[2024] Java Spring Boot Microservices with k8s, Docker, AWS | Monolithic to Microservices [PART 1] | Faisal Memon | Emba X.com

Classification of Status Codes

→ 1xx (Informational)

→ 2xx (Successful)

→ 3xx (Redirection)

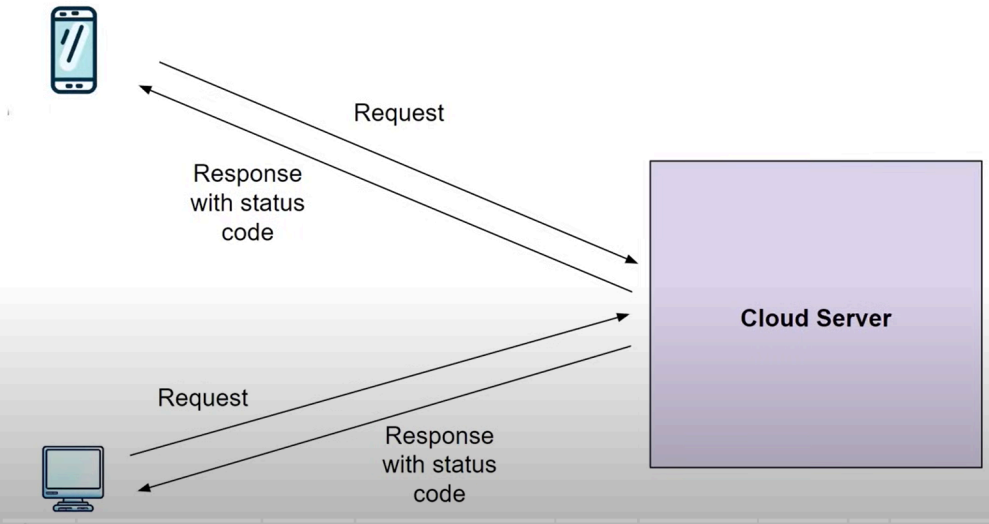
→ 4xx (Client Error)

→ 5xx (Server Error)

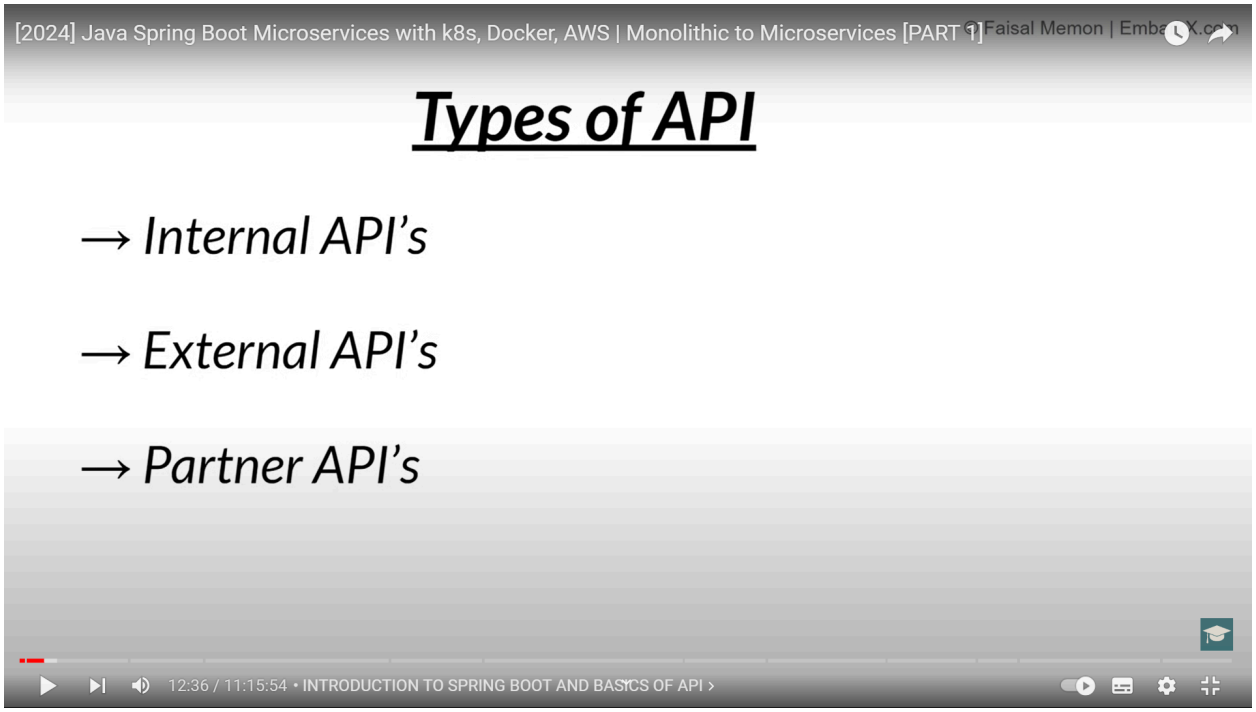
17:59 / 11:15:54 • INTRODUCTION TO SPRING BOOT AND BASICS OF API ›

