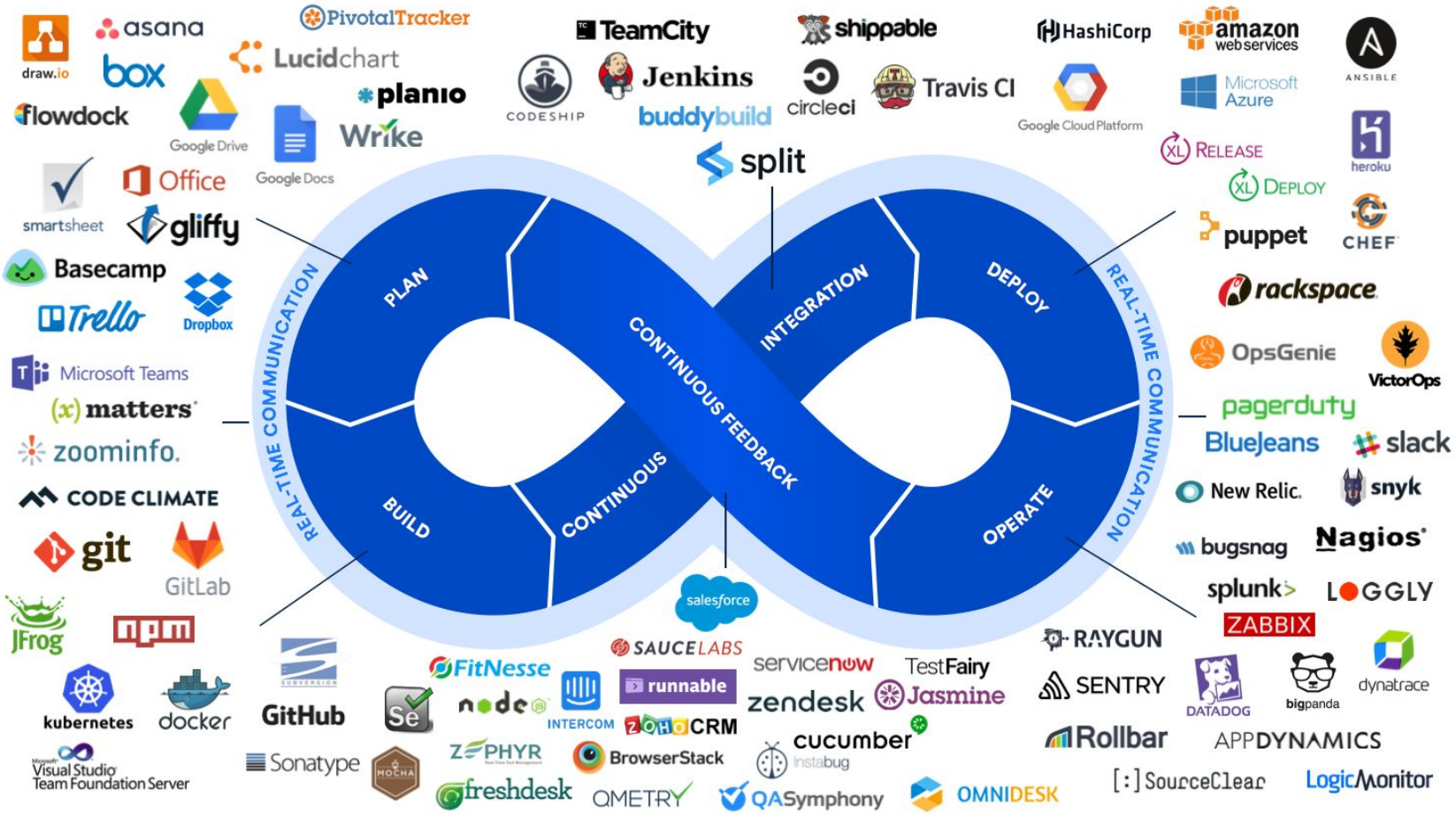
Development+Security+Operations

(DevSecOps)



Reference Pictures:

- What is the difference between DevOps and DevSecOps?



Reference Pictures:

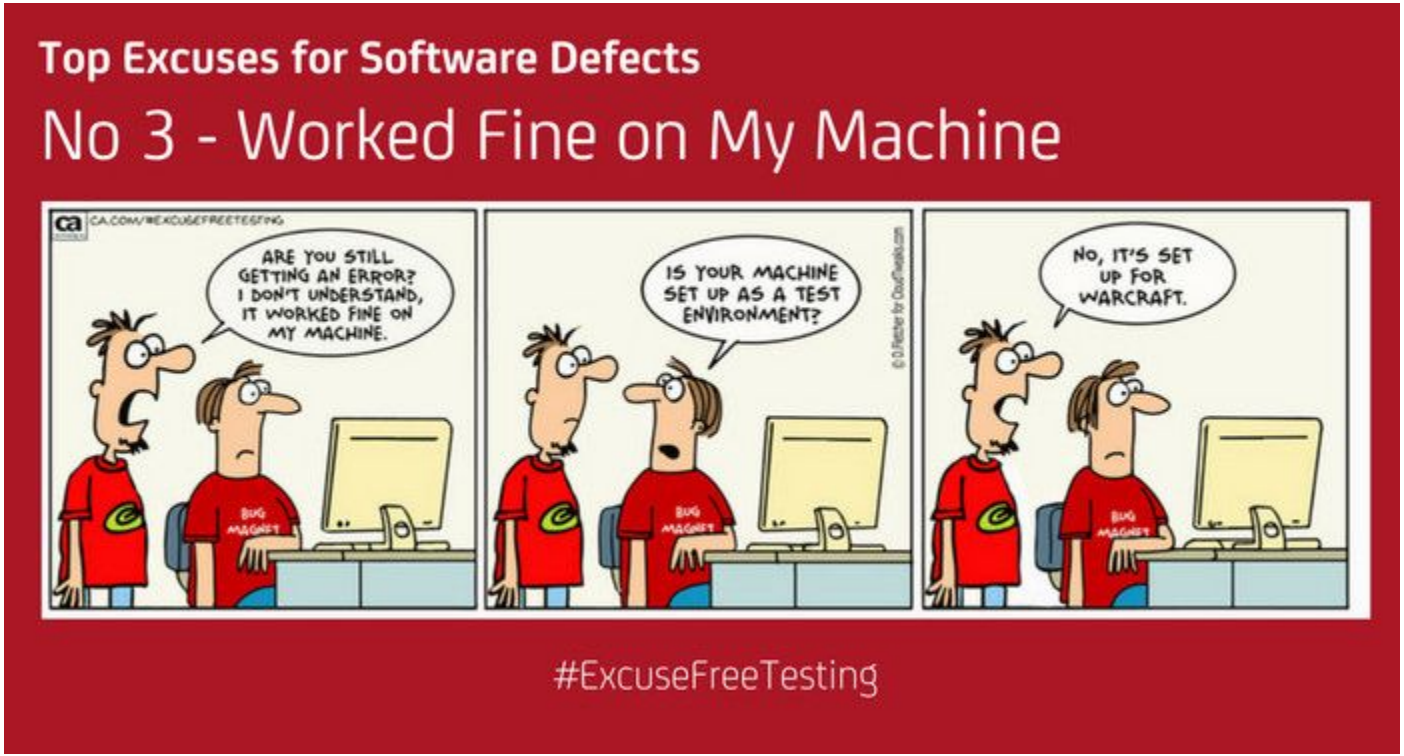
- DevOps Tools of the Trade

Jumpbox®

Get to Know Docker

Why & What Docker

- Solve it’s work on my machine.
- Ability multiple project with different version of application.
- Easy to ship.
- DevOps Automation (CI/CD)
- Docker registry
- Multiple platform
- Microservice
- Open Source
- Node cluster
- Freedom of choice (Docker registry and your own image)
- Cloud provider support (Kubernetes, Google Cloud Run, AWS ECS)

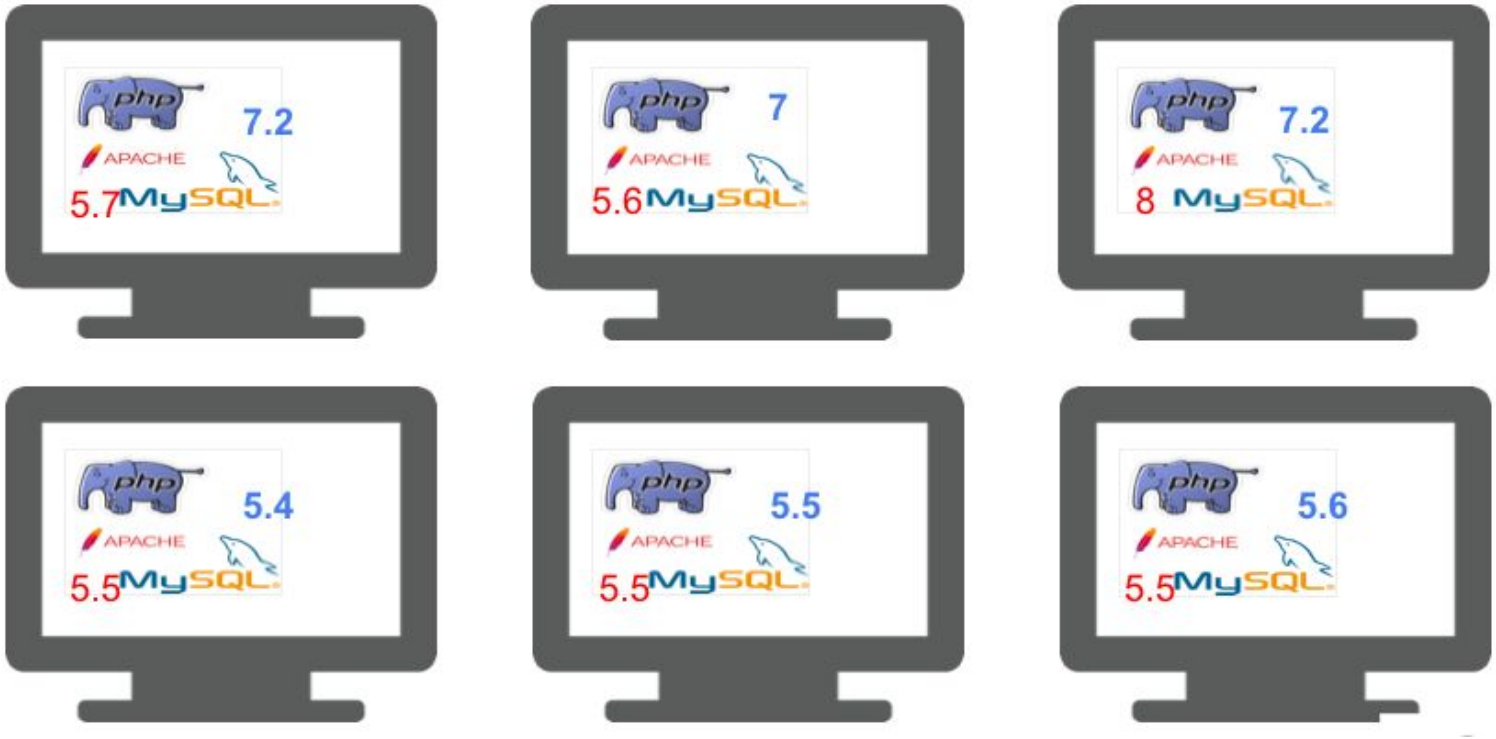


Reference Pictures:

- [Get Your Testing Game on with Service Virtualization]

Reference Pictures:

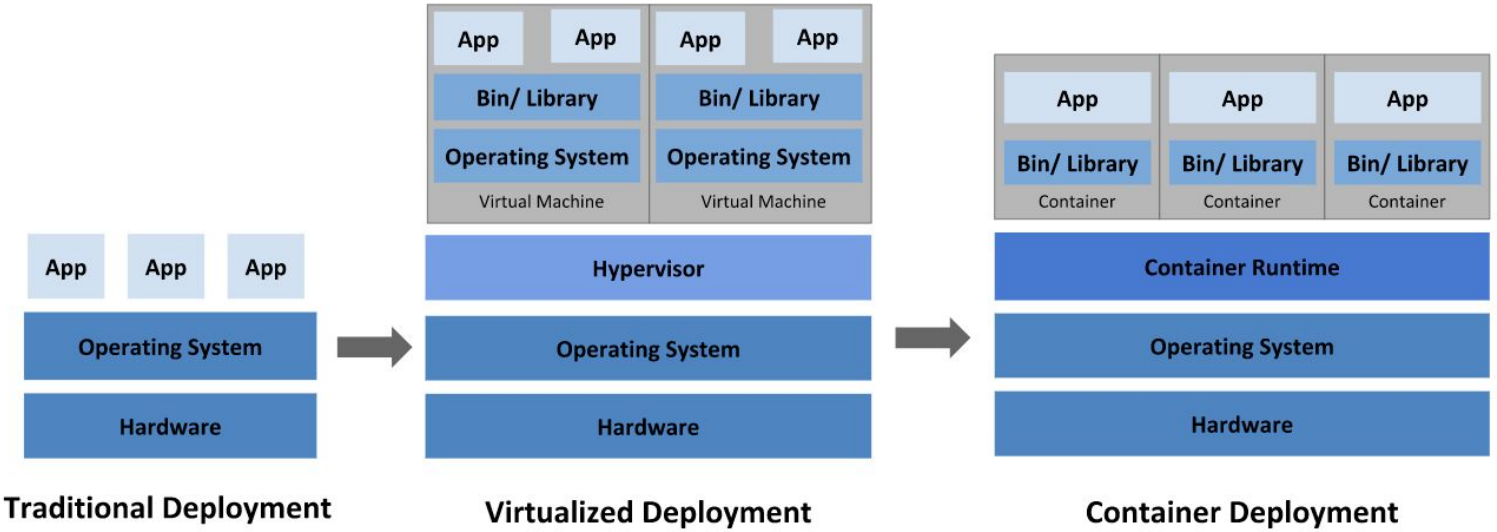
- What is a Traditional (LAMP, MAMP, WAMP, etc.,) Development?



Reference Pictures:

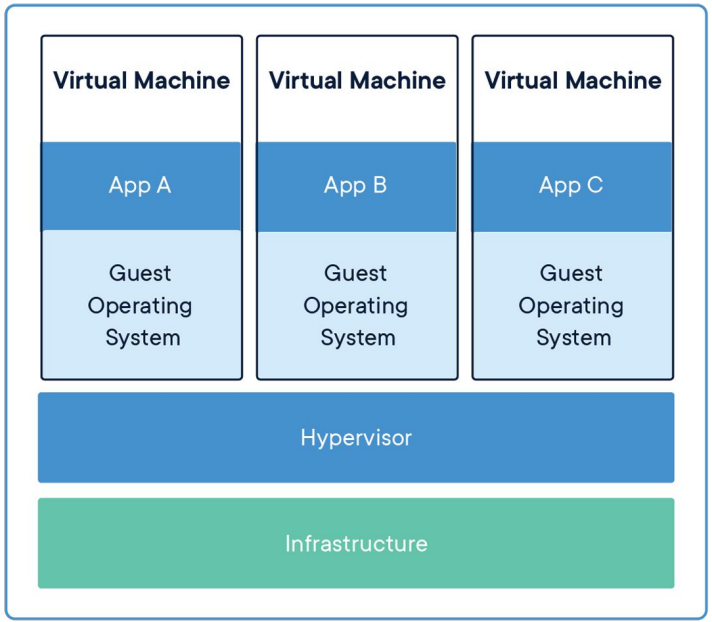
- [Laravel and Docker]

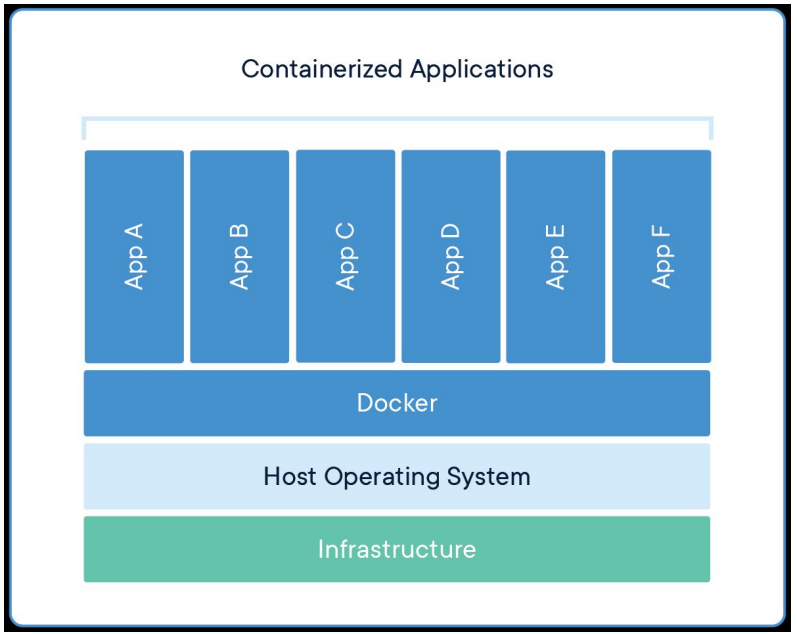
Evolution of deployment



Reference Pictures: [Overview Kubernetes]

Virtual Machine (VM) vs Container



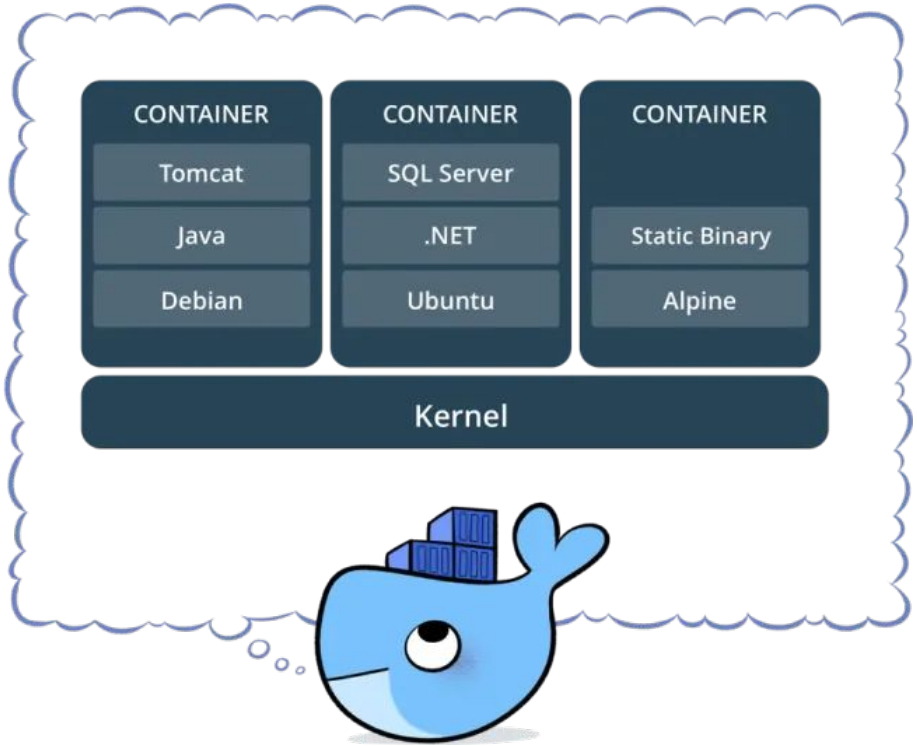


Reference Pictures:

- Use containers to Build, Share and Run your applications

Jumpbox®

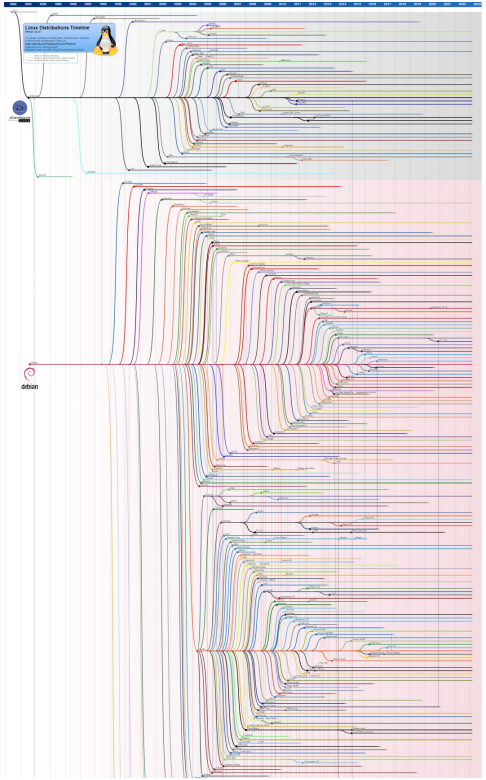
Example



Reference Pictures:

- What is Docker?

