Containerization and Orchestration - An Overview

Application Frameworks (SE3040)

Vishan Jayasinghearachchi
Lecturer
Department of Software Engineering, Faculty of Computing
vishan.j@sliit.lk

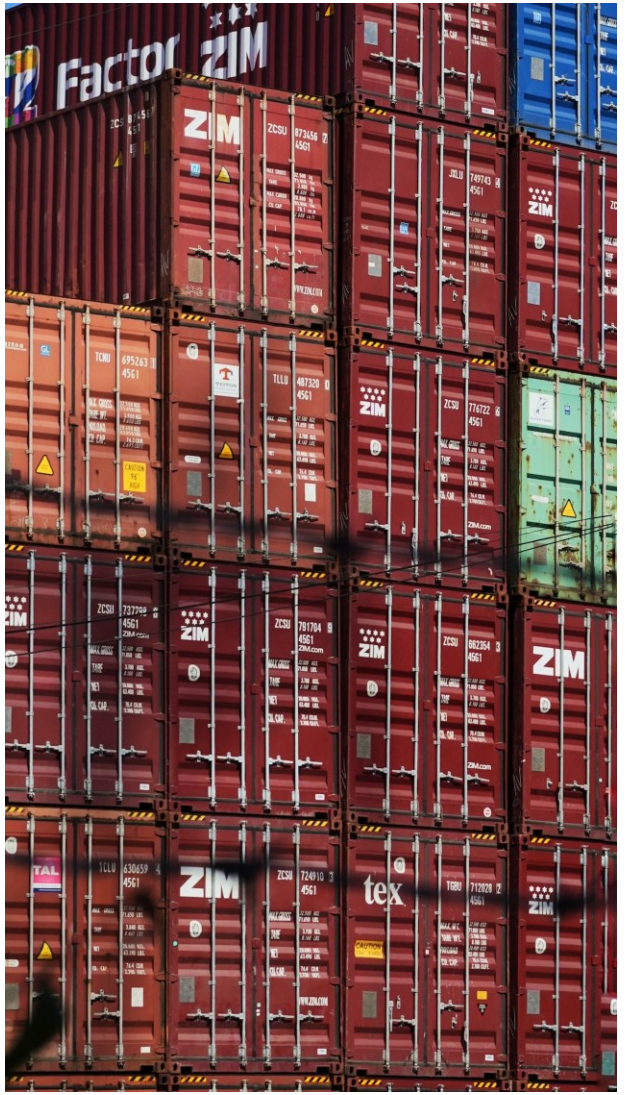
Learning Objectives

After completing this lecture, you will be able to,

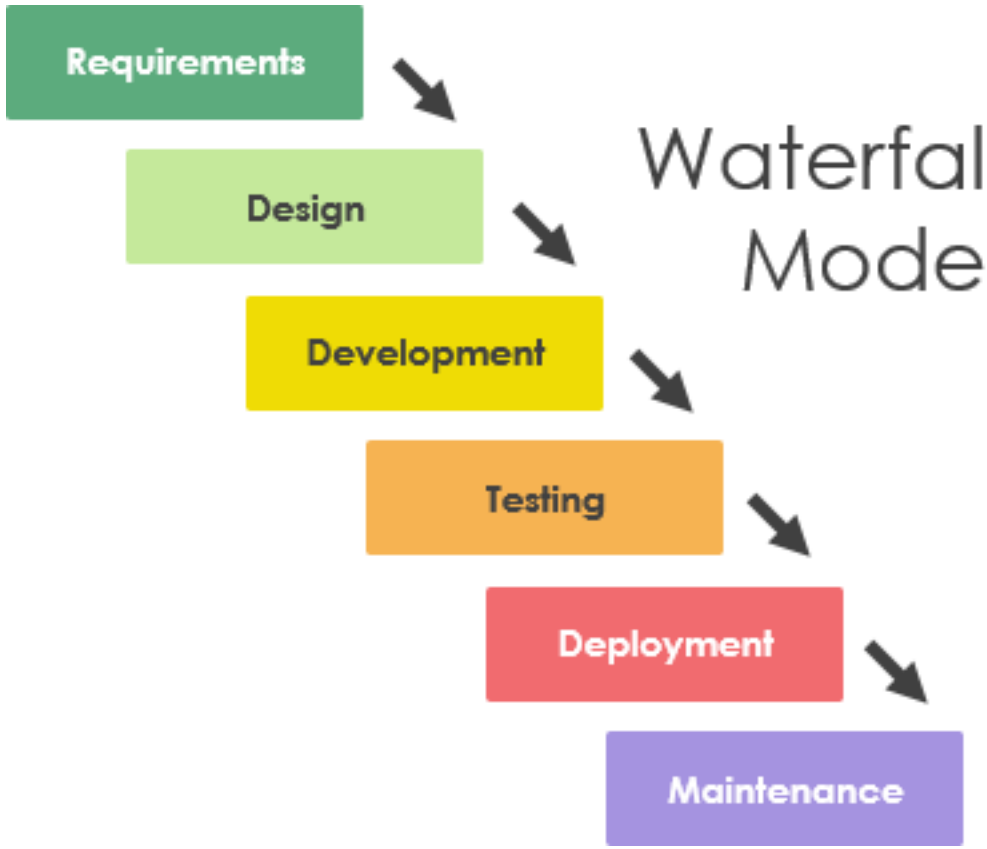
- Describe two main virtualization technologies (Hardware/ Operating System) available in the computing domain.
- Describe what containerization is and the need for it.
- Analyze a scenario to identify which virtualization technologies such as virtual machines and containerization are applicable to the given scenario.