[2024] Java Spring Boot Microservices with k8s, Docker, AWS | Monolithic to Microservices [PART 1] | Faisal Memon | Emba X.com

Need for Status Codes



15:14 / 11:15:54 • INTRODUCTION TO SPRING BOOT AND BASICS OF API ›



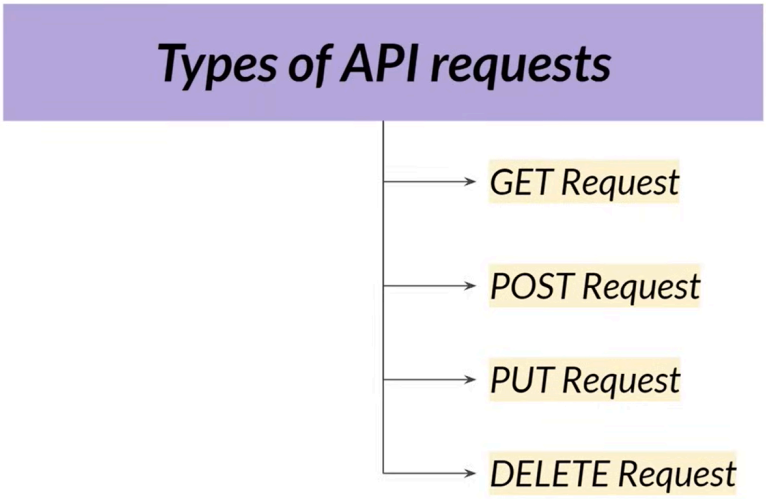
Commonly used Status Codes

→ 401 Unauthorized

→ 403 Forbidden

→ 404 Not Found

→ 500 Internal Server Error



[2024] Java Spring Boot Microservices with k8s, Docker, AWS | Monolithic to Microservices [PART 1]

GET Request

→ Retrieve or GET resources from server

→ Used only to read data

PUT Request

→ *Update existing resources on Server*

DELETE Request

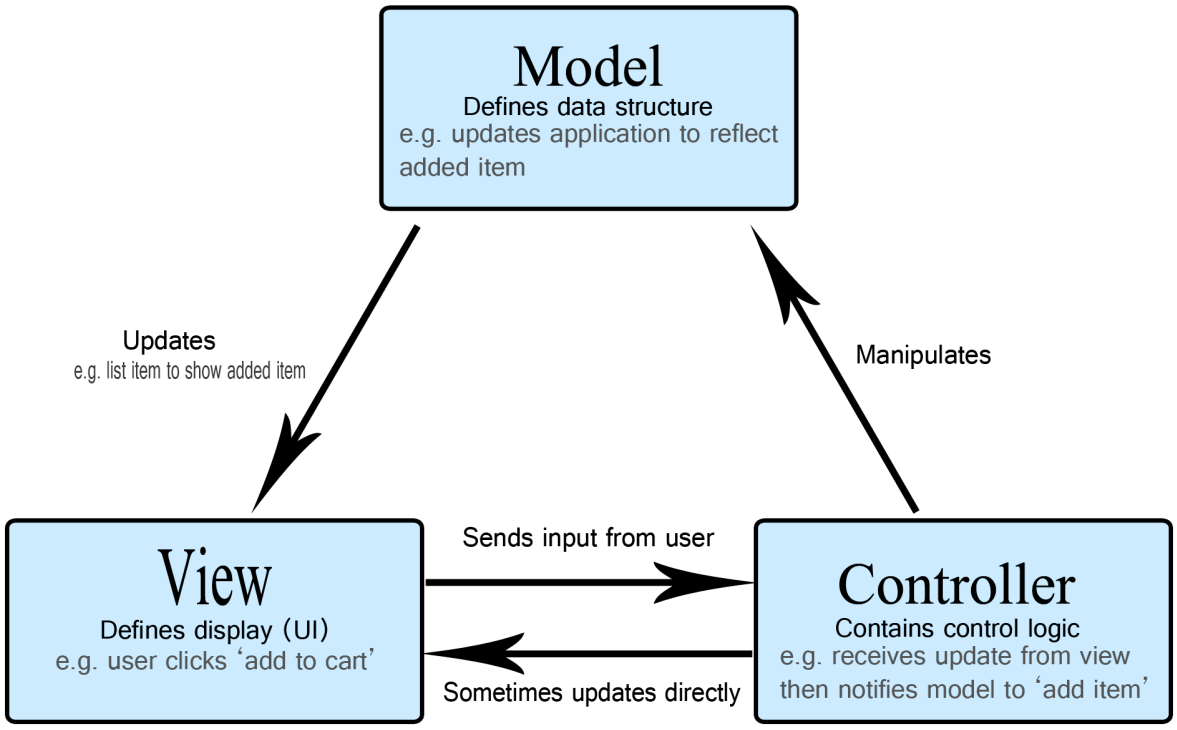
→ *Used to DELETE resources from Server*