List of Linux distributions



Reference Pictures:

- File: Linux Distribution Timeline

	CONTAINER BENEFITS	VIRTUAL MACHINE BENEFITS
Consistent Runtime Environment	✓	✓
Application Sandboxing	✓	✓
Small Size on Disk	✓	
Low Overhead	✓	

Reference Pictures:

- Containers

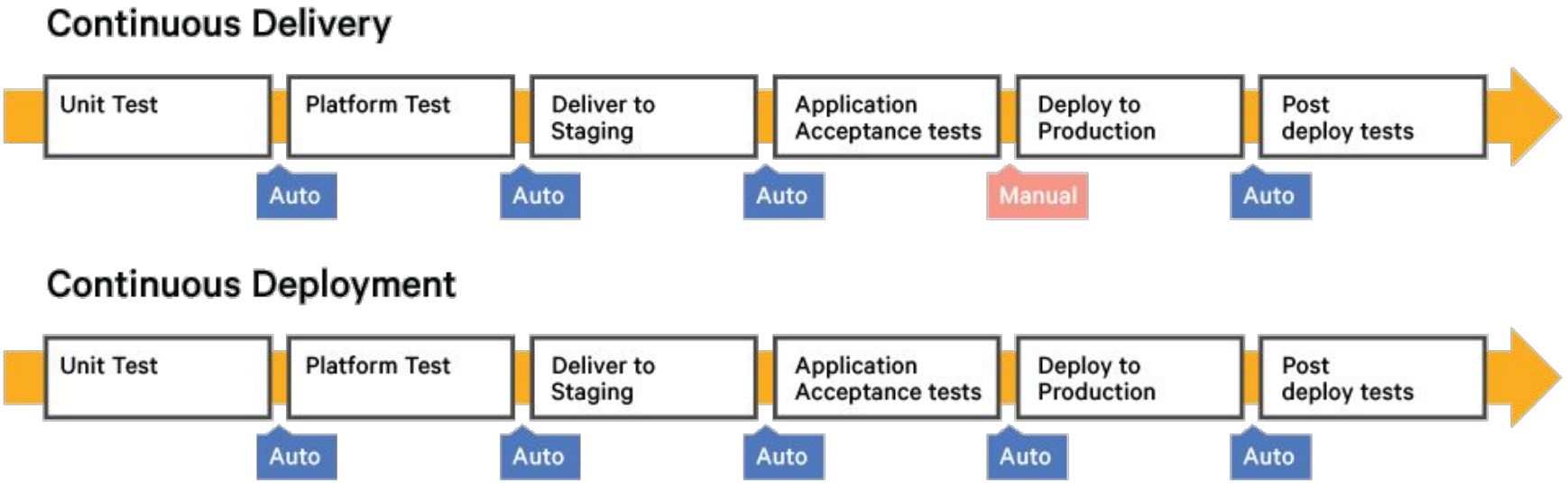
Containerization

- Flexible
- Lightweight
- Interchangeable
- Portable
- Scalable
- Stackable

Reference Pictures:

- File: Linux Distribution Timeline

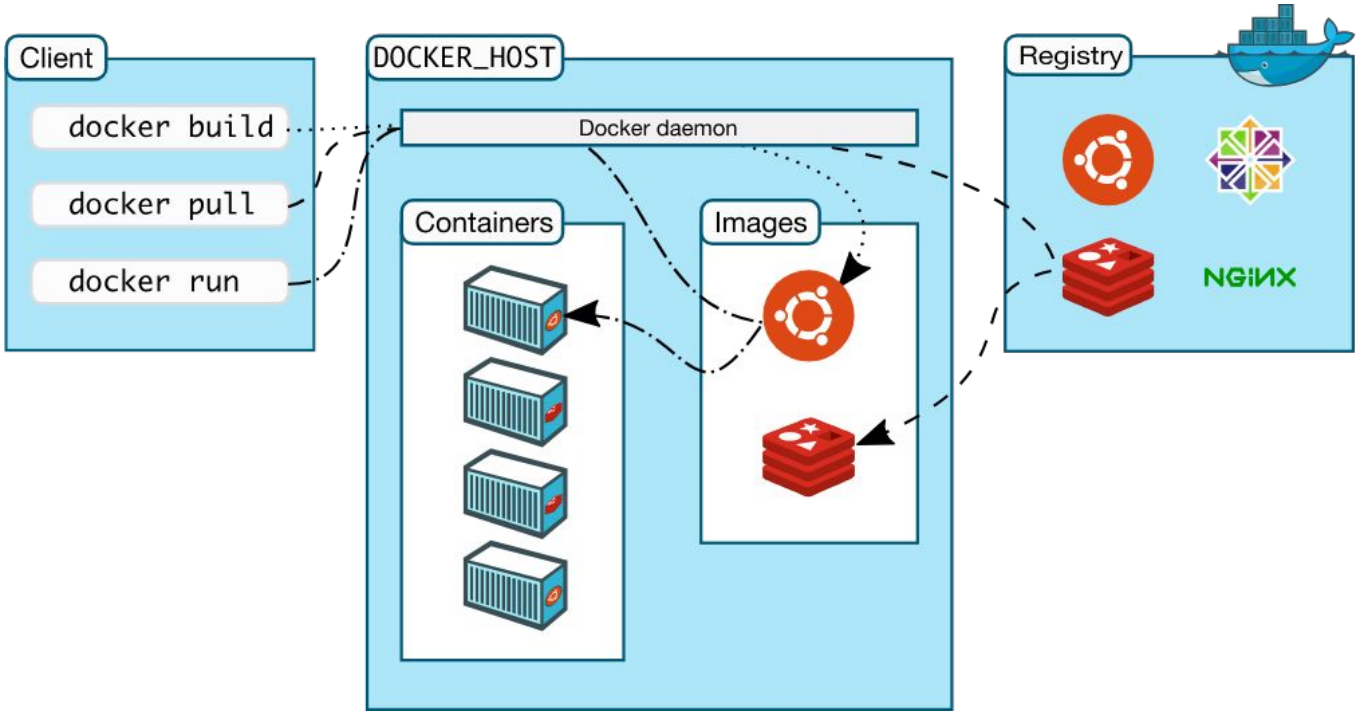
CD



Reference Pictures:

- Continuous Delivery vs. Continuous Deployment: An Overview

Docker Architecture: A Complete Docker Introduction



Reference Pictures:

- Docker Architecture: A Complete Docker Introduction

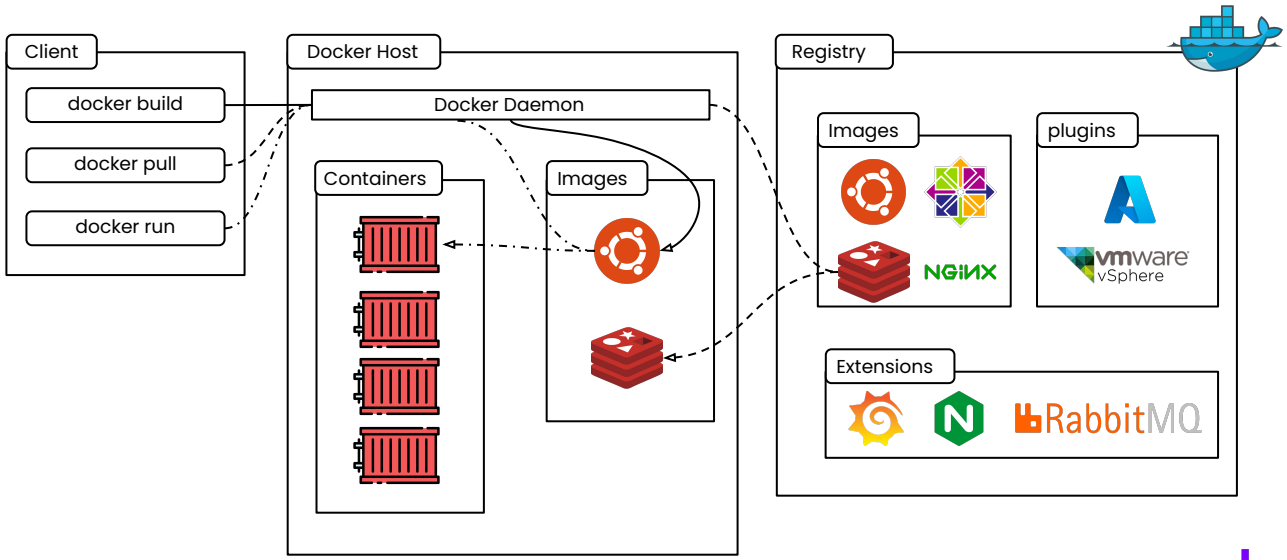
Architecture Overview

- Docker Architecture -

Jumpbox®

Docker Architecture (Cont.)

- Docker architecture is Client/Server.
- The docker Client communicates with the Server process to build, run and publish containers.
- The Client and Server can run on the same host, or they can communicate remotely across a network.



Docker Components

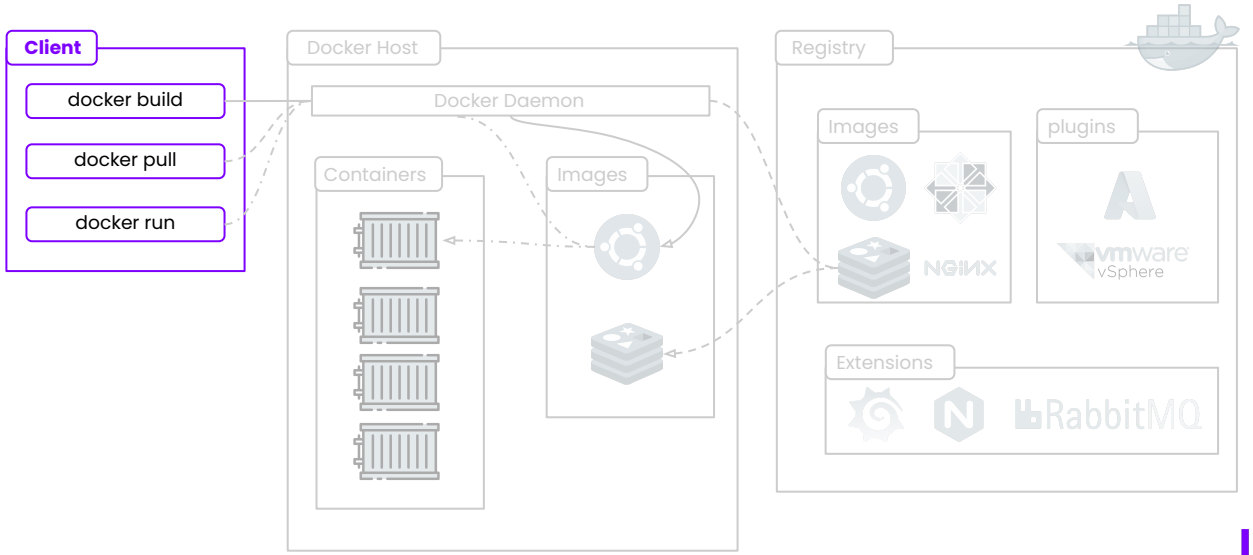
- Docker Architecture -

Jumpbox®

Docker Components (Cont.)

Client:

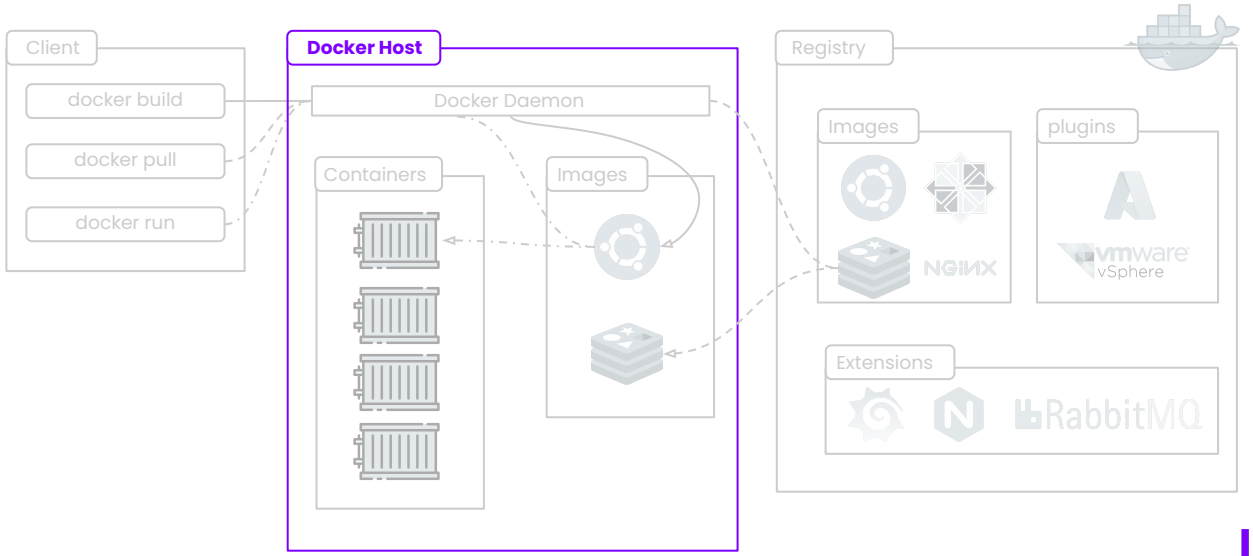
- The Docker client (docker / docker CLI) is the primary way that many Docker users interact with Docker. When you use commands such as docker run



Docker Components (Cont.)

Docker Host:

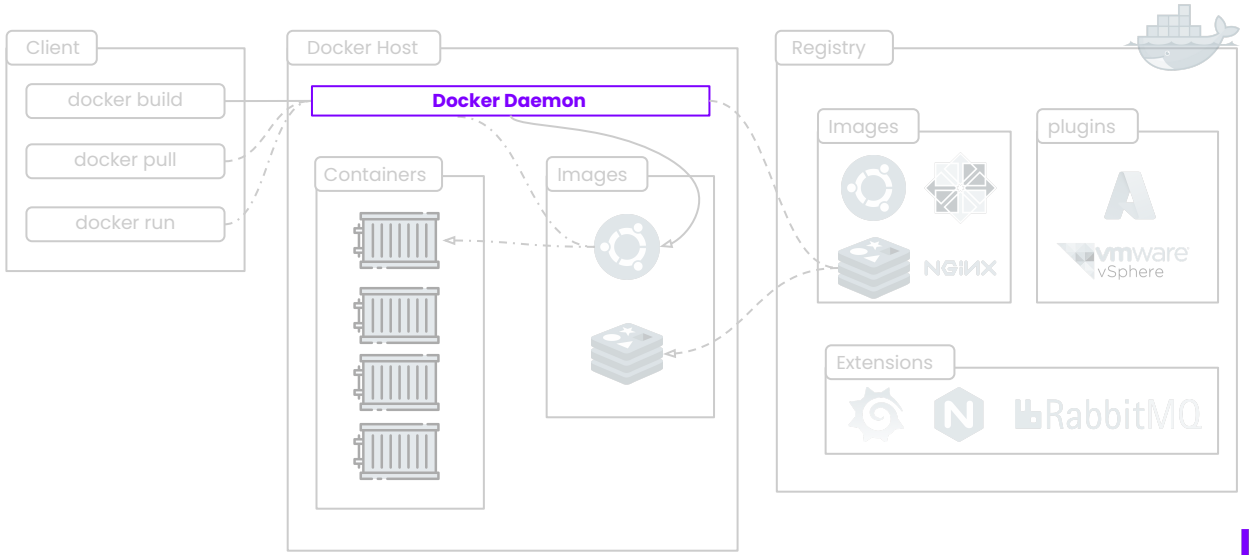
- Docker Host is the computer or server that installed Docker Runtime Engine.



Docker Components (Cont.)

Docker Daemon:

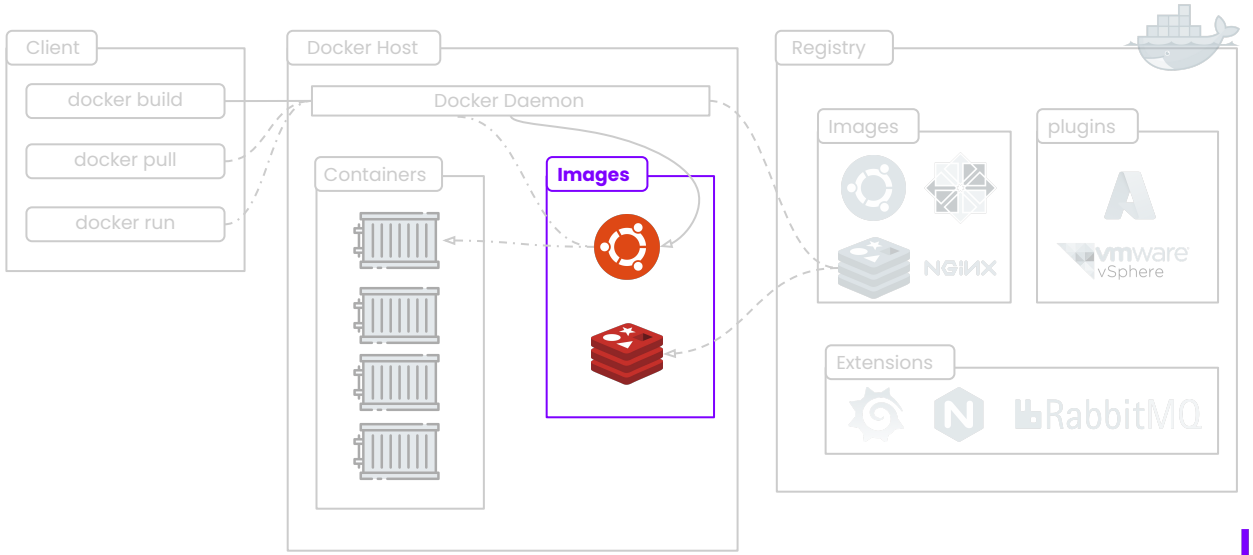
- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.



Docker Components (Cont.)

Images:

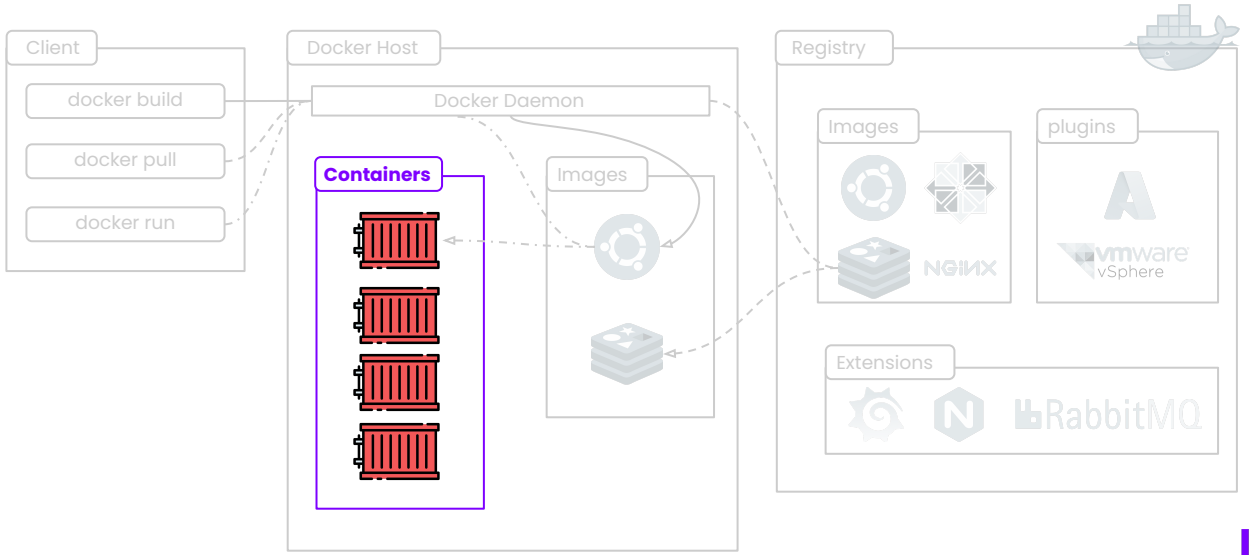
- An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.



Docker Components (Cont.)

Containers:

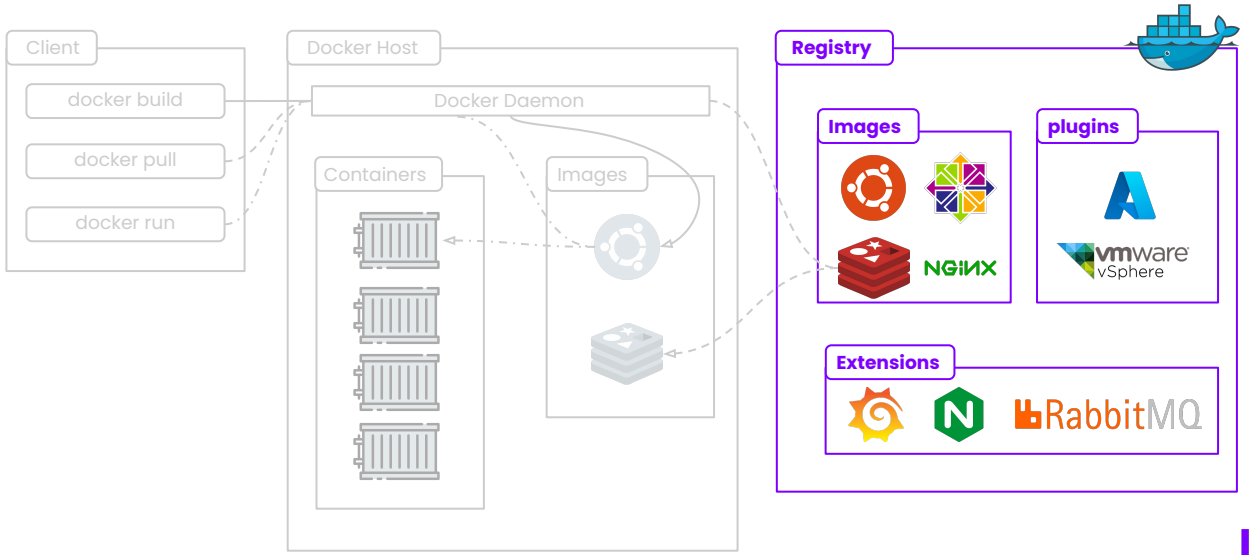
- A container is a runnable instance of an image.
- You can create, start, stop, move, or delete a container using the Docker API or CLI.



Docker Components (Cont.)

Registry:

- A Docker registry stores Docker images.
- Docker Hub is a Cloud registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.