Contents

- Traditional Application Deployment
- Cloud Native Applications
- Microservices Architecture
- Containers
  - VMs vs Containers
  - Containers: Underlying Technology
  - Docker Overview
- Container Orchestration
  - Kubernetes Overview



Traditional Application Deployment



Waterfall Model

Capacity Planning

Source: https://www.visual-paradigm.com/guide/software-development-process/what-is-a-software-process-model/

Traditional Application Deployment

- Capacity Planning — key part of deploying applications traditionally (i.e., without containers or cloud-native tools).
  - Estimate the required amount computational resources to deploy the developed application and purchase them. — CPU, RAM, disk space, bandwidth, etc.
  - An expensive operation.
  - Involves guesswork. — basically bunch of guesses and you tell that this is the amount of resources
  - A video on this.

Cloud Native Applications

modern software development approach designed to make the most of the cloud environment.

- With the introduction of **cloud computing**, users were provided with the ability to simply utilize **virtualized hardware** on cloud platforms such as VMs instead of actual hardware.

With the rise of cloud computing, developers no longer needed to buy physical servers.

Instead, they could use virtual machines (VMs) on cloud platforms (like AWS, Azure, or GCP), making infrastructure more accessible and scalable.

- However, it was not possible to take full advantage of the flexibility provided by the Cloud with the traditional application development architectures and practices.