Container Registry

- https://hub.docker.com/
- https://cloud.google.com/artifact-registry
- https://aws.amazon.com/ecr/
- https://quay.io/

Activity Time

- With Paper Base -

Output what you learned in **Practical Action**

Hands-on Workshop: Running Docker Container

- Install Docker Desktop
- Running your 1st Docker container

Get Code Snippet

SCAN ME

Jumpbox®

Install Docker via command-line

macOS

- Docker Desktop for Mac

Windows

- Docker Desktop for Windows

Linux

- Docker Engine

Install Docker via command-line (Cont.)

macOs with Homebrew

- brew install --cask docker

Windows with Winget

- winget install -e --id Docker.DockerDesktop

Install WSL 2

Window Only

- [บันทึก]การติดตั้ง WSL 2, Docker Desktop บน Windows 10 Home

Linux old version installation

Docker

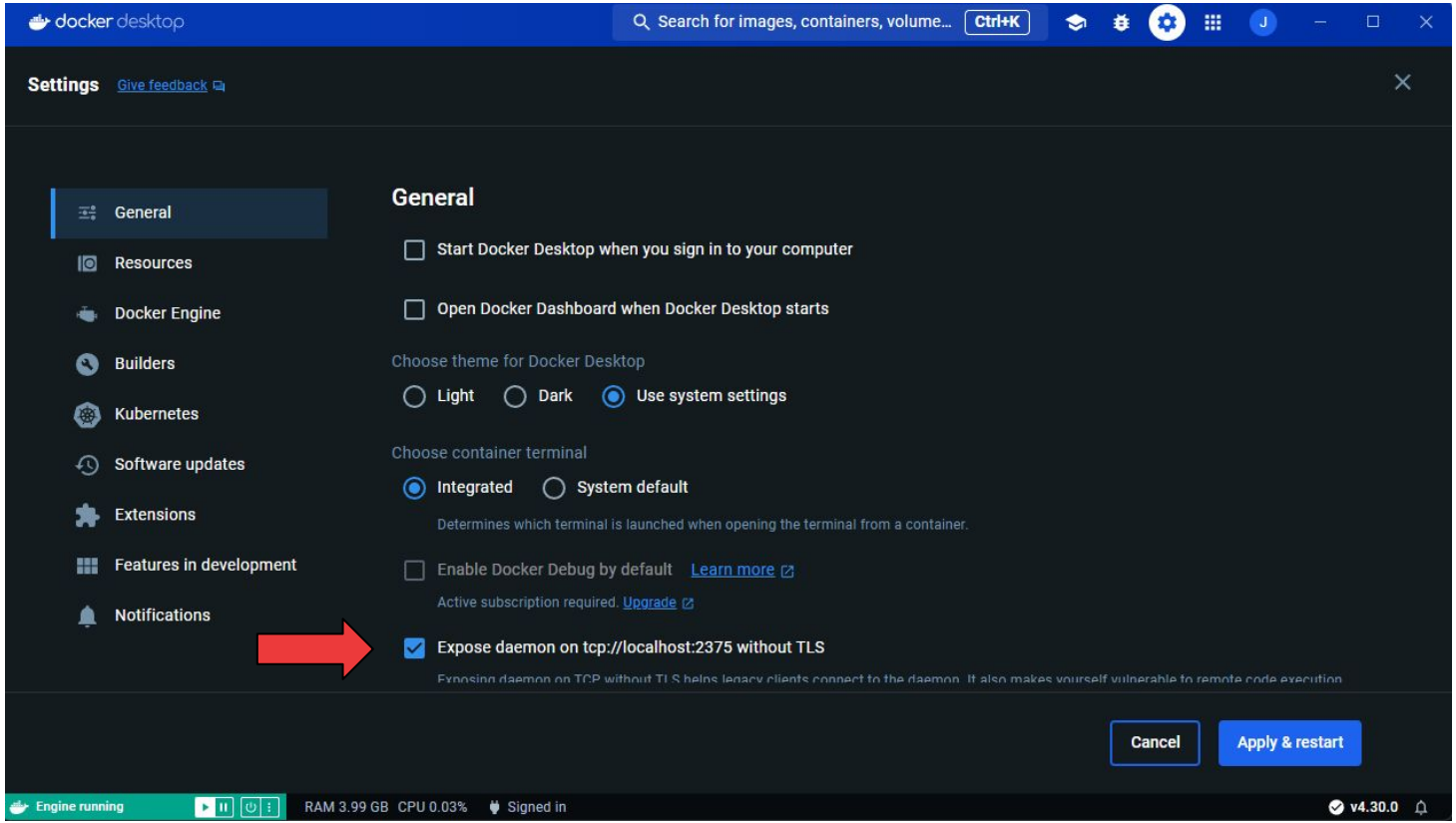
- curl -fsSL https://get.docker.com -o get-docker.sh
- sudo sh ./get-docker.sh
- sudo usermod -aG docker $USER

Docker Compose

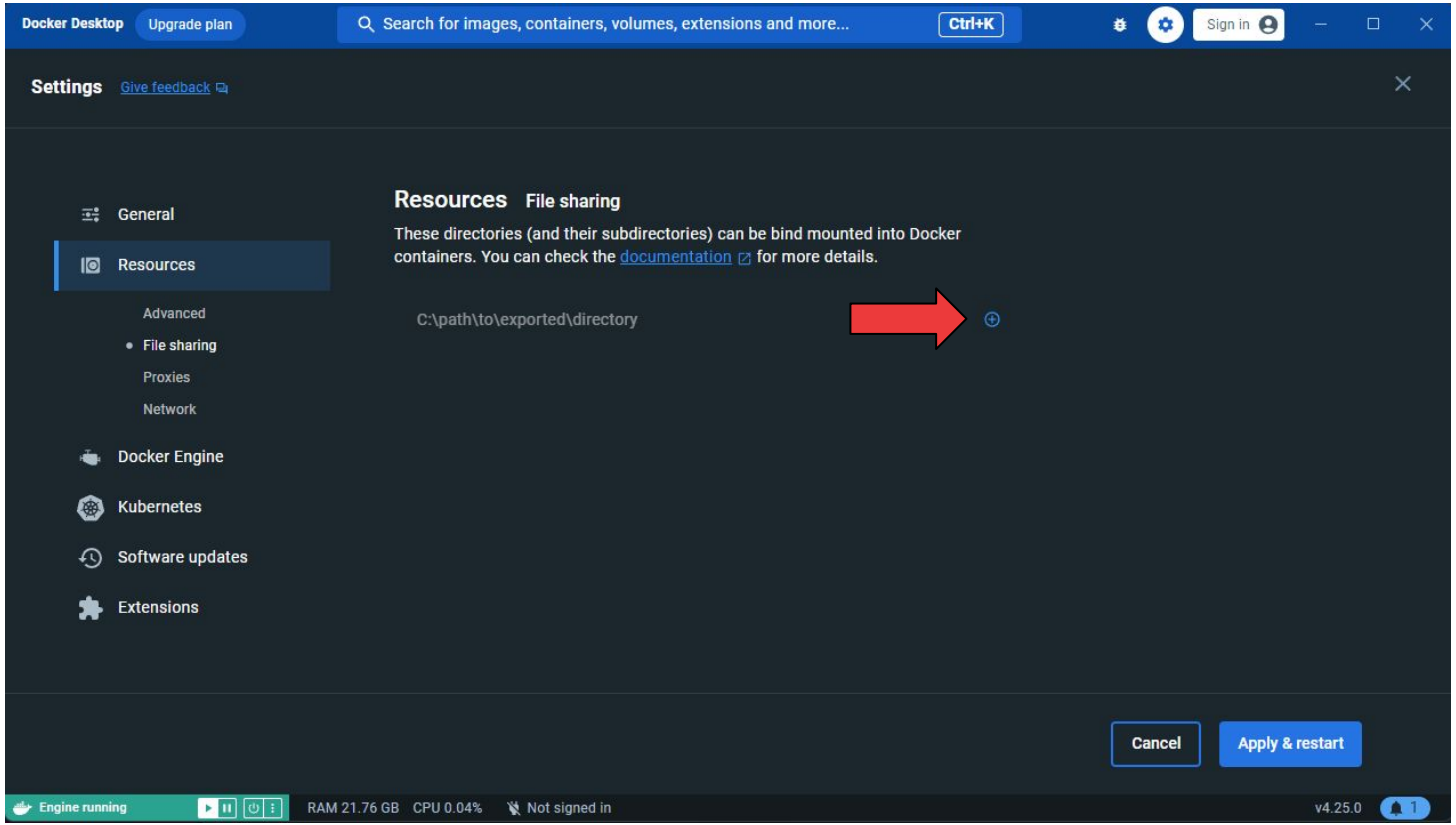
- sudo curl -L "https://github.com/docker/compose/releases/download/2.27.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
- sudo chmod +x /usr/local/bin/docker-compose

Code Snippet: https://bit.ly/3kTJKzn

Setting for Windows without WSL2

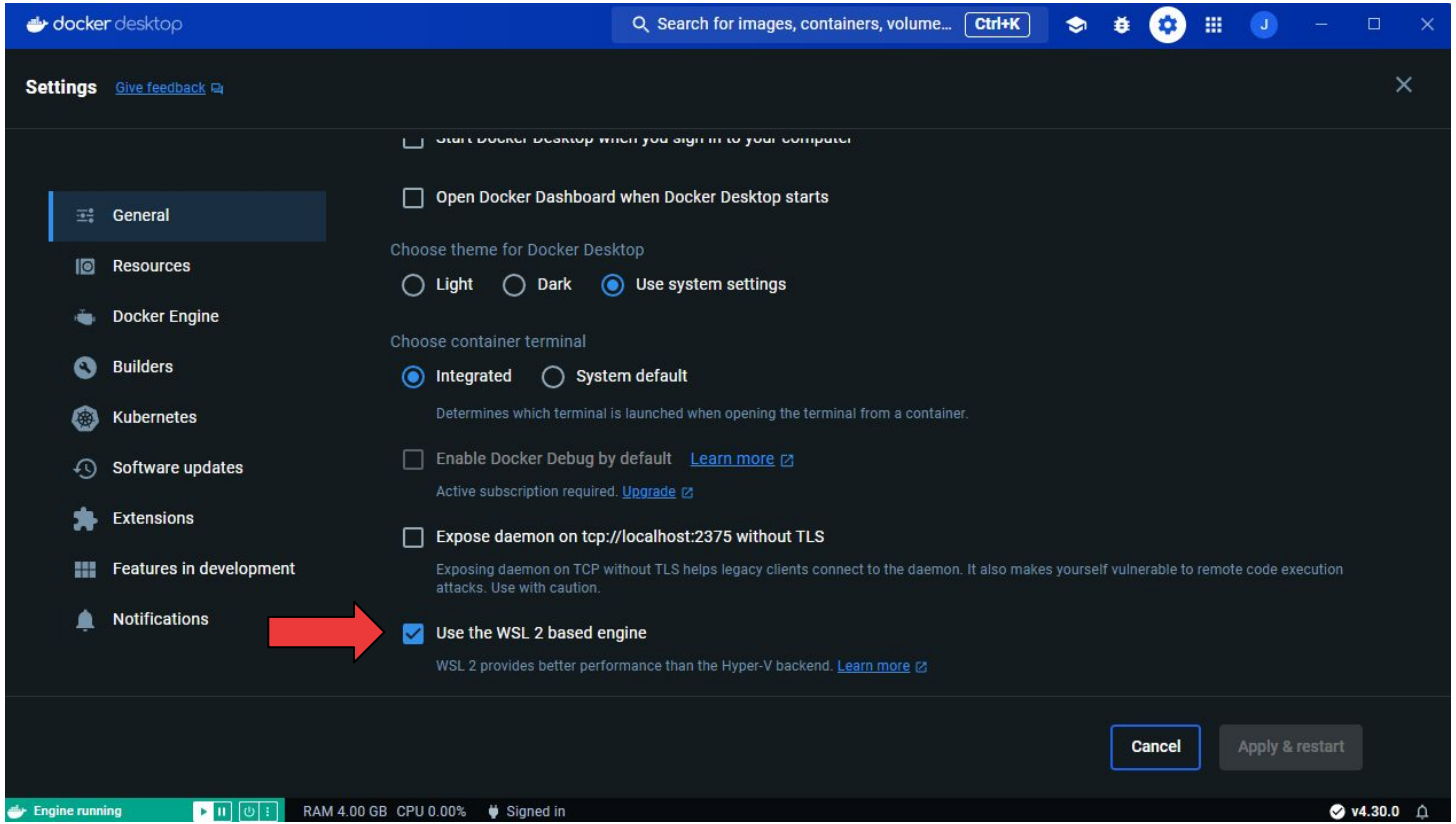


Setting for Windows without WSL2 (Cont.)



Jumpbox®

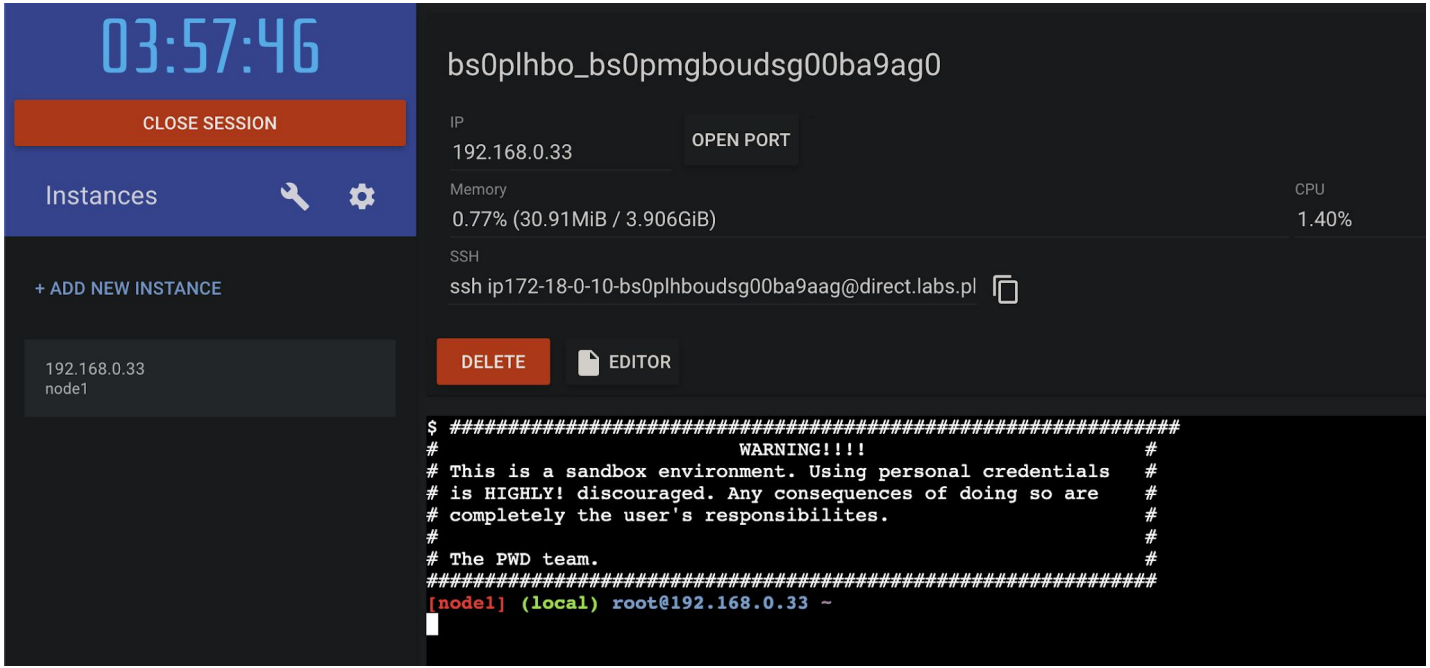
Setting for Windows with WSL2



Jumpbox®

Docker Playground

- https://labs.play-wiath-docker.com



Jumpbox®

Already setup

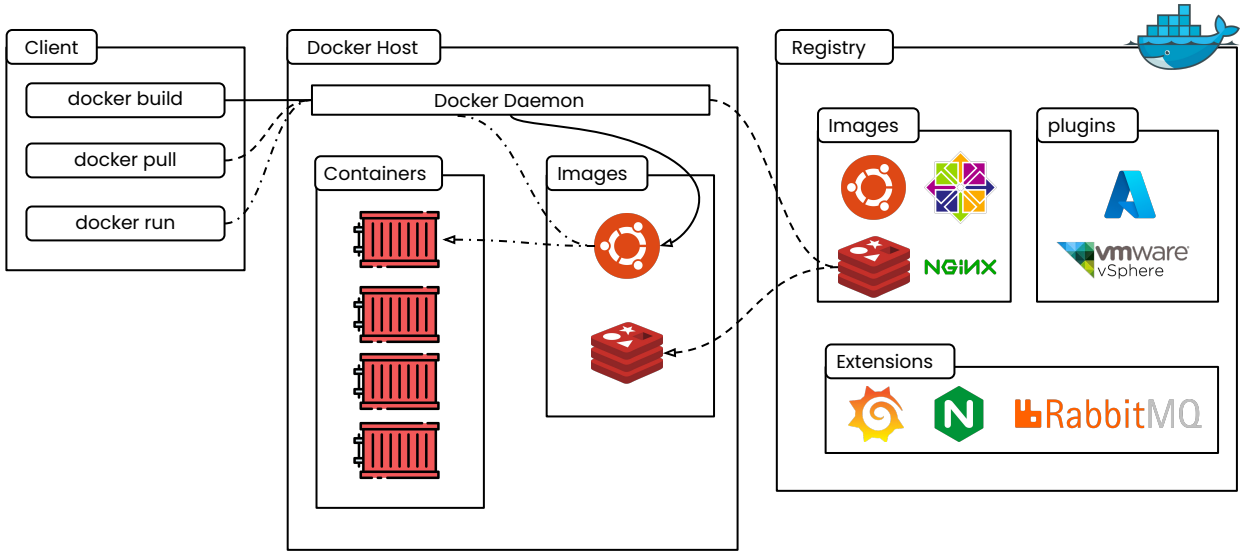
Lunch break

Get GitHub Repo

SCAN ME

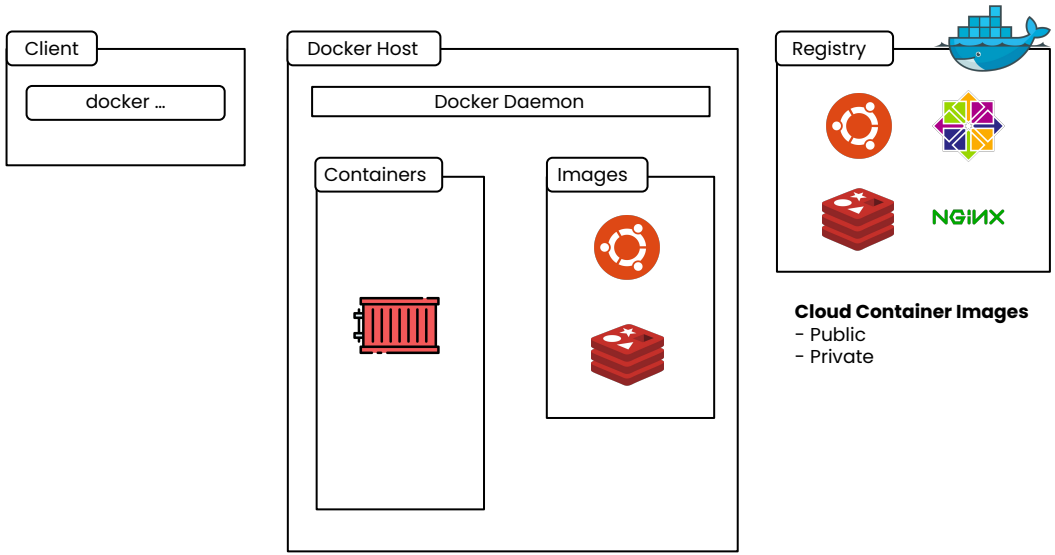
Jumpbox®

Docker Architecture (Cont.)



Make it simple

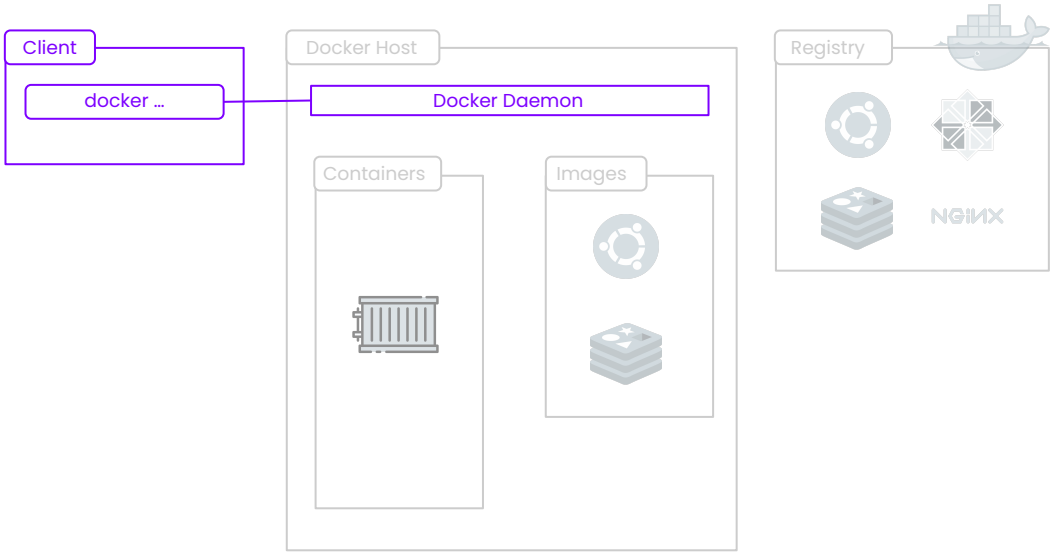
Docker Simple Architecture (Cont.)