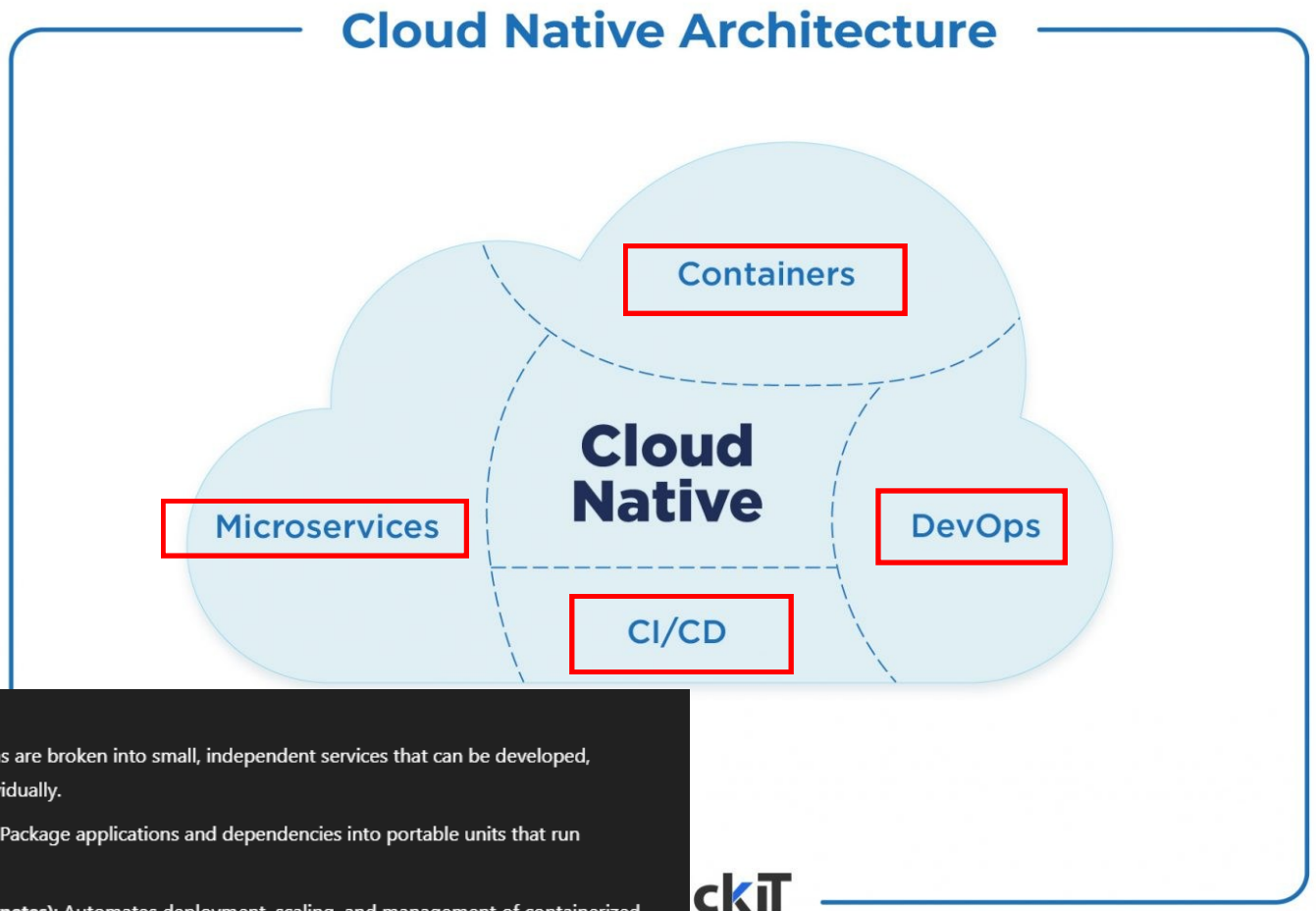
moving traditional applications to the cloud didn't fully utilize the cloud's flexibility

Cloud Native Applications

- Cloud Native Applications to the rescue!

- Cloud native architecture is an innovative software development approach that is **specially designed to fully leverage the cloud computing model**.

Cloud Native Applications



✅ **Key Features:**

- **Microservices**: Applications are broken into small, independent services that can be developed, deployed, and scaled individually.
- **Containers (e.g., Docker)**: Package applications and dependencies into portable units that run consistently anywhere.
- **Orchestration (e.g., Kubernetes)**: Automates deployment, scaling, and management of containerized applications.
- **DevOps & CI/CD**: Encourages automation and faster delivery cycles through continuous integration and delivery.
- **Scalability & Resilience**: Automatically scales based on demand and recovers from failures without manual intervention.