Let’s focus on Client / CLI

3. Docker Management (Cont.)

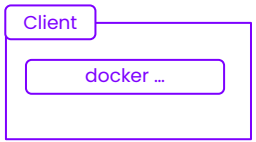
Users use CLI for Docker management



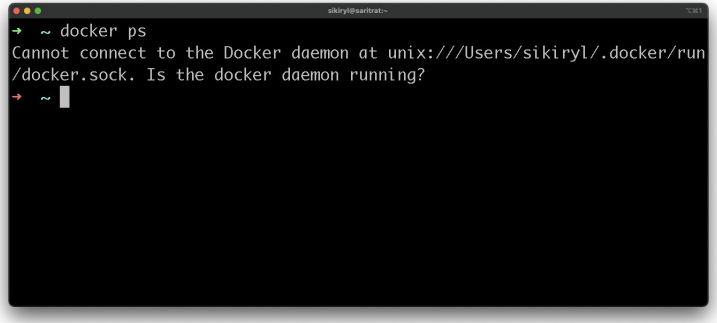
Let's look at the difference between logical and real world

Connection to Docker Daemon(Server) (Cont.)

Logical

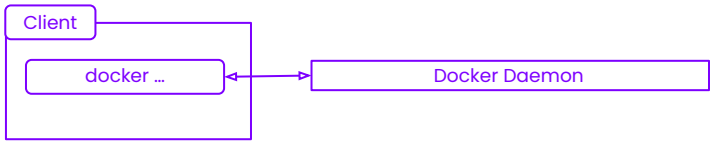


Real World

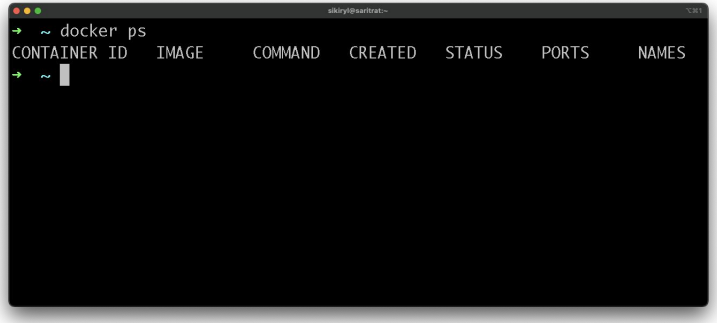


Connection to Docker Daemon(Server) (Cont.)

Logical



Real World



Running your
1st Docker container

How to run containers workshop

1. Change directory to simple-demo
2. Run hello-world container
3. Run simple website with nginx container

Utilities Commands Overview

1. Show all running containers
2. Show all containers
3. Logs website with nginx container
4. Open website
5. Open another shell
6. Inspect container
7. Shell to website with nginx container
8. Exit

$ docker run

- Container Instance Managements -

Jumpbox®

$ docker run (Cont.)

$ docker run (Cont.)

Syntax:

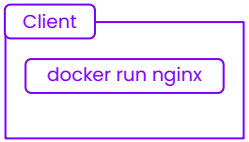
```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

$ docker run (Cont.)

Syntax:

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Logical:

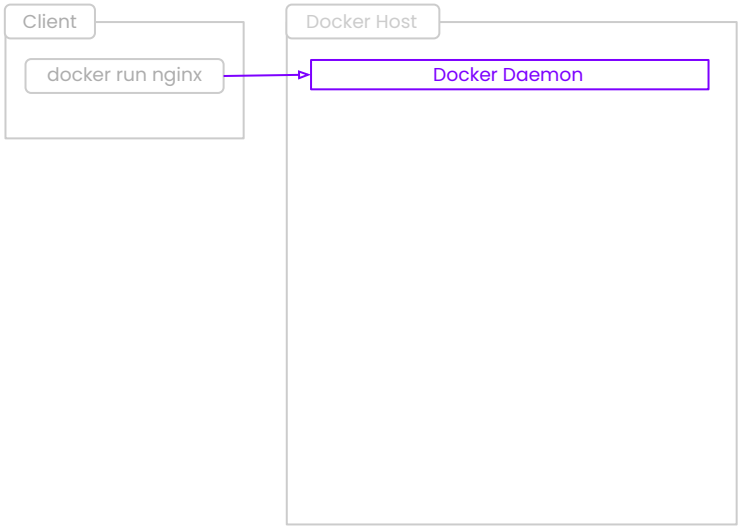


$ docker run (Cont.)

Syntax:

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Logical:

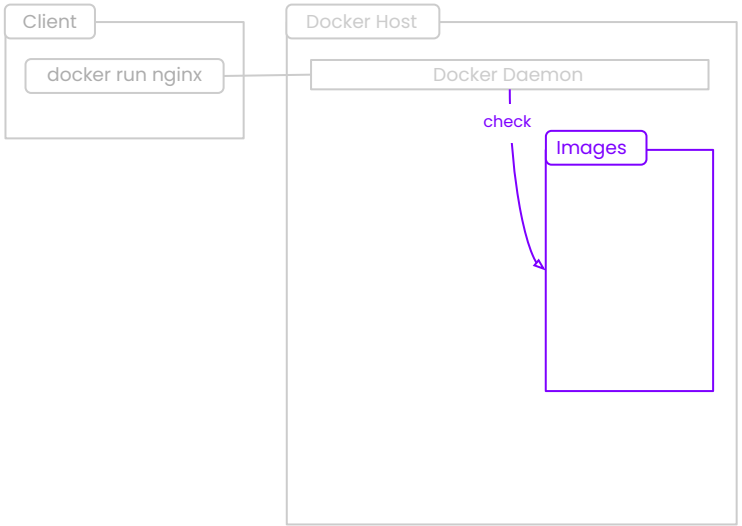


$ docker run (Cont.)

Syntax:

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Logical:

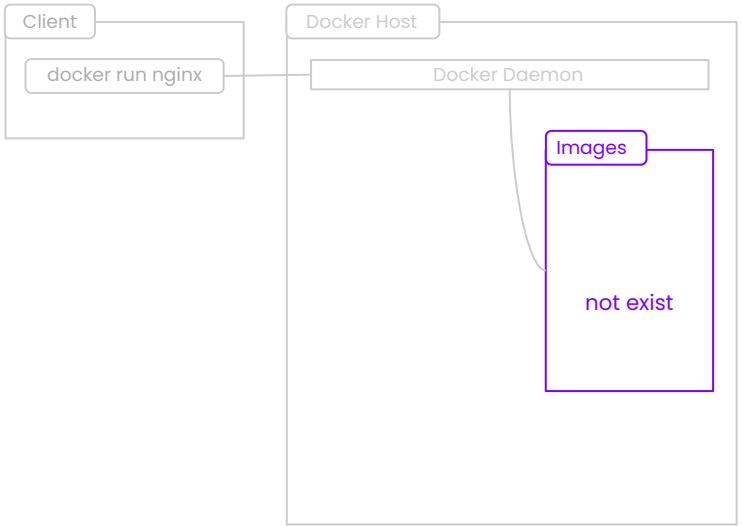


$ docker run (Cont.)

Syntax:

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Logical:

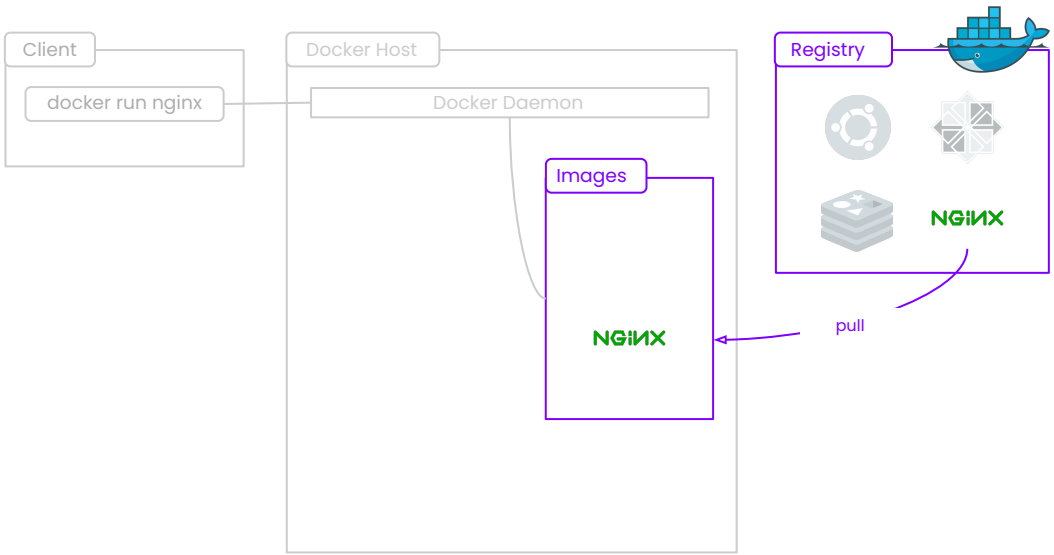


$ docker run (Cont.)

Syntax:

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Logical:

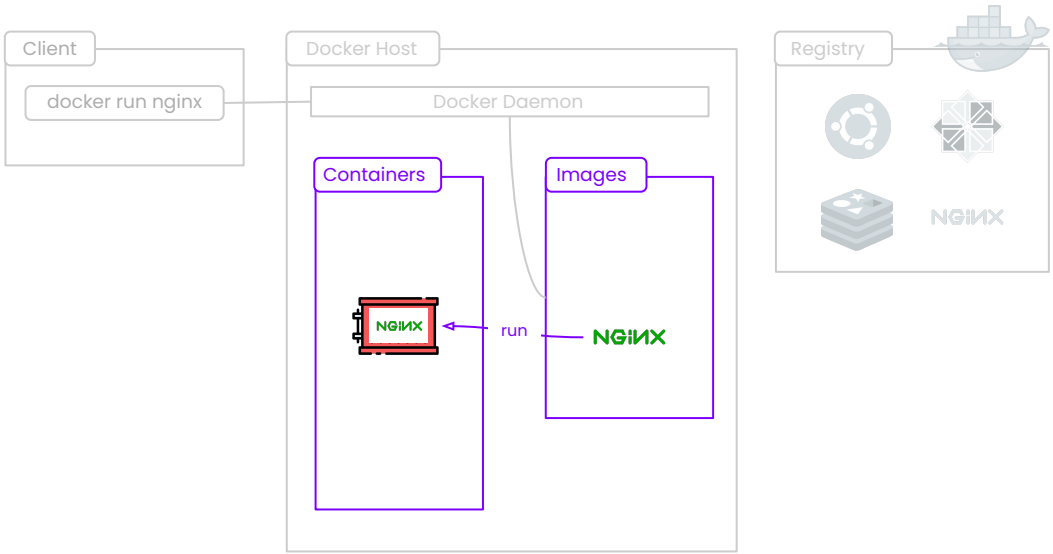


$ docker run (Cont.)

Syntax:

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Logical:

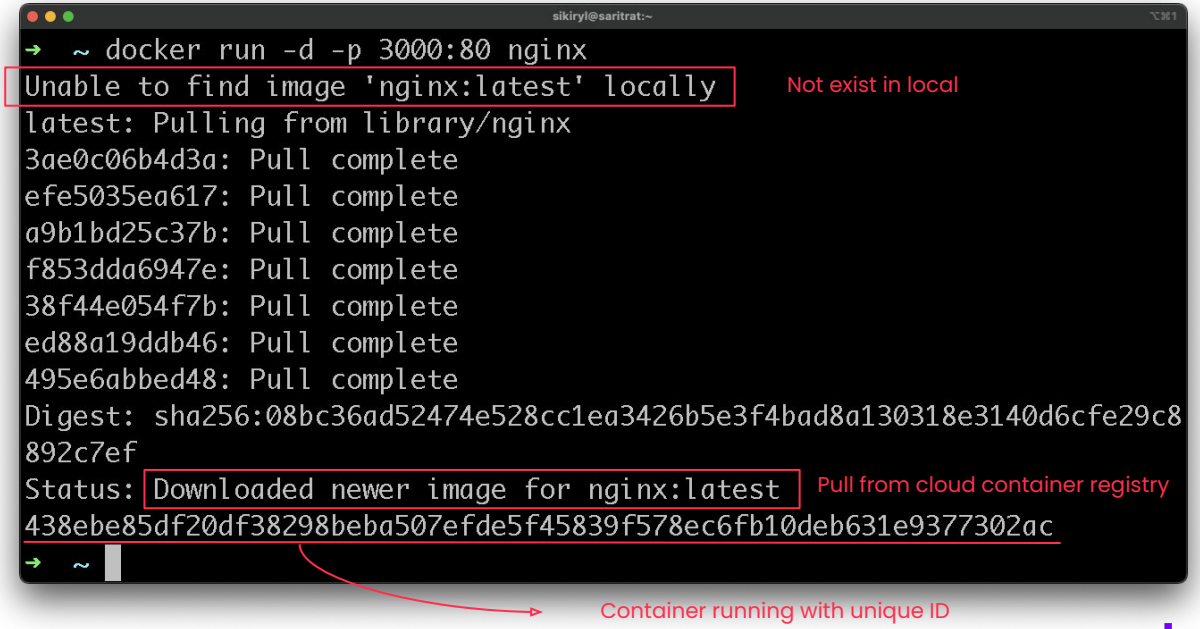


$ docker run (Cont.)

Syntax:

```
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Real world:



$ docker run (Cont.)

command:

```
$ docker run -d -p 3000:80 nginx
```

options:

- -d, --detach
  - Run container in background and print container ID
- -p, --publish list
  - Publish a container's port(s) to the host

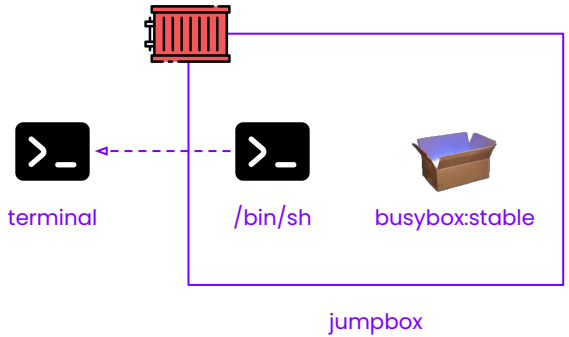
$ docker run (Cont.)

command:

```
$ docker run --name jumpbox -it busybox:1.29 /bin/sh
```

options:

- --name string
  - Assign a name to the container
- -i, --interactive and -t, --tty (-it)
  - -t, --tty: Allocate a pseudo-TTY
  - -i, --interactive: Keep STDIN open even if not attached



Popular $ docker run command options

- -d or --detach
- -p or --publish
- -v or --volume
- -e or --env
- --rm
- --link

Demo

Exercise

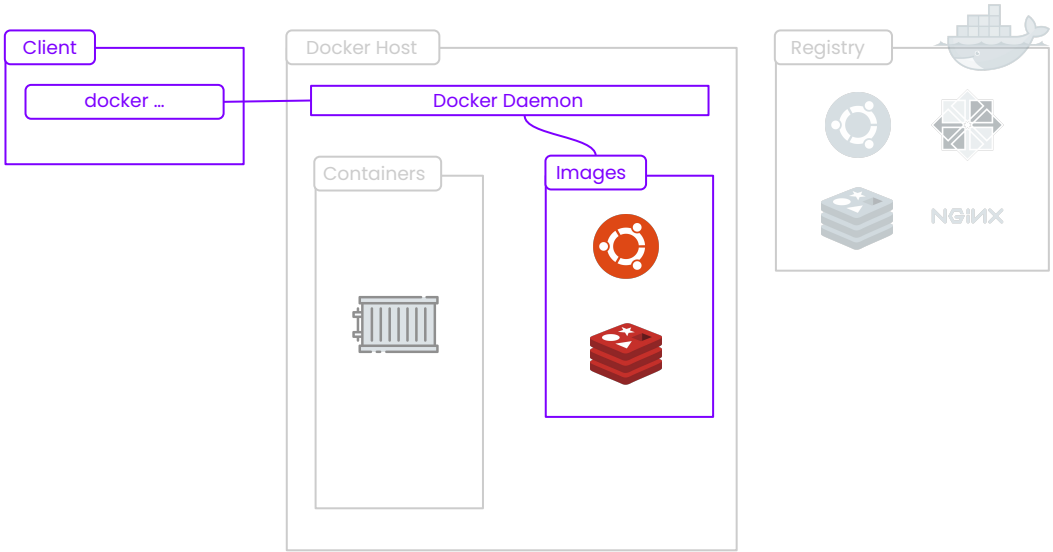
https://kahoot.it/

Game PIN:

Container image

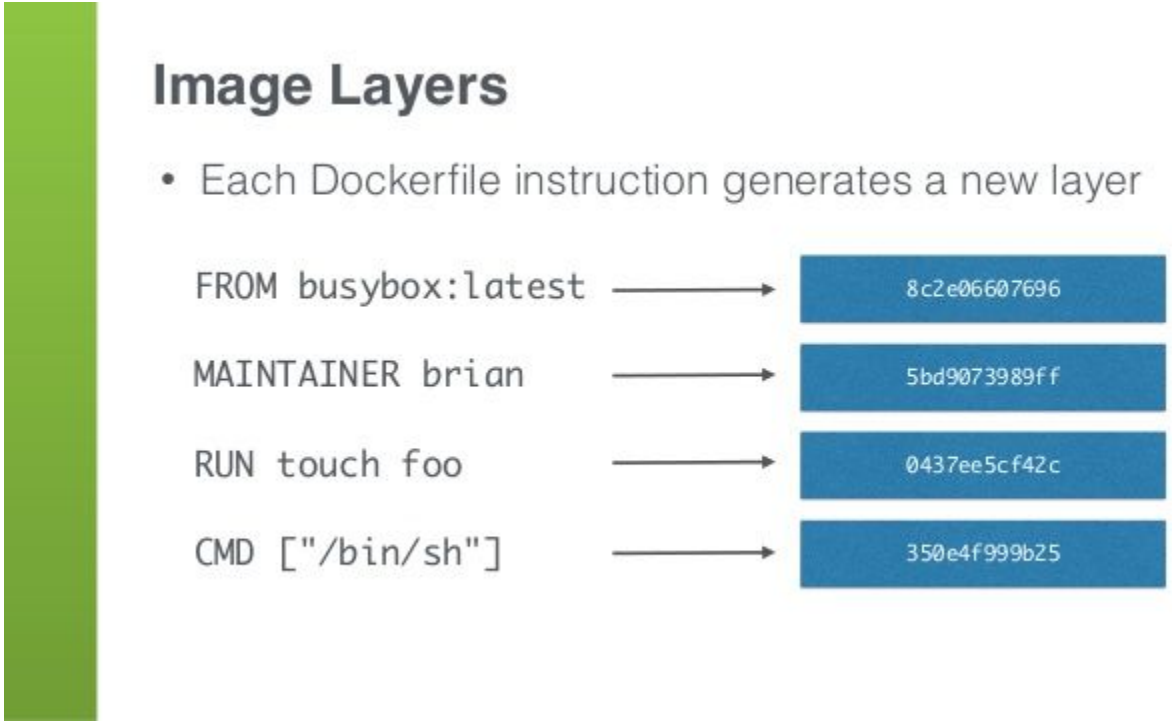
Container Images (Cont.)

Users use CLI to access Container Images



Dockerfile

Layers Theory



Reference Pictures:

- Optimizing Docker Images

Dockerfile

Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.

Dockerfile

```
FROM python:3.11.5
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
EXPOSE 5000
ENTRYPOINT ["flask", "run"]
CMD ["--host=0.0.0.0"]
```

Reference:

- Dockerfile

Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.

Dockerfile

```
FROM python:3.11.5 ──▶ hub.docker.com
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
EXPOSE 5000
ENTRYPOINT ["flask", "run"]
CMD ["--host=0.0.0.0"]
```

Reference:

- Dockerfile

Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.

Dockerfile

```
FROM python:3.11.5      ⟶ hub.docker.com
WORKDIR /app            ⟶ create app directory and open it
COPY . .
RUN pip install -r requirements.txt
EXPOSE 5000
ENTRYPOINT ["flask", "run"]
CMD ["--host=0.0.0.0"]
```

Reference:

- Dockerfile

Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.

Dockerfile

```
FROM python:3.11.5      ──▸ hub.docker.com
WORKDIR /app            ──▸ create app directory and open it
COPY . .                ──▸ copy directory to container
RUN pip install -r requirements.txt
EXPOSE 5000
ENTRYPOINT ["flask", "run"]
CMD ["--host=0.0.0.0"]
```

Reference:

- Dockerfile

Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.

Dockerfile

```
FROM python:3.11.5          ──> hub.docker.com
WORKDIR /app                ──> create app directory and open it
COPY . .                    ──> copy directory to container
RUN pip install -r requirements.txt  ──> Execute shell command
EXPOSE 5000
ENTRYPOINT ["flask", "run"]
CMD ["--host=0.0.0.0"]
```

Reference:

- Dockerfile

Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.