ckiT
ware Development

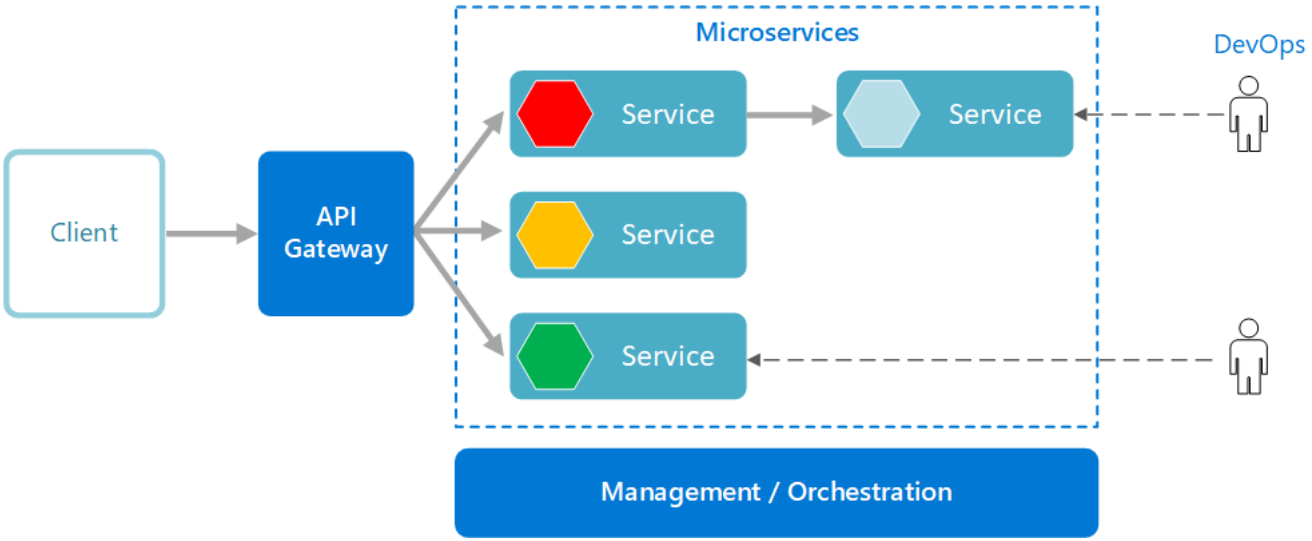
levops/cloud-native-architecture/

Microservices Architecture

What are Microservices?

"Microservices architecture is an approach in which a single application is composed of many loosely coupled and independently deployable smaller services." (Source – IBM)

Microservices Architecture



Source: https://learn.microsoft.com/en-us/azure/architecture/microservices/

Microservices Architecture

- Characteristics of Microservices
  - Have their own technology stack and typically means of managing data as well (Databases etc.).

    each microservice can use its own programming language, database, and tools best suited for its purpose.
  - Communicate with each other using APIs, Message Brokers and event streaming.
  - Organized by the business capability (bounded context) and not as horizontal layers such as data access, messaging etc.

    billing, user management

    not technical layers (e.g., controllers, services).
  - Services are loosely coupled, highly functionally cohesive and independent.

    service is functionally focused

Microservices Architecture

- Benefits of Microservices
  - Each microservice could be developed and deployed independent of other services.
  - Because of this independence, the development team could use the most suitable technology stack for each service, rather than depending on an agreed set of tools for all services (Polyglot programming).
    Teams can choose different languages, frameworks, and databases based on the service's needs.
  - Precise scaling - each component of the application could be scaled independently of others, increasing the resource usage efficiency.
    scale only the needed services, saving resources and improving performance

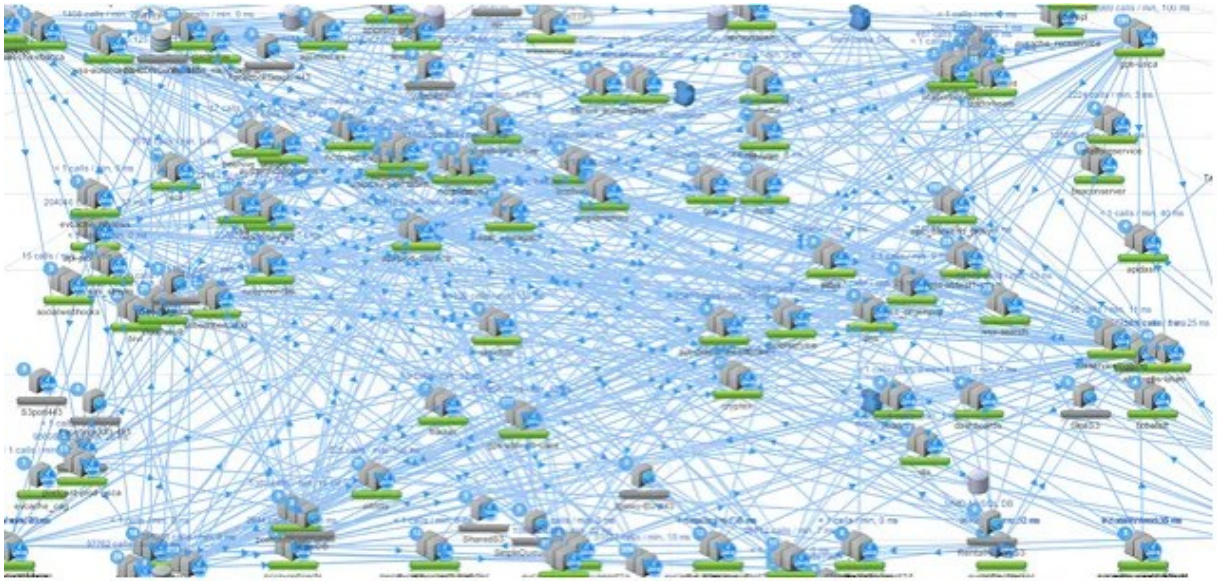
Microservices Architecture

- Read more on Microservices
  - Martin Fowler – Microservices
  - Microservices.io reference

Microservices Architecture

Q: Microservices is just an architecture. How is this architecture implemented?