Dockerfile

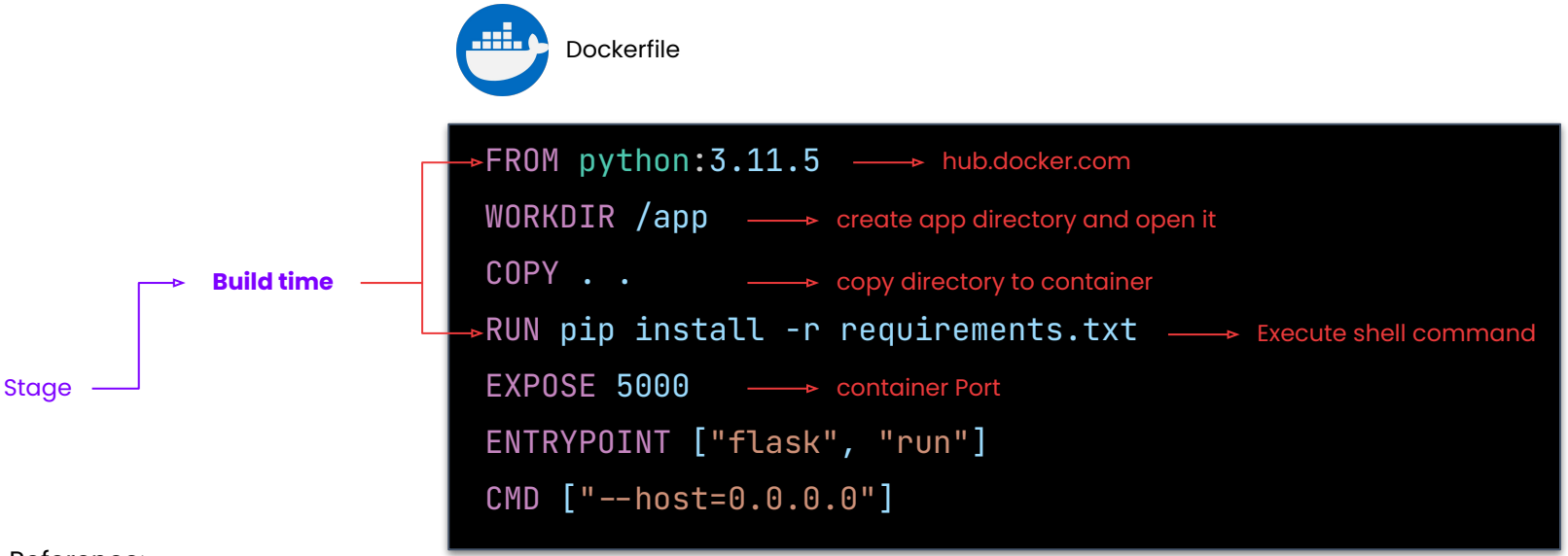
```
FROM python:3.11.5          ⟶ hub.docker.com
WORKDIR /app                ⟶ create app directory and open it
COPY . .                    ⟶ copy directory to container
RUN pip install -r requirements.txt  ⟶ Execute shell command
EXPOSE 5000                 ⟶ container Port
ENTRYPOINT ["flask", "run"]
CMD ["--host=0.0.0.0"]
```

Reference:

- Dockerfile

Dockerfile (Cont.)

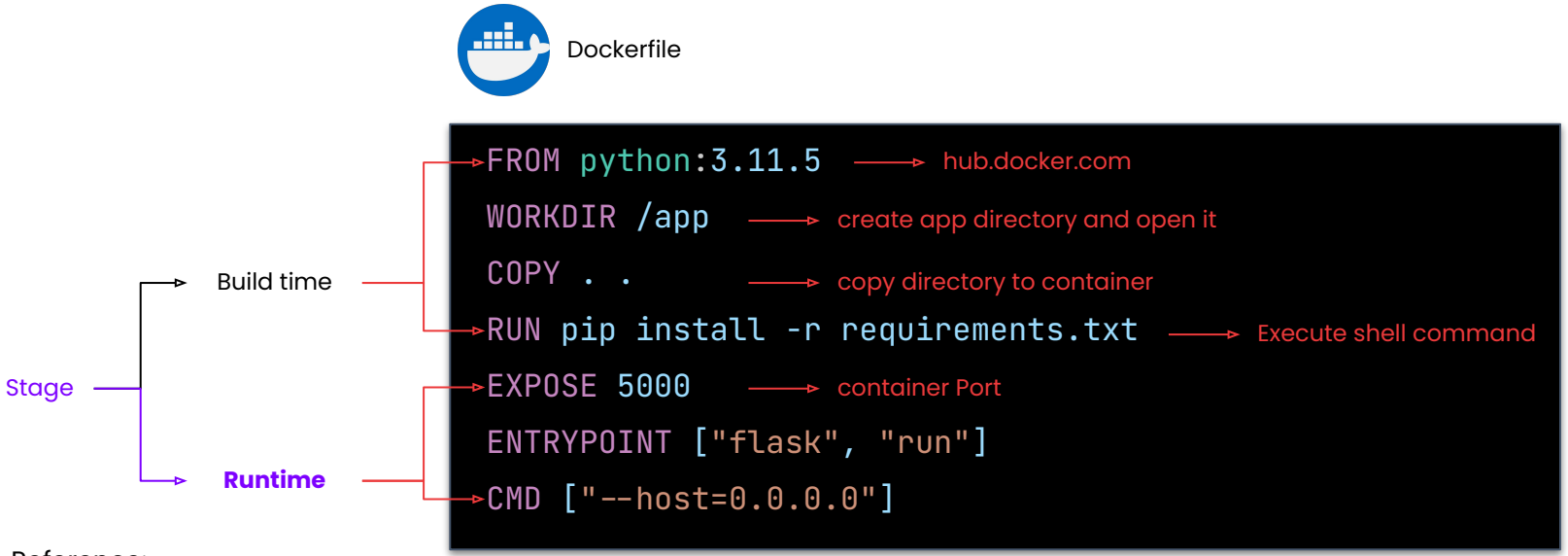
- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.



Reference:
- Dockerfile

Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.



Reference:

- Dockerfile

$ docker build

Reference:

- $ docker build

$ docker build (Cont.)

Syntax:

```
$ docker build [OPTIONS] PATH | URL | -
```

Reference:

- $ docker build

$ docker build (Cont.)

Syntax:

```
$ docker build [OPTIONS] PATH | URL | -
```

Ex 1:

```
$ docker build -t <container-image-name> <context>
```

Reference:

- $ docker build

$ docker build (Cont.)

Syntax:

```
$ docker build [OPTIONS] PATH | URL | -
```

Ex 1:

```
$ docker build -t <container-image-name> <context>
```

Ex 2:

```
$ docker build -t <container-image-name>:<tag> <context>
```

Reference:

- $ docker build

$ docker build (Cont.)

Syntax:

```
$ docker build [OPTIONS] PATH | URL | -
```

Ex 1:

```
$ docker build -t <container-image-name> <context>
```

Ex 2:

```
$ docker build -t <container-image-name>:<tag> <context>
```

Ex 3:

```
$ docker build -t <container-image-name>:<tag> --no-cache <context>
```

- The docker build command builds container images from a **Dockerfile** and a **"context"**.

Reference:

- $ docker build

$ docker build (Cont.)

Syntax:

```
$ docker build [OPTIONS] PATH | URL | -
```

Ex 1:

```
$ docker build -t <container-image-name> <context>
```

Ex 2:

```
$ docker build -t <container-image-name>:<tag> <context>
```

Ex 3:

```
$ docker build -t <container-image-name>:<tag> --no-cache <context>
```

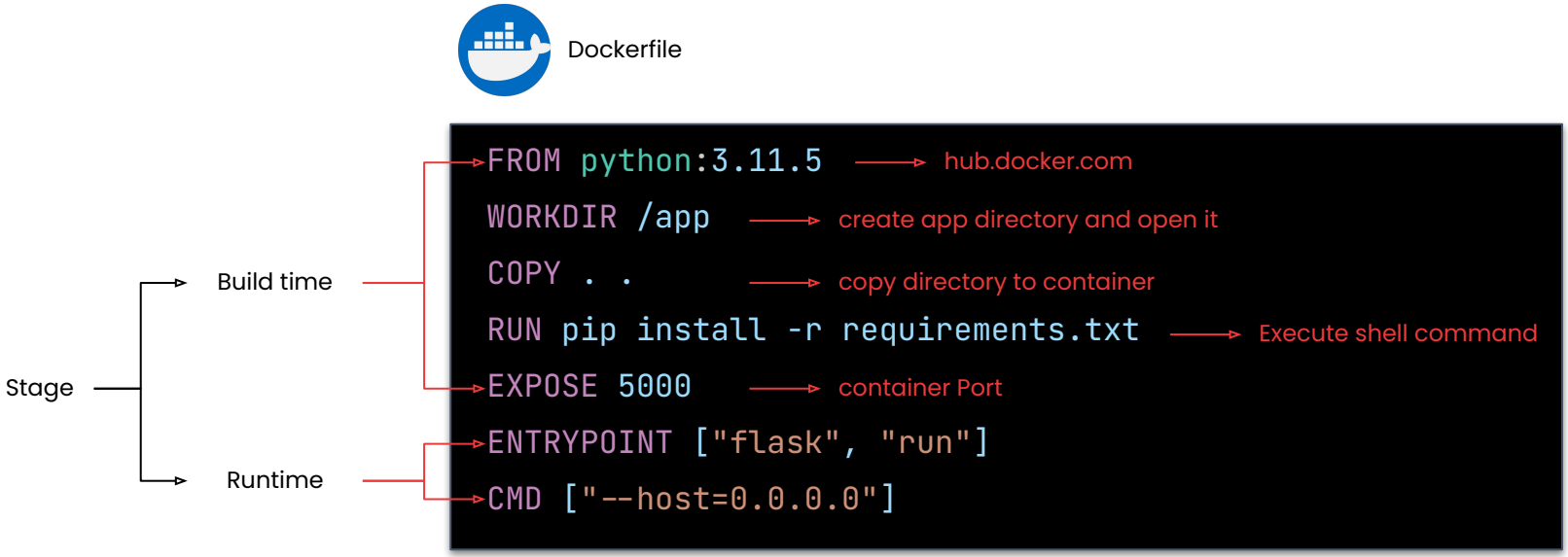
- The docker build command builds container images from a **Dockerfile** and a **"context"**.
- A build's context is the set of files located in the specified PATH or URL.

Reference:

- $ docker build

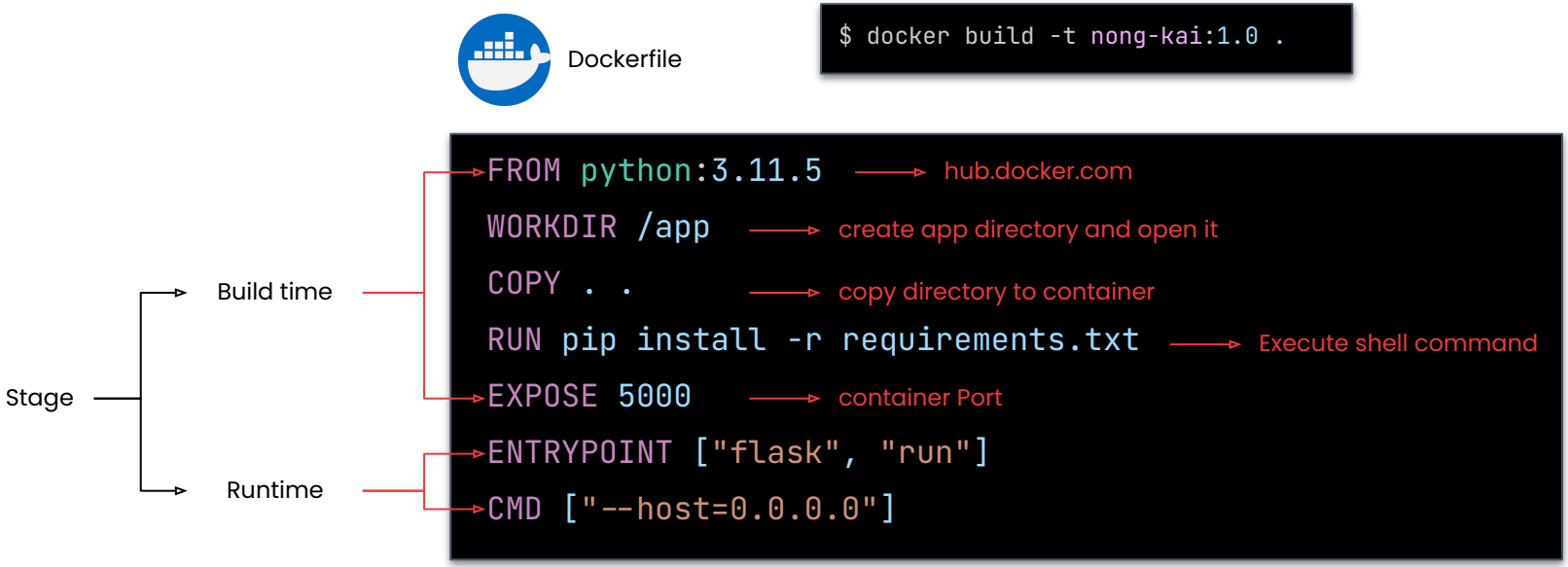
Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.



Dockerfile (Cont.)

- Dockerfile describes step by step instructions of all the commands you need to run to assemble a Container Image.



FROM instruction

FROM instruction (Cont.)

Command:

```
$ docker build -t nong-kai:1.0 .
```

FROM instruction (Cont.)

Real World:

```
FROM python:3.11.5
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

FROM instruction (Cont.)

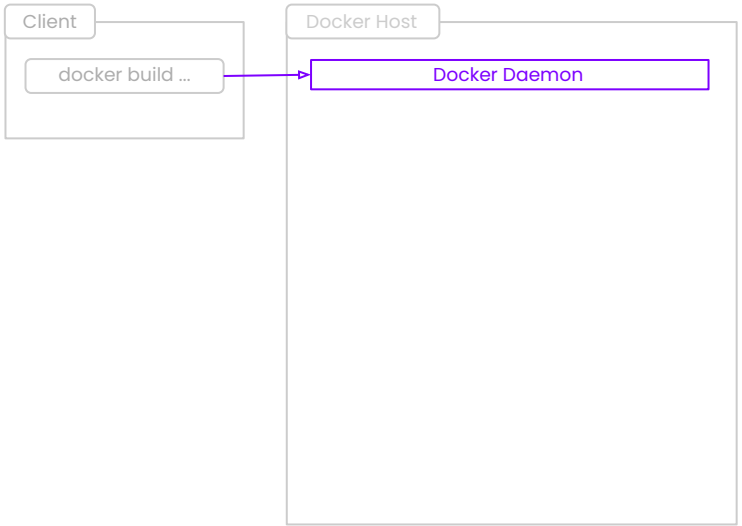
Real World:

```
FROM python:3.11.5
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



FROM instruction (Cont.)

Real World:

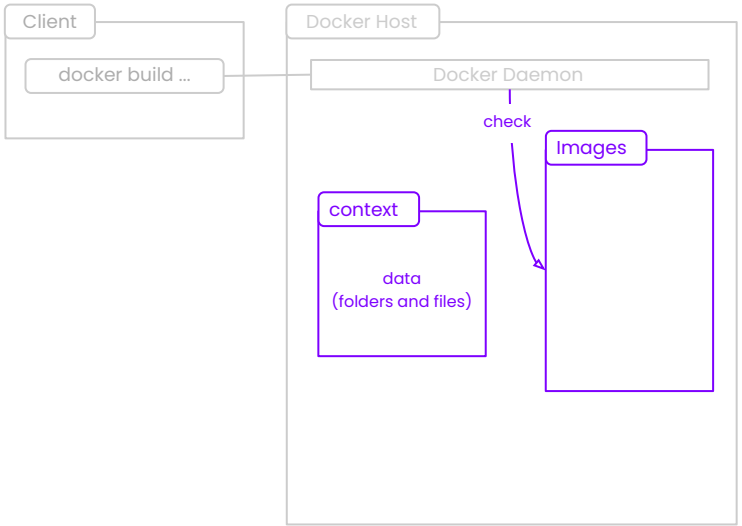
```
FROM python:3.11.5
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

context

Logical:



FROM instruction (Cont.)

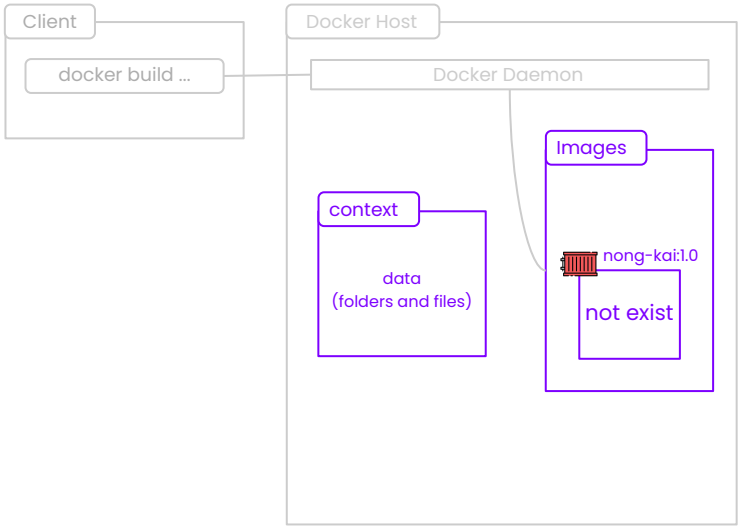
Real World:

```
FROM python:3.11.5
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



FROM instruction (Cont.)

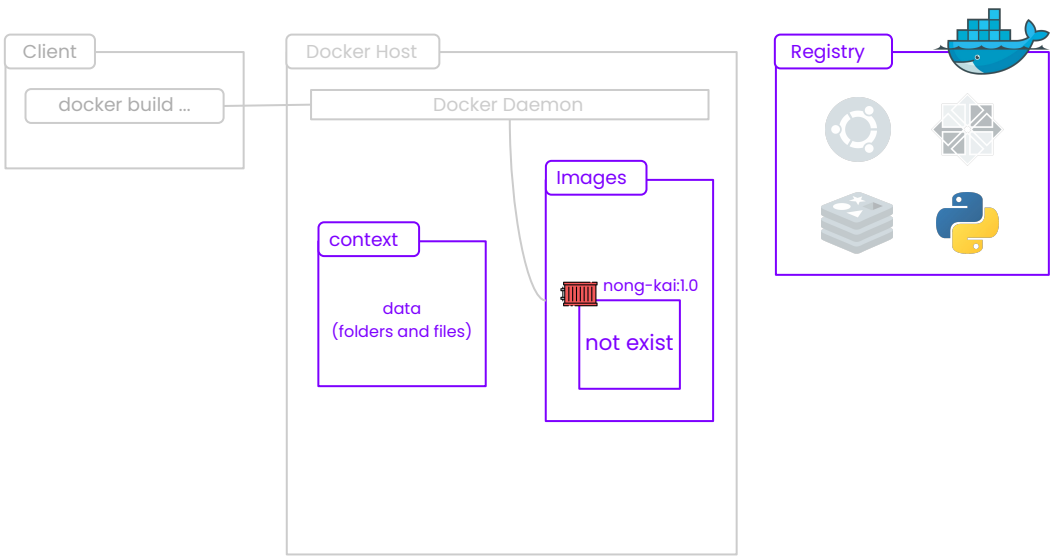
Real World:

```
FROM python:3.11.5
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



FROM instruction (Cont.)

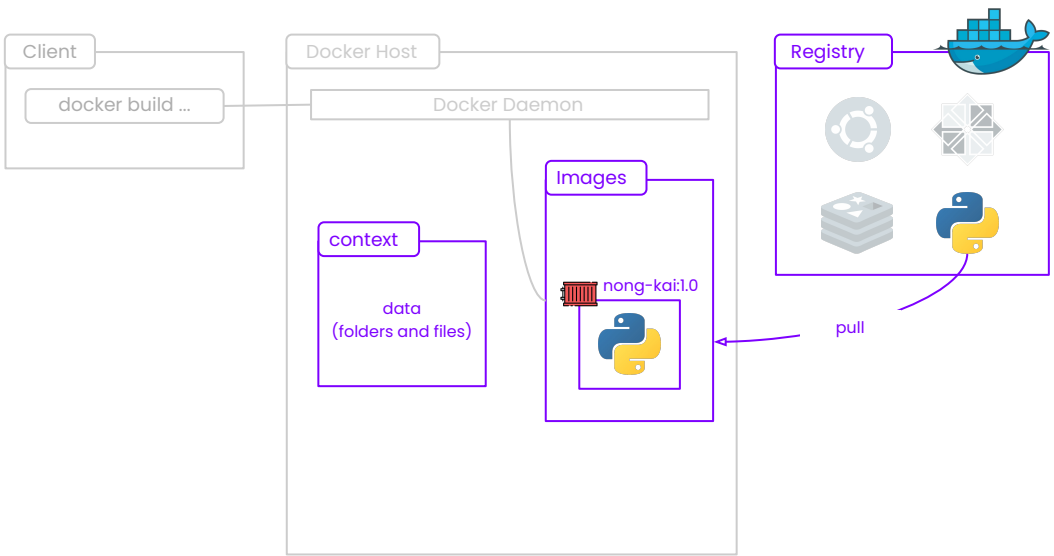
Real World:

```
FROM python:3.11.5
...
```

Command:

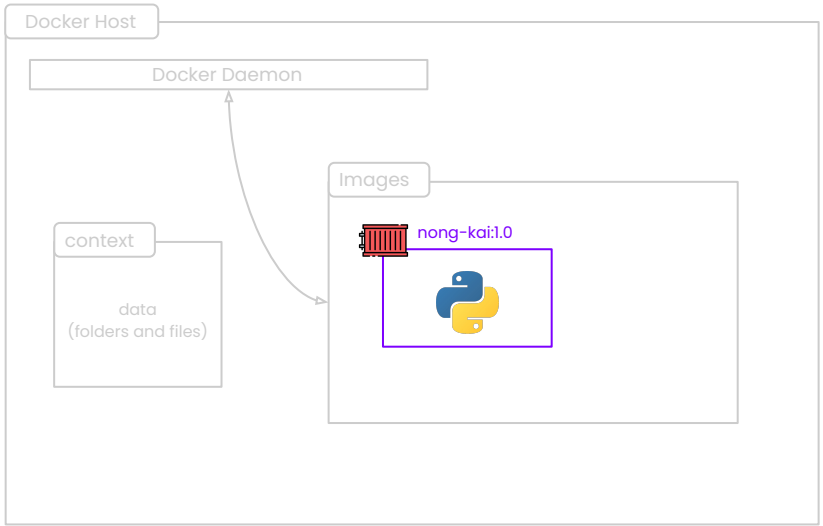
```
$ docker build -t nong-kai:1.0 .
```

Logical:



Let's focus on

Let's focus on
container image component



FROM instruction (Cont.)

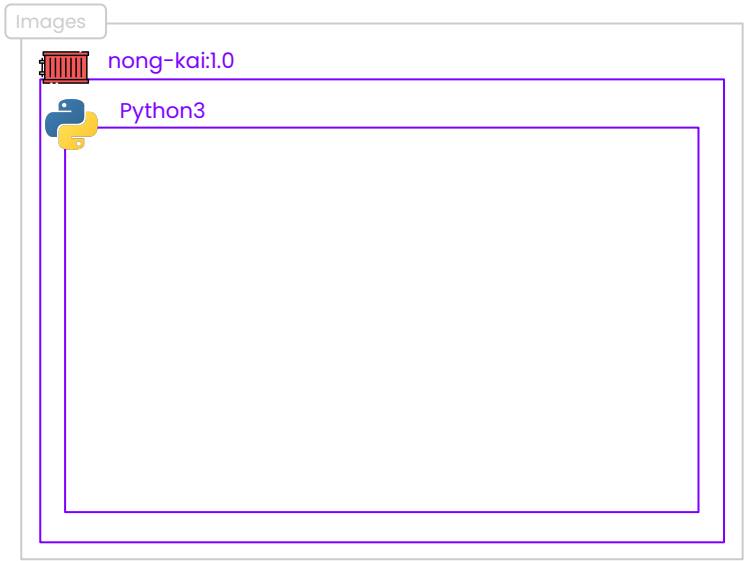
Real World:

```
FROM python:3.11.5
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



FROM instruction (Cont.)