Use Containers



Source: https://netflixtechblog.com/announcing-ribbon-tying-the-netflix-mid-tier-services-together-a89346910a62

Containers

- Containers are a very popular approach for implementing Microservices Architectures.

Containers

- What is a container?
  - A Container is a package of software that contain all the necessary elements of a software (binaries, libraries, files, configurations etc.) in a single lightweight executable which runs consistently on any infrastructure.

A container is a lightweight, standalone, and executable software package that includes: Application code, Binaries, Libraries, Configuration files,Dependencies

ensures that the application runs consistently across different computing environments

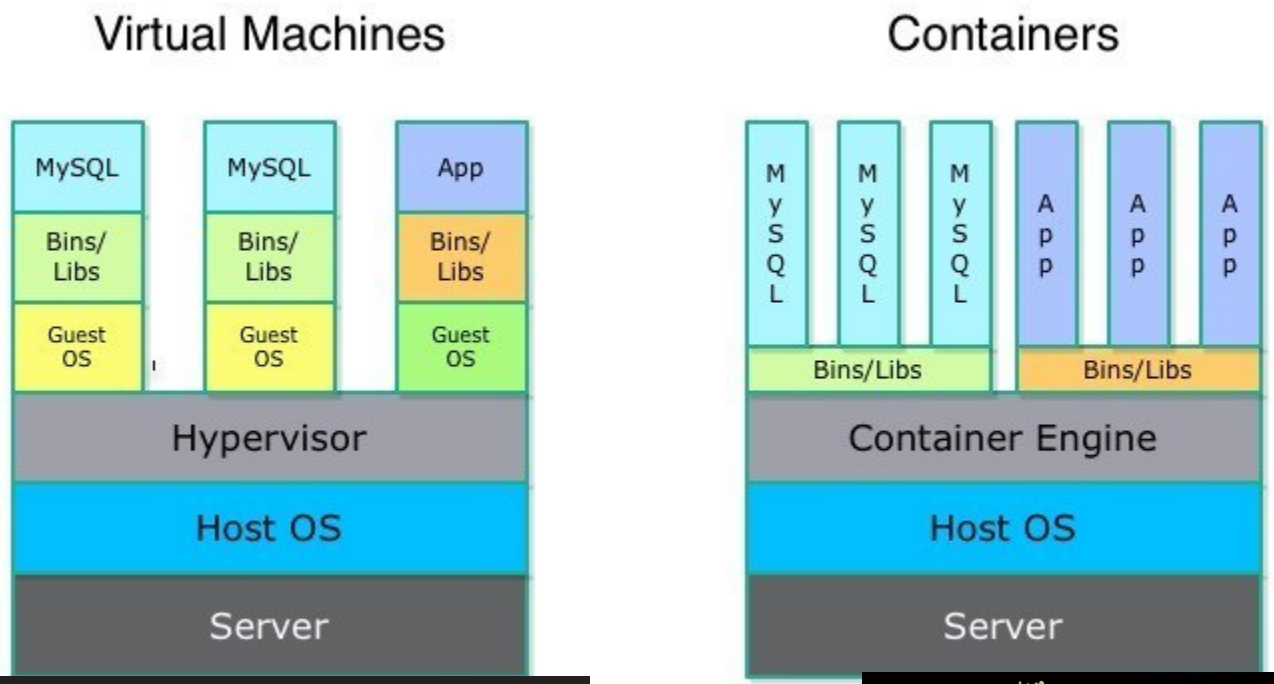
Before Containers

After Containers

Virtual Machines vs Containers

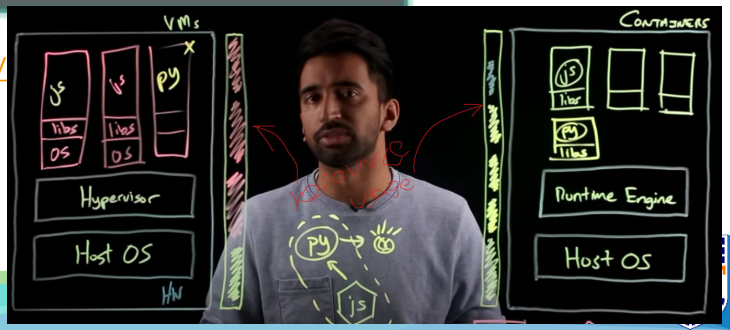
- Why is it containers, why not VMs?

  can gain compatibility issue



Feature	Virtual Machines (VMs)	Containers
Virtualization	Hardware-level virtualization	OS-level virtualization
Size	Large (includes full OS)	Lightweight (shares host OS kernel)
Boot Time	Slow (minutes)	Fast (seconds)
Isolation	Strong, full OS isolation	Process-level isolation via namespaces/cgroups
Resource Usage	More memory & CPU	More efficient and scalable

ntent/uploads/2015/



Virtual Machines vs Containers

- Containerization introduction: watch this video.
- When to use which? watch this video.

- To summarize,
  - virtual machines provide **Hardware (H/W) level virtualization**.
  - Containers provide **Operating System (O/S) level virtualization**.

Containers

Why choose containers?

- Traditionally, code was developed in an environment with specific configurations, outside of which the application is not guaranteed to perform as expected.

problem in traditional

- Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run.

soultion in containers

Environment Consistency -
Avoids "it works on my machine" problems by bundling everything the app needs.

Containers

Why choose containers?

Runs on any system (cloud, local, server) without changes.

- This single package of software or "container" is abstracted away from the host operating system hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.
- Containers are 'lightweight' (compared to VMs) – they share the machine's operating system kernel (therefore, resource efficient and faster).
- Containerization allows applications to be "written once and run anywhere."

Microservices in containers

- A microservice, developed within a container, gains the inherent benefits of containerization;
  - Portability and no vendor lock-in
  - Developer agility
  - Fault isolation
  - Efficient resource utilization
  - Automation of installation, scaling and management
  - More layers of security

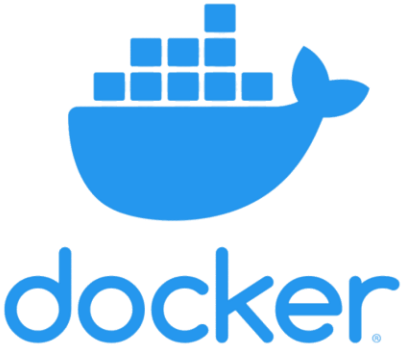
Containers – Underlying technology

- Two Linux concepts - cgroups and namespaces together form the basis of containerization technologies.
  - **Namespaces** provide process isolation, allowing process/es to run independently of each other and the host system.
    Provide process isolation (like file system, network, users, etc.)
    Each container sees its own private environment
  - **Cgroups** provide resource management and isolation for a collection of processes (i.e. a namespace), ensuring that these processes do not consume excessive resources.
    Control groups
    Manage and limit CPU, memory, I/O usage for containers
    Prevent one container from hogging resources