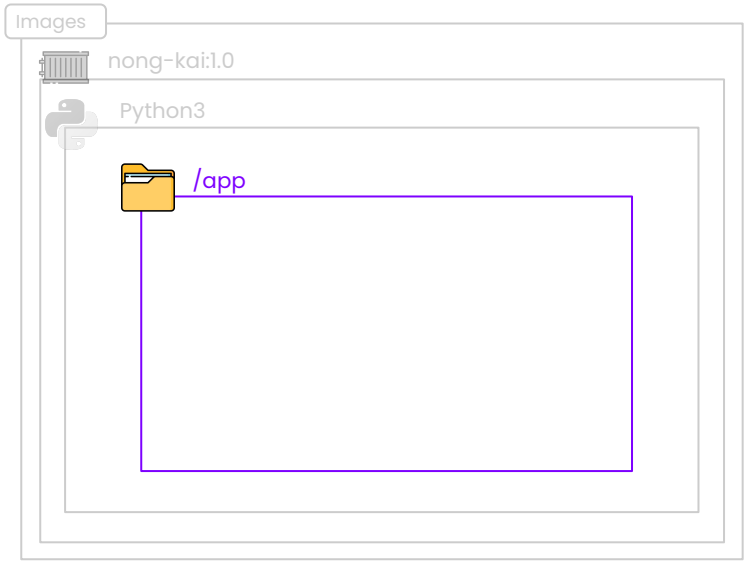
Real World:

```
WORKDIR /app
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

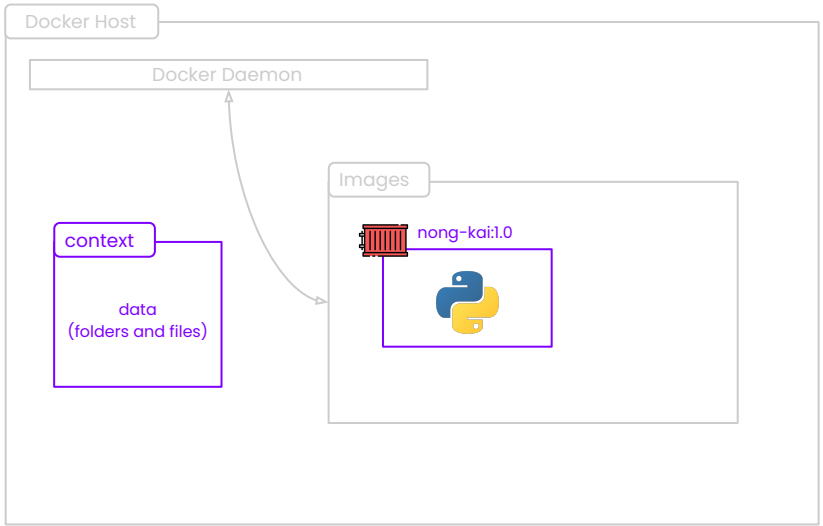
Logical:



and then focus on

and then focus on

context



FROM instruction (Cont.)

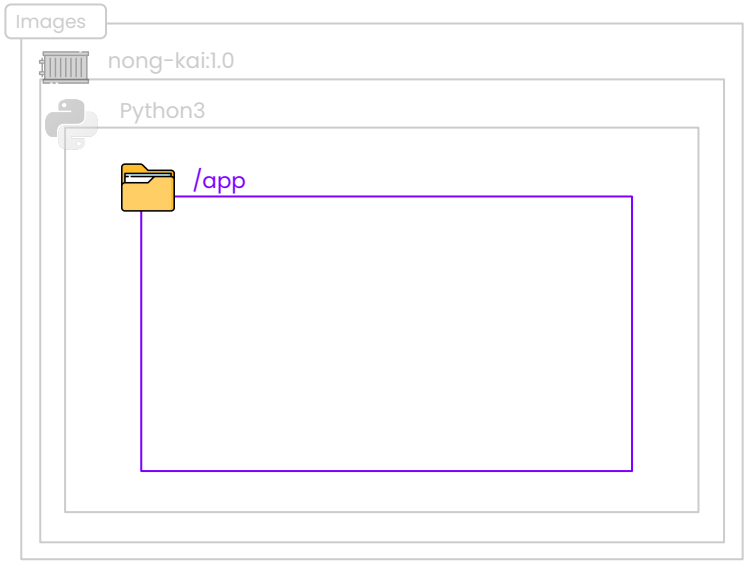
Real World:

```
WORKDIR /app
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



FROM instruction (Cont.)

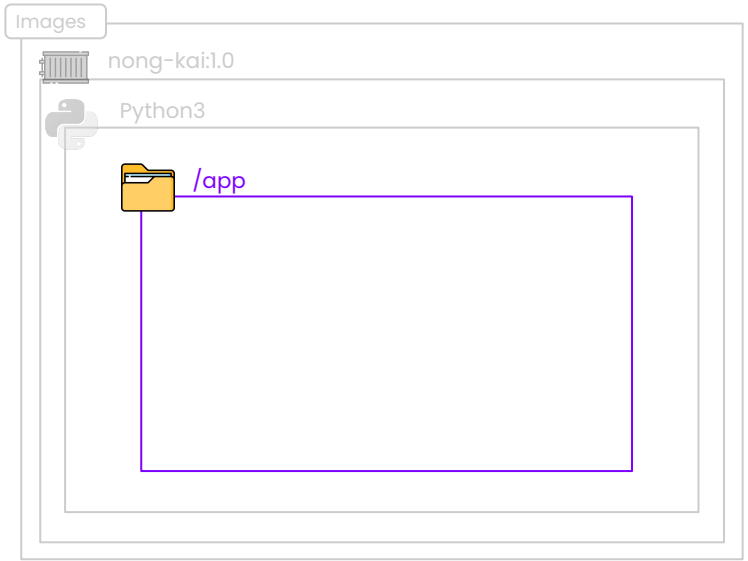
Real World:

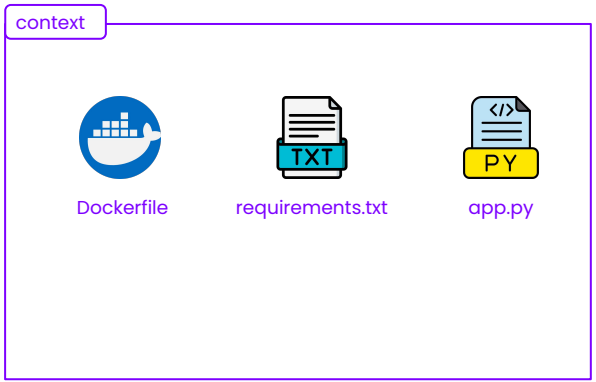
```
WORKDIR /app
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:





FROM instruction (Cont.)

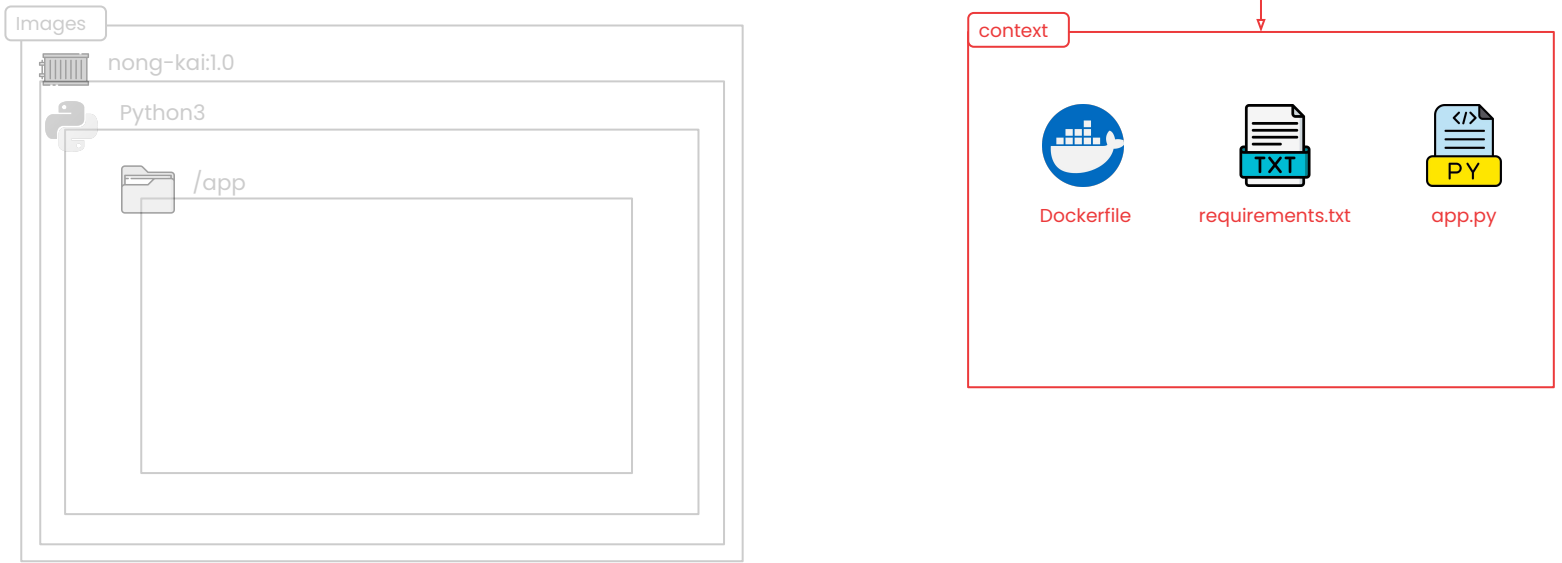
Real World:

```
COPY . .
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



FROM instruction (Cont.)

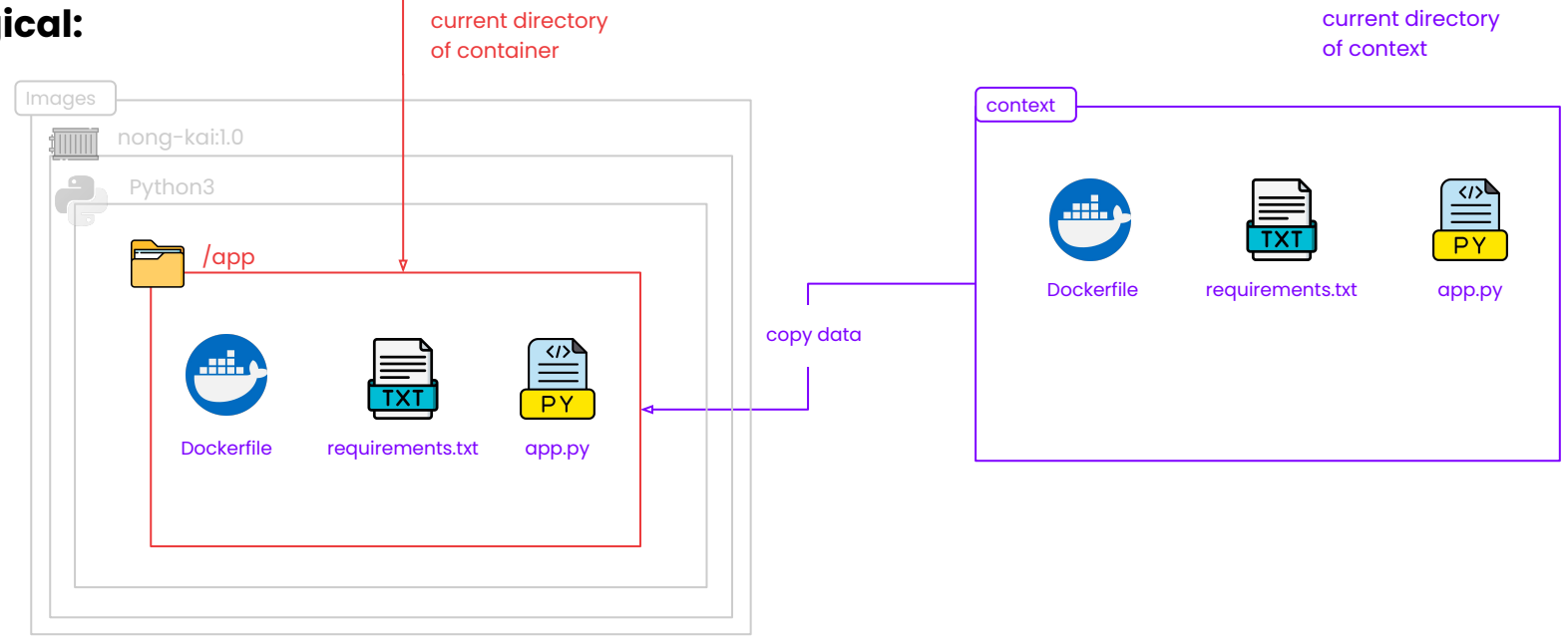
Real World:

```
COPY . .
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



Jumpbox®

Execute Shell Command

Execute Shell Command
from base image (Python)

FROM instruction (Cont.)

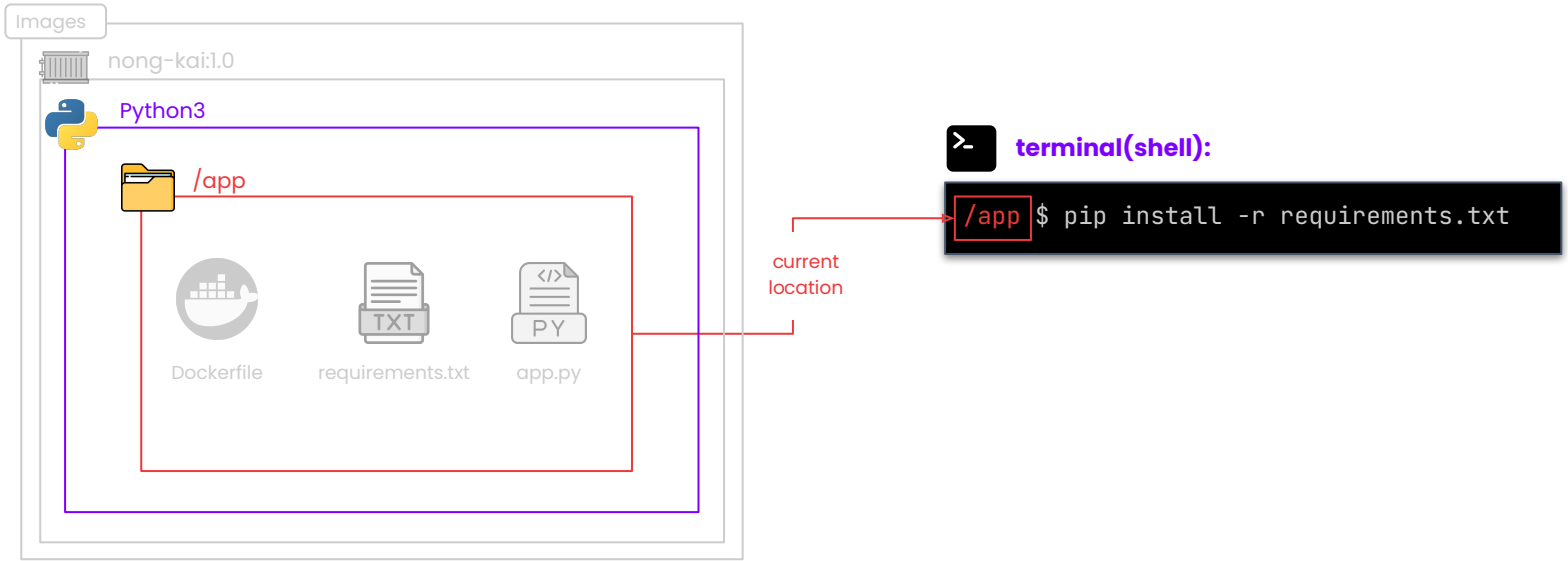
Real World:

```
RUN pip install -r requirements.txt
...
```

Command:

```
$ docker build -t nong-kai:1.0 .
```

Logical:



Jumpbox®

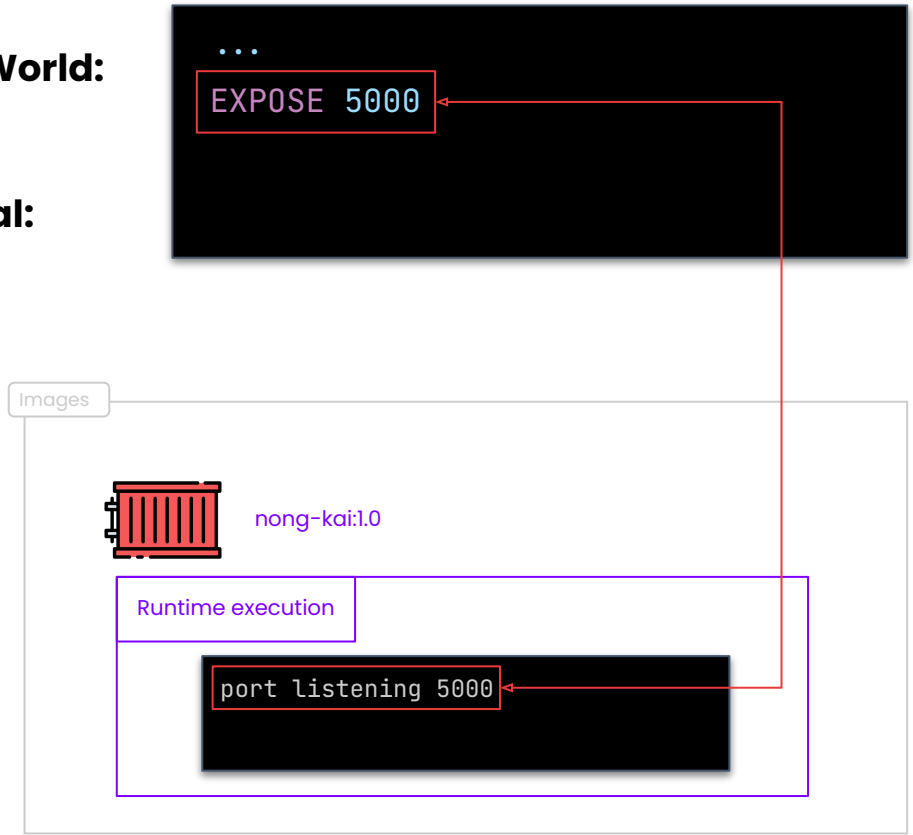
Take the instruction process

Take the instruction process
and execute it at runtime.

FROM instruction (Cont.)

Real World:

Logical:



Command:

```
$ docker build -t nong-kai:1.0 .
```

FROM instruction (Cont.)

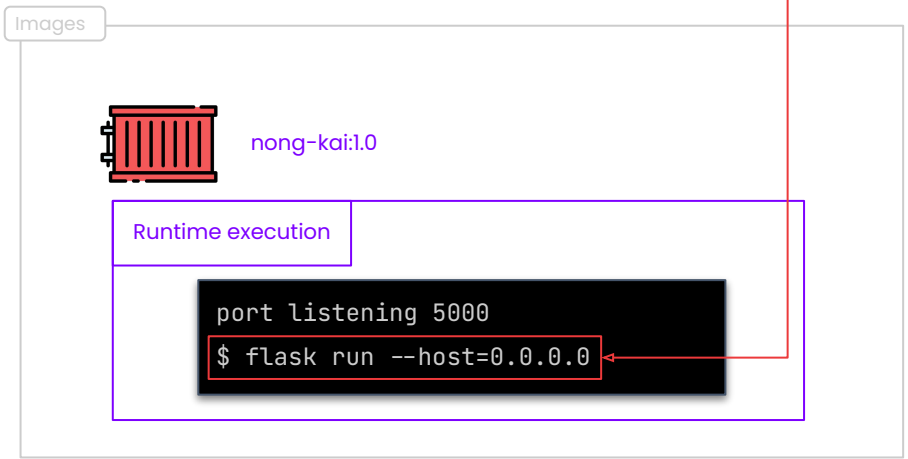
Real World:

Logical:

```
...
EXPOSE 5000
ENTRYPOINT ["flask", "run"]
CMD ["--host=0.0.0.0"]
```

Command:

```
$ docker build -t nong-kai:1.0 .
```



Activity Time

- With Paper Base -

Output what you learned in **Practical Action**

Hands-on Workshop:

Create your own Container Image

Create your own Container Image

Create your own image (Cont.)

1. Edit Dockerfile
   - Change ENTRYPOINT line:
     - ENTRYPOINT ["echo", "<your name>"]
2. Build your own image
   - $ docker build -t <your name> .
3. Run your own image
   - $ docker run <your name>

ENTRYPOINT vs CMD

ENTRYPOINT vs CMD (Cont.)

ENTRYPOINT

ENTRYPOINT configures a container that will run as an executable.

CMD

CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs

Reference:

- Optimizing Docker Images

Shell and Exec form of ENTRYPOINT

```
ENTRYPOINT ["executable", "param1", "param2"] (exec form, this is the preferred form)
ENTRYPOINT command param1 param2 (shell form)
```

Reference:

- Dockerfile

Shell and Exec form of CMD

```
CMD ["executable","param1","param2"] (exec form, this is the preferred form)
CMD ["param1","param2"] (as default parameters to ENTRYPOINT)
CMD command param1 param2 (shell form)
```

Reference:

- Dockerfile

Shell and Exec form

```
CMD echo “Hello World” (shell form)
CMD ["echo", "Hello World"] (exec form)
ENTRYPOINT echo "Hello World" (shell form)
ENTRYPOINT ["echo", "Hello World"] (exec form)
```

Reference:

Activity Time

- With Paper Base -

Output what you learned in **Practical Action**

Hands-on Workshop:

ENTRYPOINT vs CMD

ENTRYPOINT vs CMD (Cont.)

1. Change directory to entrypoint-and-cmd-demo
2. Build Dockerfile.cmd
3. Run your own image
4. Run cmd-demo image with override command
5. Build Dockerfile.entrypoint
6. Run entrypoint-demo image with default cmd
7. Run entrypoint-demo image with override command

Demo

Image Type

- Building Container Image -

Jumpbox®

Image Type

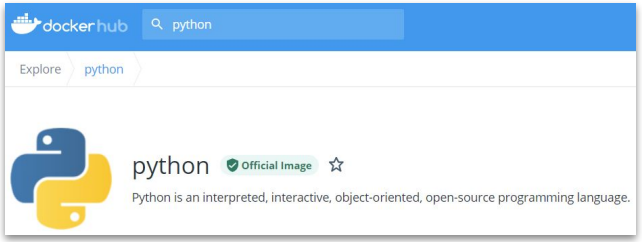
Image Type (Cont.)

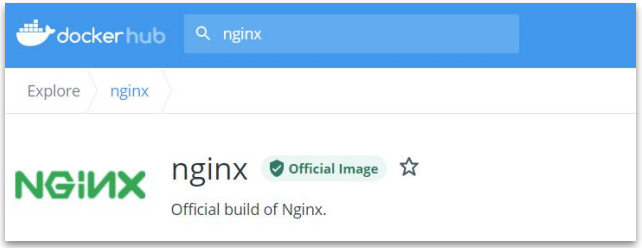
Docker Official Images: DOCKER OFFICIAL IMAGE

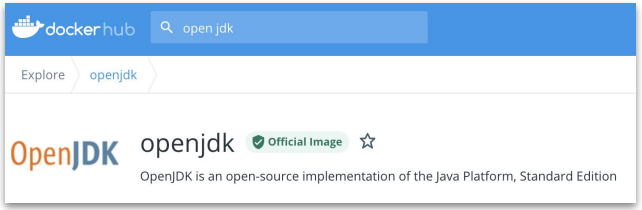
- it's recommended you use the Docker Official Images in your projects.
- These images have clear documentation, promote best practices, and are designed for the most common use cases.
- Advanced users can review Docker Official Images as part of your Dockerfile learning process.

Example:

- python:3.11.5
- nginx:1.25







Reference:

- Docker Official Images

Image Type (Cont.)

Debian Releases (-bookworm/-bullseye/-buster):

- Image that have -bookworm/ -bullseye/-buster/-stretch extension. Images tagged are codenames for different Debian releases:
  - bookworm: the **stable** Debian release is 12
  - bullseye: all version 11 variations
  - buster: all version 10 variations, under LTS support
  - stretch: all version 9 variations, extended LTS support

Example:

- python:3.11.5-bookworm
- python:3.11.5-bullseye

Reference:
- Debian Releases

Image Type (Cont.)

Slim (-slim):

- The slim image is a paired down version of the full image.
- This image generally only installs the minimal packages needed to run your particular tool.

Example:

- python:3.11.5-slim-bookworm
- python:3.11.5-slim-bullseye

Reference:
- Docker Official Images

Image Type (Cont.)

Alpine (-alpine):

- Alpine images are based on the Alpine Linux Project, which is an operating system that was built specifically for use inside of containers.
- Alpine Linux is built around musl libc and busybox. This makes it small and very resource efficient.
- A container requires no more than 8 MB and a minimal installation to disk requires around 130 MB of storage.

Example:

- python:3.11.5-alpine, python:3.11.5-alpine3.18
- python:3.11.5-alpine3.17

Reference:
- Alpine Linux

Image Type (Cont.)

windowsservercore (-windowsservercore):

- This is a base image for Windows Server containers. This image carries the Windows Server Core base OS image.

- For more information about servicing lifecycles, visit Base Image Servicing Lifecycles.

Example:

- python:3.11.5-windowsservercore-ltsc2022
- python:3.11.5-windowsservercore-1809

Reference:
- Windows Server Core
- Base Image Servicing Lifecycles

Image Type (Cont.)

```
docker pull --quite python:3.9.6
docker pull --quite python:3.9.6-slim
docker pull --quite python:3.9.6-buster
docker pull --quite python:3.9.6-alpine
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
python	3.9.6-alpine	d4d6be1b90ec	3 days ago	45.1MB
python	3.9.6-slim	c4102d3b0680	3 days ago	115MB
python	3.9.6	b2278d5ae327	3 days ago	886MB
python	3.9.6-buster	b2278d5ae327	3 days ago	886MB

How to choose image type?

How to choose Image Type (Cont.)

Size and Capabilities:

- If size of image is small, cluster or server can download and deploy faster.
- Ubuntu/Debian has apt for a package manager.
- Alpine uses musl libc and has apk.

Long Term Support (LTS):

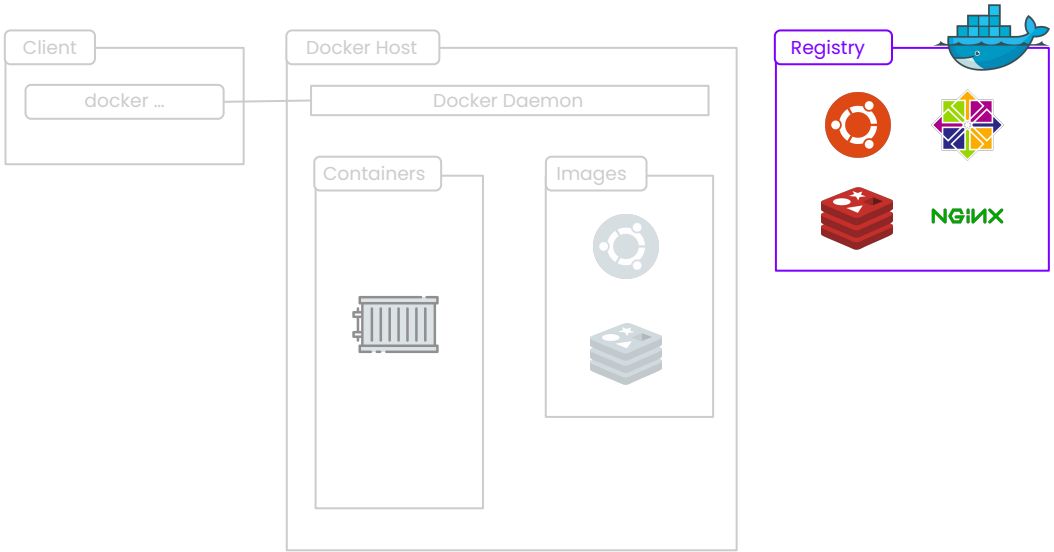
- a product lifecycle management policy in which a stable release of computer software is maintained for a longer period of time than the standard edition.

Reference:

- Windows Server Core
- Base Image Servicing Lifecycles

Container Registry (Cont.)

A Container registry stores Container Images.



Container Registry

- Publish to Cloud Container Registry -

Jumpbox®

Publish to Cloud Container Registry

Publish to Cloud Container Registry (Cont.)

- To **publish** an image to Container Registry, you must first name your local image using your Container Registry username and the repository name that you created.

Reference:
- Publish to Container Registry
- $ docker build
- $ docker tag

Publish to Cloud Container Registry (Cont.)

- To **publish** an image to Container Registry, you must first name your local image using your Container Registry username and the repository name that you created.

Syntax:

```
$ docker build -t <username>/<container-image-name>:<tag>
```

Or (if you have a container image already)

```
$ docker tag <existing-image> <username>/<container-image-name>:<tag>
```

Reference:

- Publish to Container Registry
- $ docker build
- $ docker tag

$ docker login

- Publish to Cloud Container Registry -

Jumpbox®

$ docker login

Reference:

- $ docker login

$ docker login (Cont.)

Syntax:

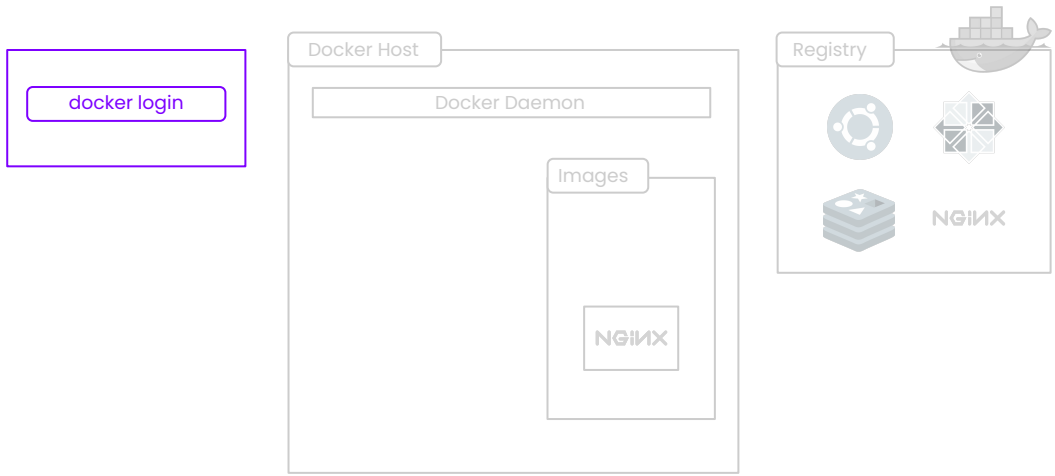
```
$ docker login [OPTIONS] [SERVER]
```

$ docker login (Cont.)

Syntax:

```
$ docker login [OPTIONS] [SERVER]
```

Logical:

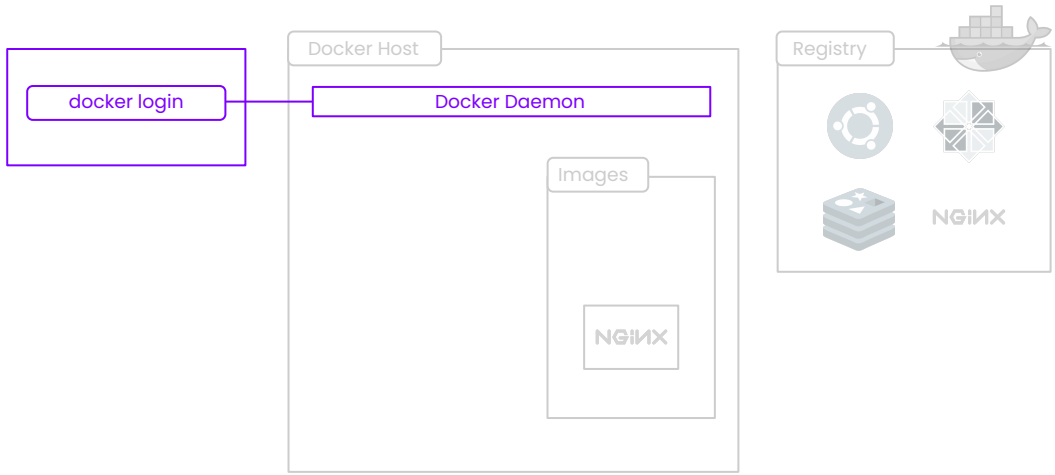


$ docker login (Cont.)

Syntax:

```
$ docker login [OPTIONS] [SERVER]
```

Logical:

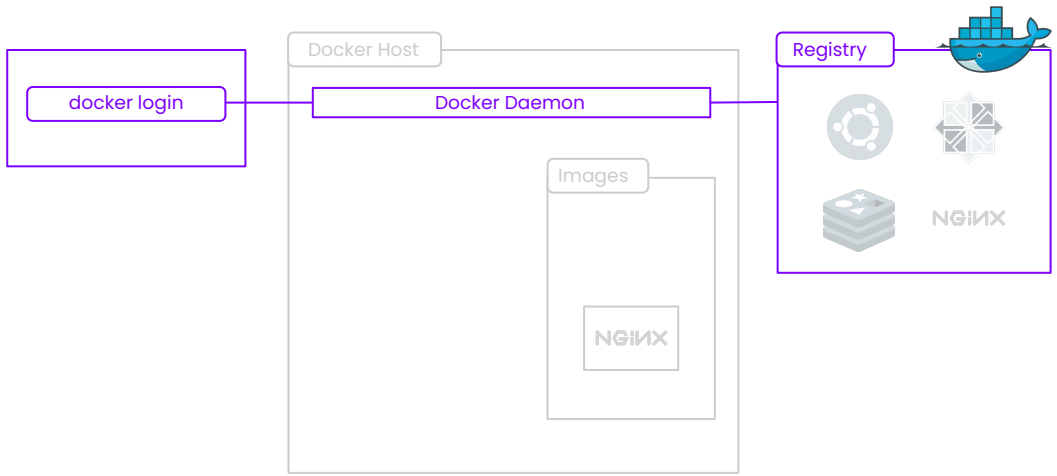


$ docker login (Cont.)

Syntax:

```
$ docker login [OPTIONS] [SERVER]
```

Logical:

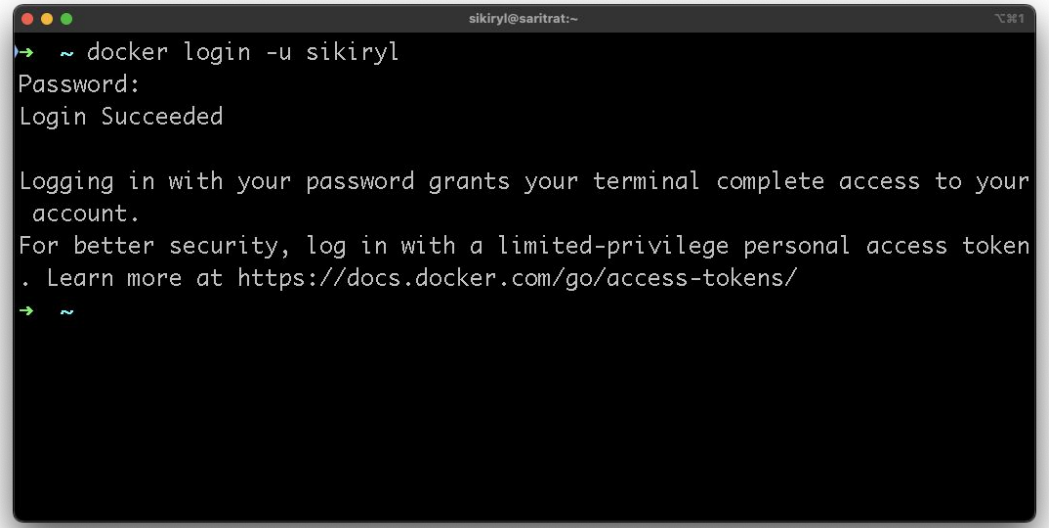


$ docker login (Cont.)

Syntax:

```
$ docker login [OPTIONS] [SERVER]
```

Real world:



$ docker push

- Publish to Cloud Container Registry -

Jumpbox®

$ docker push

Reference:

- $ docker push

$ docker push

Syntax:

```
$ docker push [OPTIONS] NAME[:TAG]
```

Reference:

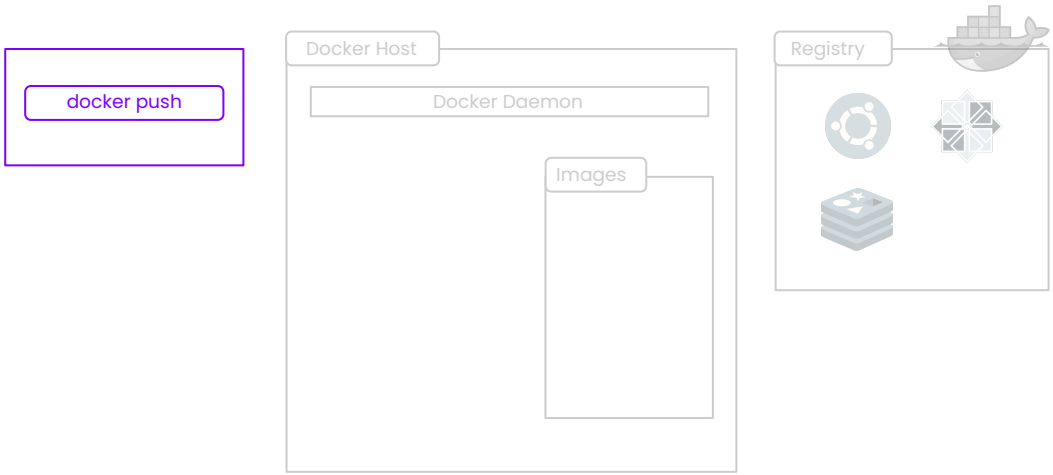
- $ docker push

$ docker push (Cont.)

Syntax:

```
$ docker push [OPTIONS] NAME[:TAG]
```

Logical:



Reference:

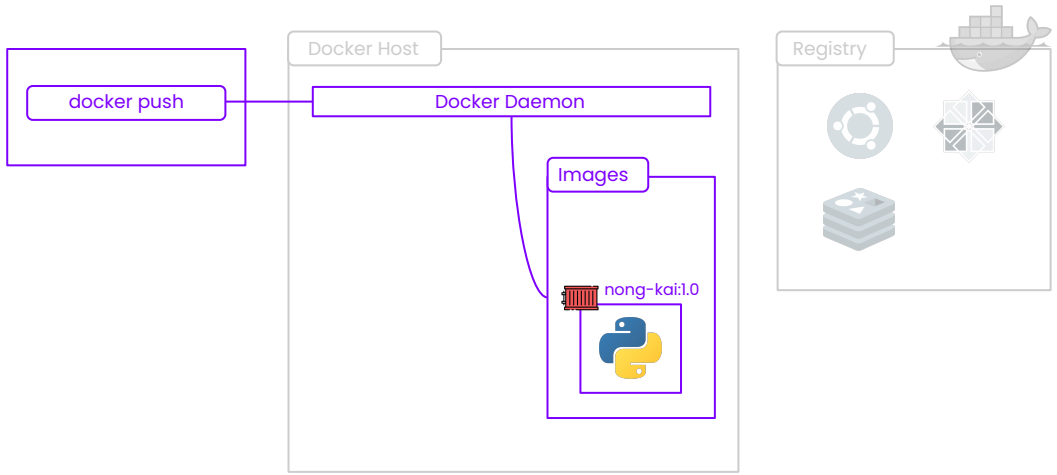
- $ docker push

$ docker push (Cont.)

Syntax:

```
$ docker push [OPTIONS] NAME[:TAG]
```

Logical:



Reference:

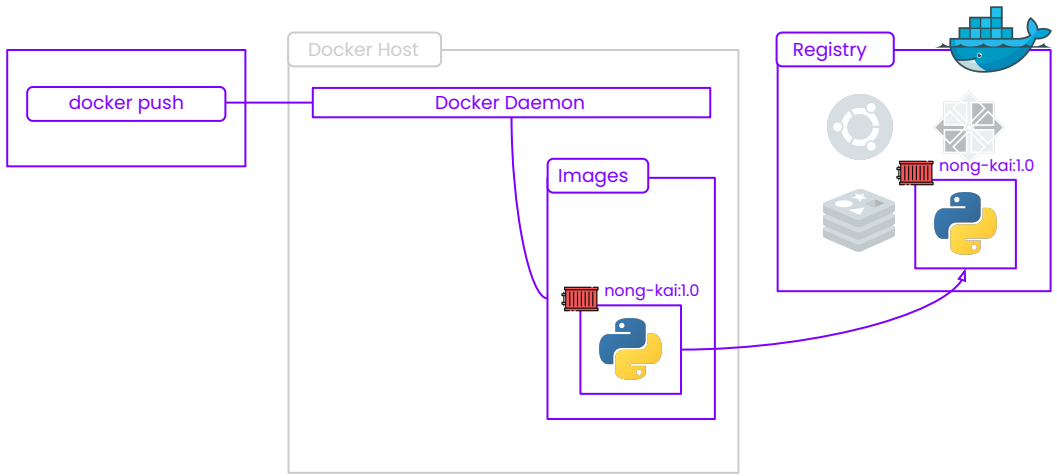
- $ docker push

$ docker push (Cont.)