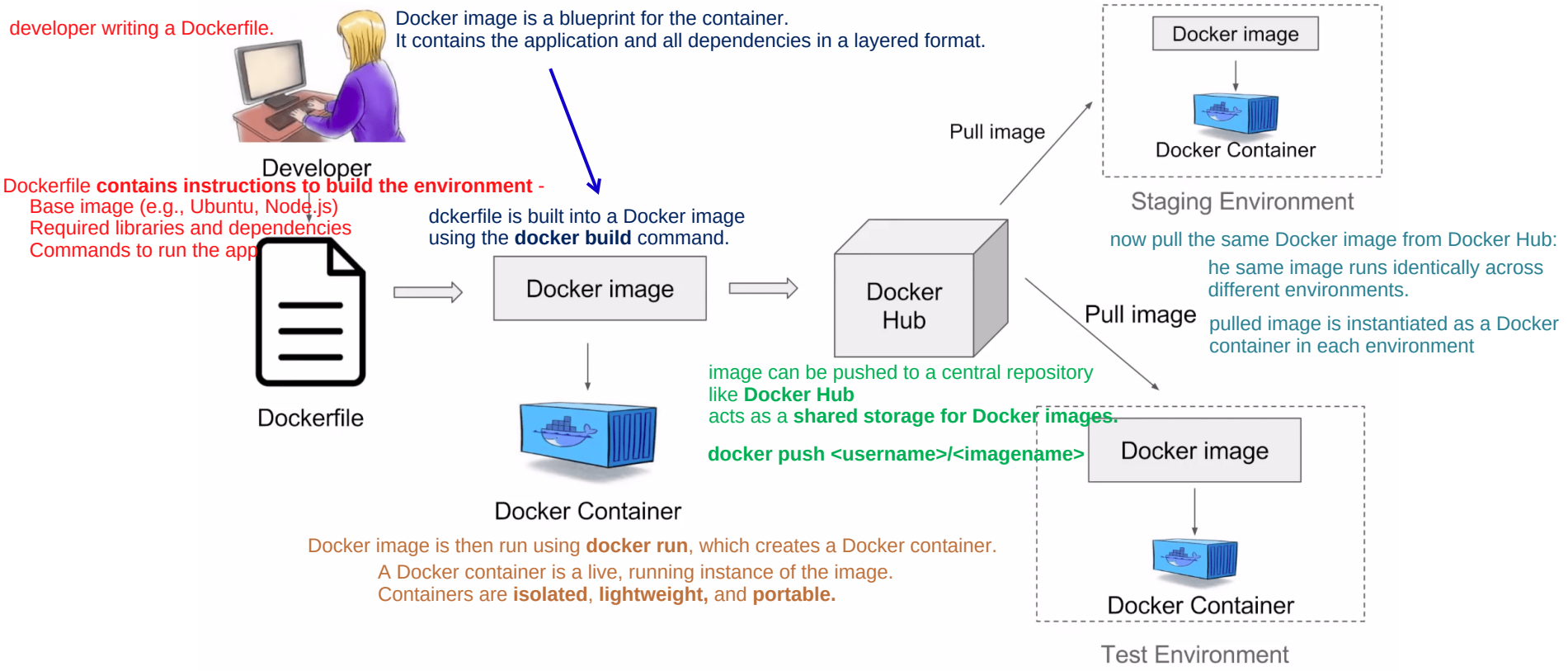
s video discusses these two concepts briefly.

Docker Overview

- Docker is the current Industry standard for containerization.
  - Docker introduction: watch this video.



Docker Overview



Source: https://k21academy.com/docker-kubernetes/docker-and-kubernetes/

Container Orchestration

- Container orchestration is comparable to the role of the conductor of an orchestra.

Source: https://www.musicgateway.com/blog/music-theory/music-conductor

Container Orchestration

- Why is it needed?
  - In small numbers, one can manually deploy and manage containers.
  - But what would happen in an enterprise scenario, where millions of customers are constantly requesting for different types of services?
    Eg: Google search?
  - It is impossible to handle this load manually. Automation is needed.
  - Container **Orchestration** is the solution.

Without orchestration, scaling and maintaining containers would be inefficient, error-prone, and resource-draining.

Container Orchestration

- How does it work?
  - Usually, a developer writes a configuration file which is understood by the Orchestration tool.
  - This configuration file defines the **desired state** the containerized application (i.e. the containers which make it up) should be in.
  - The orchestration tool, based on this configuration file, maintains the state of the containers to resemble the desired state.
  - It manages the deployment of the containers, resiliency, selection of the host (to deploy the containers).
  - Once deployed, it manages scalability, availability and performance and even collecting and logging data for later review.

Kubernetes Overview

- Kubernetes is the current Industry standard for container orchestration.
  - Kubernetes quick intro: watch this video.
  - Kubernetes in 5 mins: watch this video.

kubernetes

How Does It Work?

1. **Configuration File**
   - A developer creates a file (e.g., `YAML` for Kubernetes) describing the **desired state**:
     - Number of container instances
     - What images to use
     - Networking rules
     - Resource limits
     - Scaling rules
2. **Orchestration Tool Reads the File**
   - Tools like **Kubernetes**, **Docker Swarm**, or **Apache Mesos** use the file to:
     - Deploy containers
     - Monitor their health
     - Reschedule failed containers
     - Scale up/down based on traffic
3. **Maintains Desired State**
   - If a container crashes or a node fails, the orchestrator **automatically recovers** to ensure the application remains in its expected state.
4. **Other Capabilities**
   - **Load balancing**
   - **Rolling updates**
   - **Secrets management**
   - **Service discovery**
   - **Logging and monitoring integration**

How Container Orchestration works

ment

Acknowledgements and Additional Reading

- What is Cloud Native Architecture?
- What is DevOps?
- What Are CI/CD and the CI/CD Pipeline?
- Containerization explained.
- What are Microservices?
- What is Container Orchestration?