Syntax:

```
$ docker push [OPTIONS] NAME[:TAG]
```

Logical:



Reference:

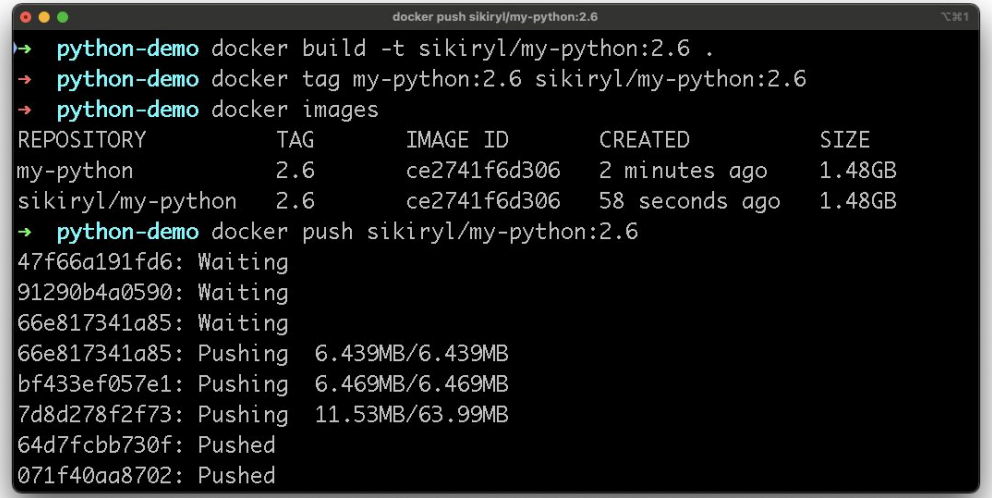
- $ docker push

$ docker push (Cont.)

Syntax:

```
$ docker push [OPTIONS] NAME[:TAG]
```

Real world:



Reference:

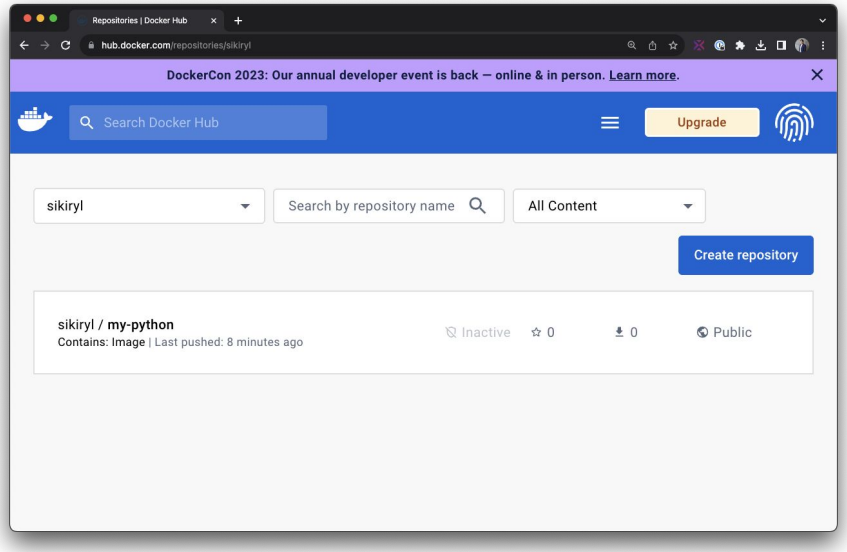
- $ docker push

$ docker push (Cont.)

Syntax:

```
$ docker push [OPTIONS] NAME[:TAG]
```

Real world:



Reference:

- $ docker push

"The palest ink is better than the best memory"

-Chinese proverb-

Activity Time

- With Paper Base -

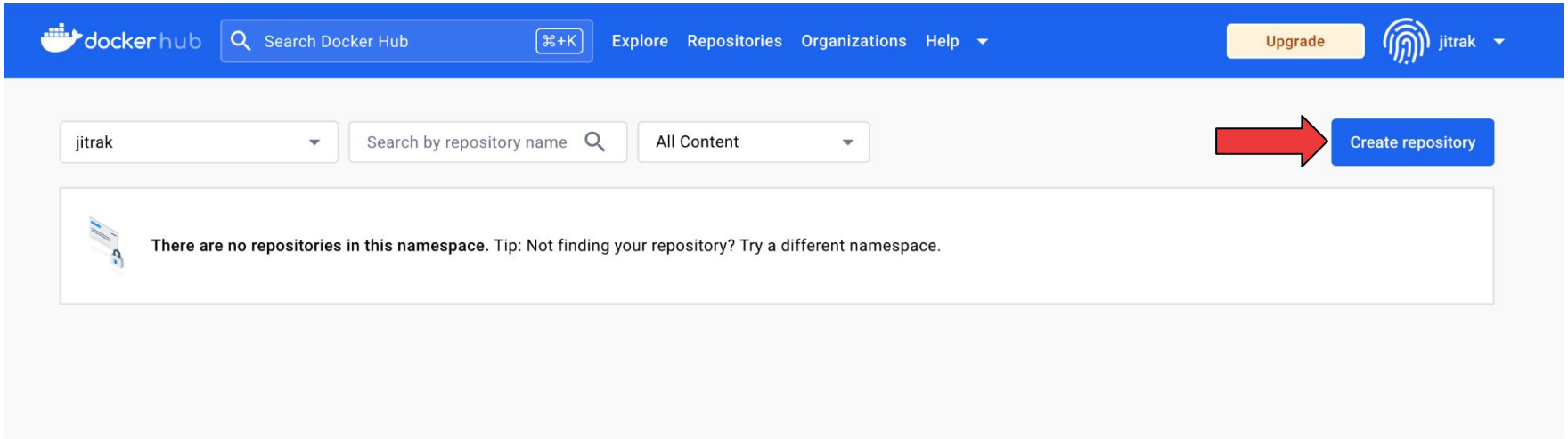
Output what you learned in **Practical Action**

Hands-on Workshop:

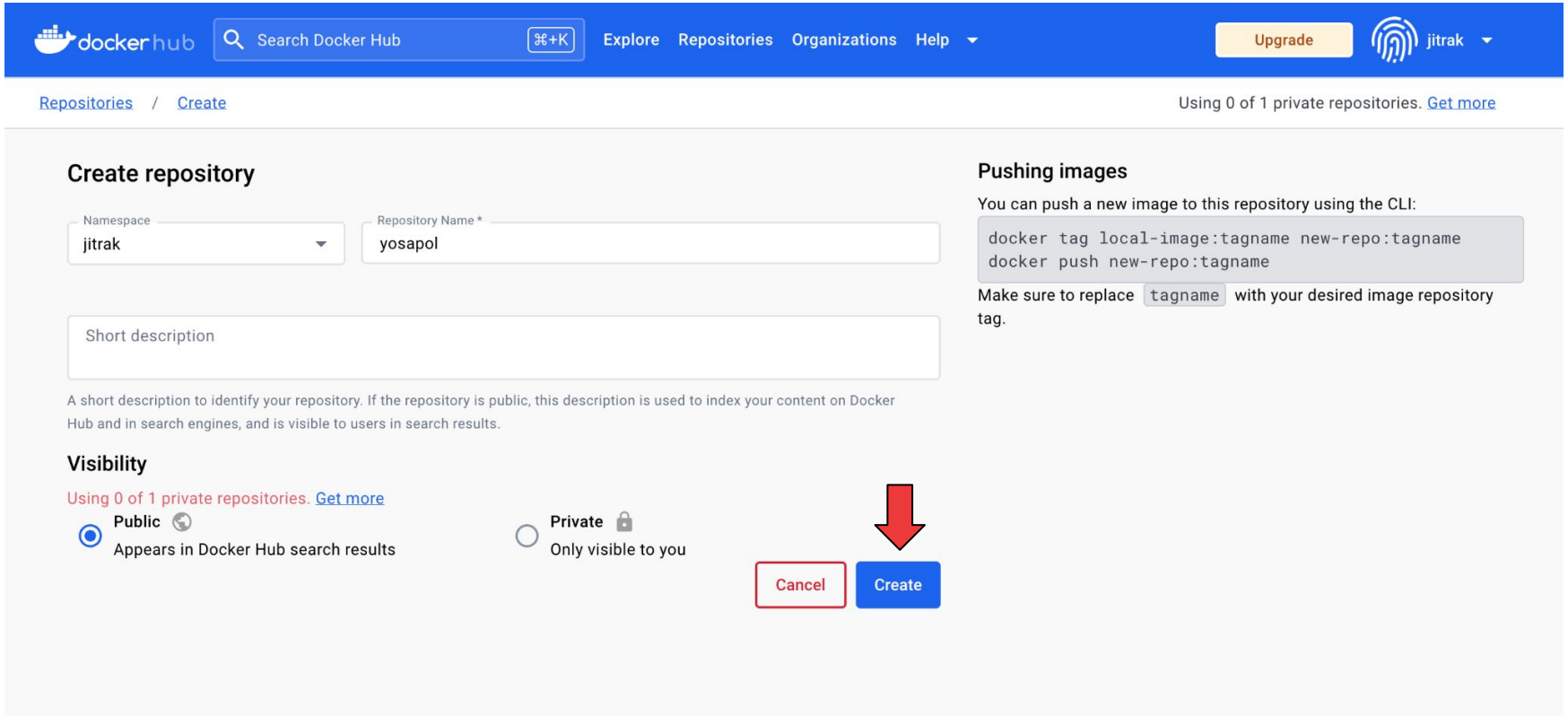
Push image to Docker Hub

Push image to Docker Hub (Cont.)

1. Change directory to simple-demo
2. Go to https://hub.docker.com/
3. Register account
4. Create repository

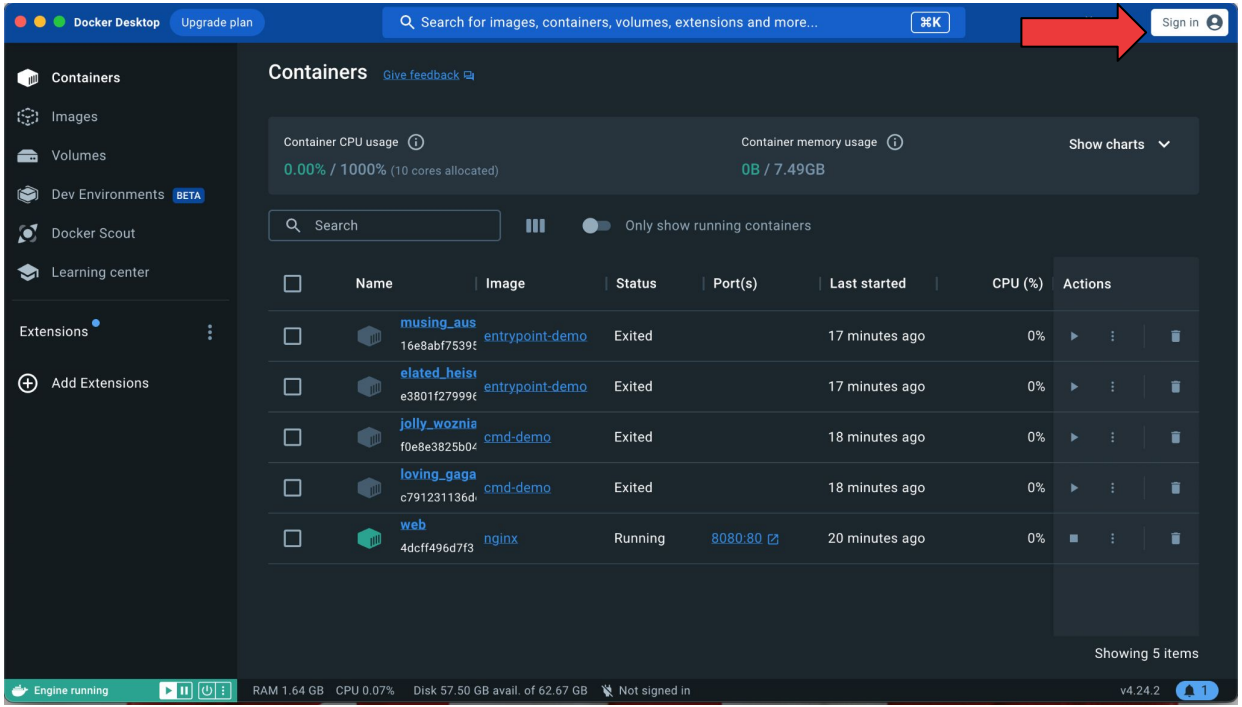


Push image to Docker Hub (Cont.)



Push image to Docker Hub (Cont.)

5. Login with Docker hub user



Push image to Docker Hub (Cont.)

6. Tag image name from first image
7. Push your docker image to Docker hub
8. Run friend image
9. Rebuild image with docker buildx and push
10. Re-run friend image with new tag

Demo

Container Image command

- docker images
  - List all docker images in machine
  - Same as docker image ls
- docker rmi
  - Remove docker image
  - Same as docker image rm
- docker pull
  - Pull image from image repository
- docker push
  - Push image to image repository
- docker build
  - Build image from Dockerfile
  - Same as docker image build
- docker tag
  - Attach tag to image
  - Same as docker image tag

Container Basic command

- docker run
  - Run container from image
- docker exec
  - Execute command to target container
  - Usually use with -it for debug or shell to container
- docker logs
  - Print logs from target container
- docker ps
  - Show all running containers
  - Use -a option for show all containers (docker ps -a)
- docker rm
  - Remove target container
- docker stop
  - Stop target running container

Docker Management command

- docker image
- docker network
- docker node
- docker secret
- docker service
- docker stack
- docker stats
- docker swarm
- docker system
- docker volume

Docker Network

Docker Management command

- Bridge: The default network driver. If you don’t specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate

- Host: For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly

- Overlay: Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other

- Macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network

- none: For this container, disable all networking

Reference Pictures:

- Networking overview

Docker Storage

Manage data in Docker

Default all files created inside a container are stored on a writable container layer

- Data doesn’t persist when that container no longer exists
- Difficult to get the data out of the container
- Extra abstraction reduces performance as compared to using data volumes

Reference Pictures:

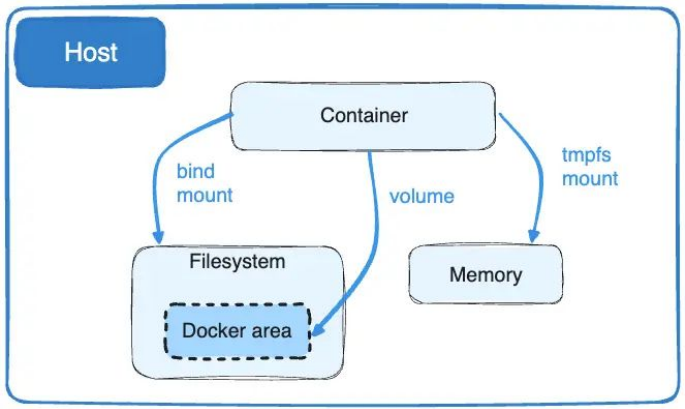
- Manage data in Docker

Type of mount

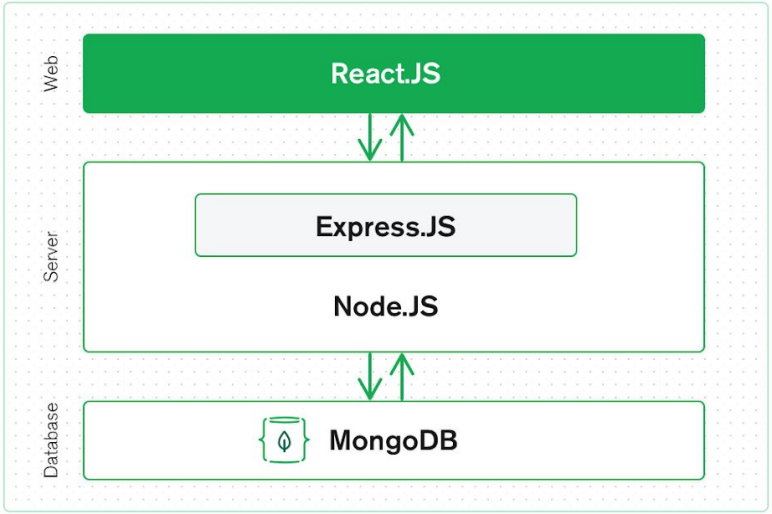
- Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.

- Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

- tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.



MERN Stack



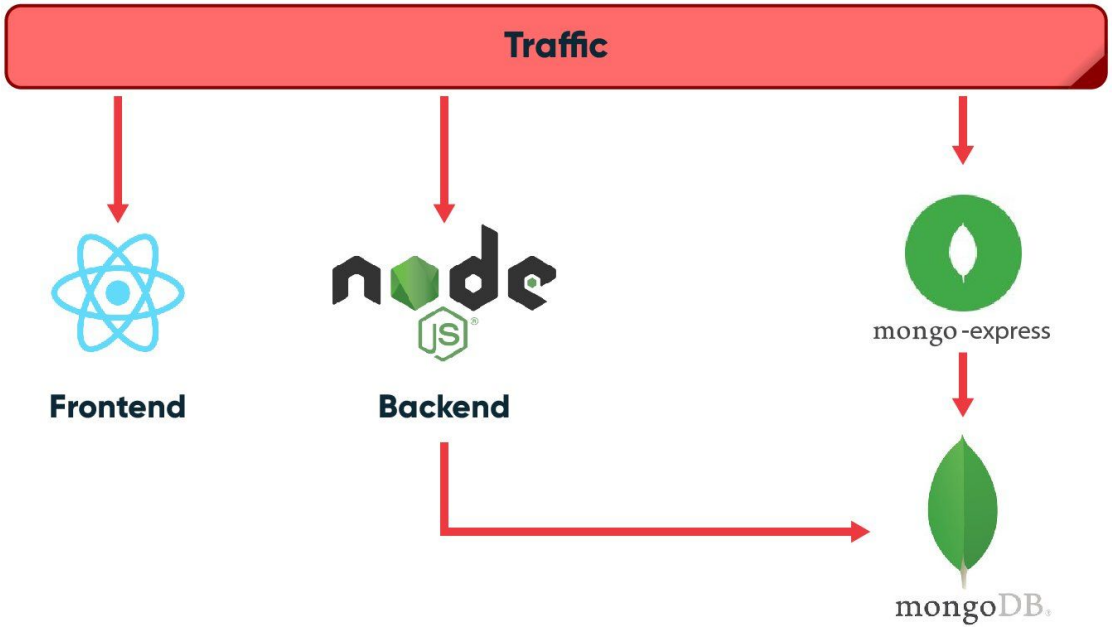
Reference Picture

- : A ComHow

Reference Pictures:

- How to Use MERN Stack: A Complete Guide

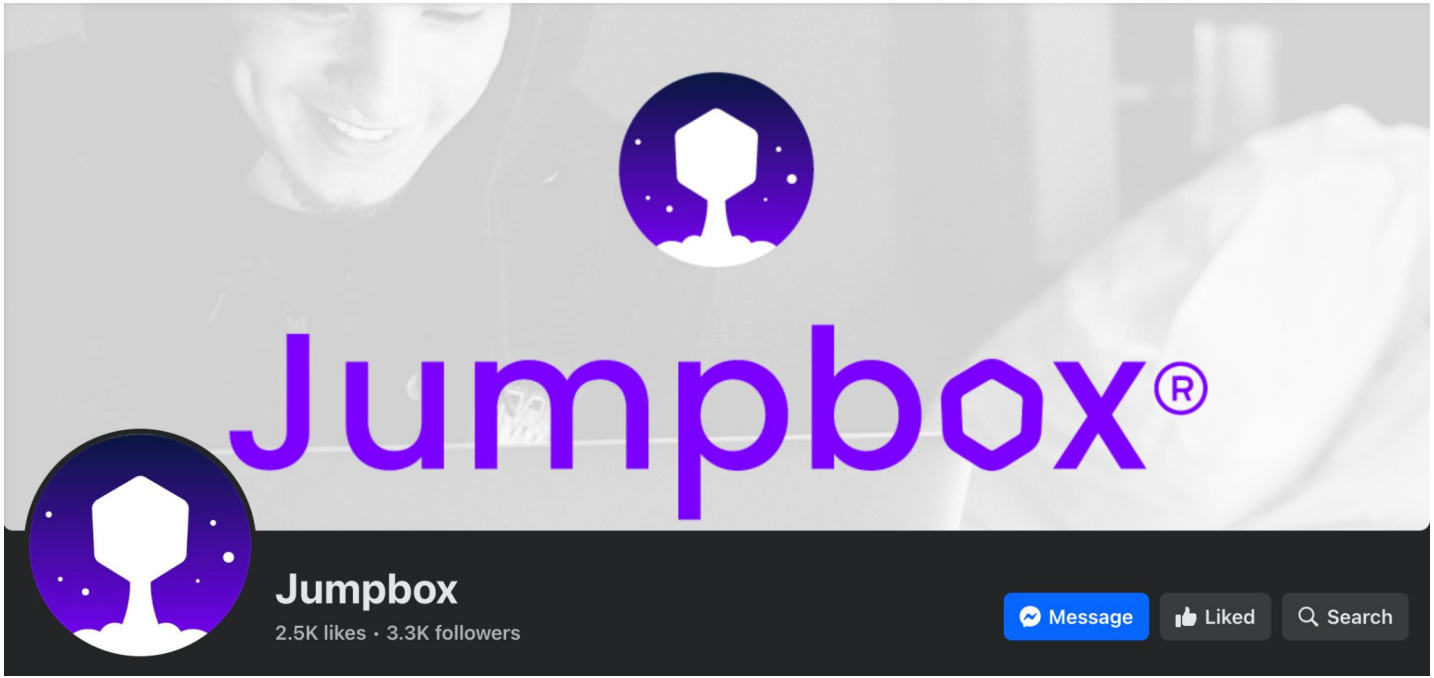
Application Overview & Running Application in Docker



Remove all container

1. Force remove all container

facebook



https://www.facebook.com/jumpbox.academy

Jumpbox®

YouTube

Jumpbox

@jumpbox.academy · 1.92K subscribers · 30 videos

More about this channel ...more

https://www.youtube.com/@jumpbox.academy

Contact Us

Jumpbox

@jumpbox

admin@jumpbox.co

063-245-2168 (JoJo)

062-796-1559 (Beau)

Jumpbox®

"เราเชื่อว่าการเรียนรู้ทำให้ชีวิตคุณดีขึ้น"

-Jumpbox Team-

Jumpbox®

Jumpbox®