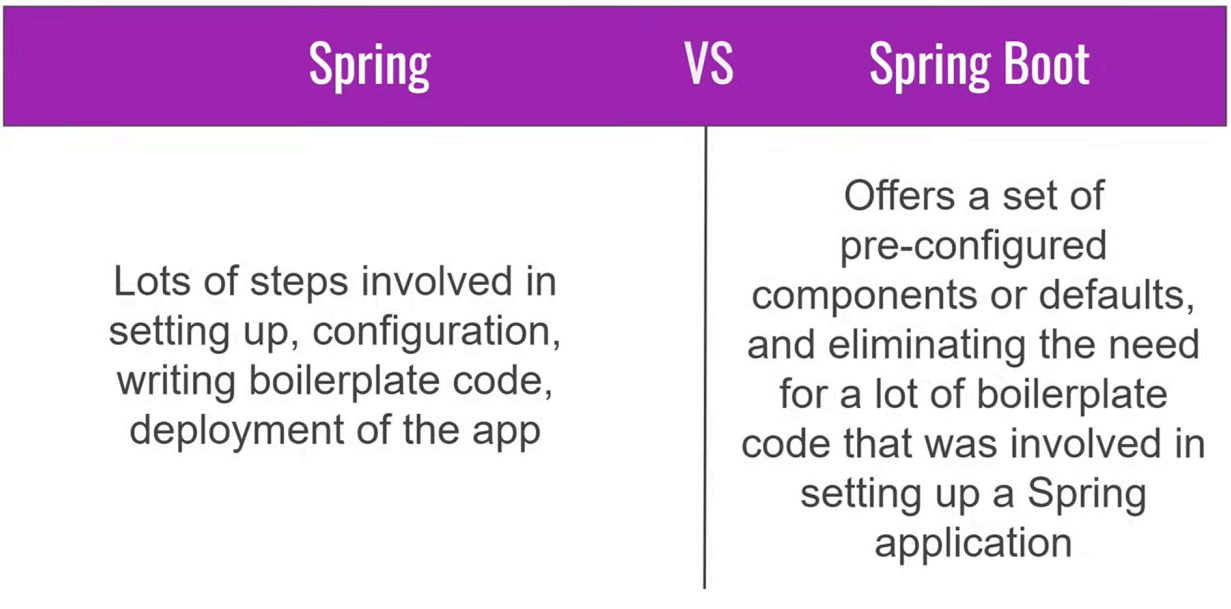
What is Web Framework

Web Framework is nothing but collection of tools and modules that is needed to do standard tasks across every web application.

Features of Spring Framework

- Inversion of Control (IoC)
- Data Access
- MVC Framework
- Transaction Management
- Security





Spring boot = Spring Framework + Prebuilt Configuration + Embedded Servers

---

Components of Spring Boot

→ *Spring Boot Starters*

→ *Auto Configuration*

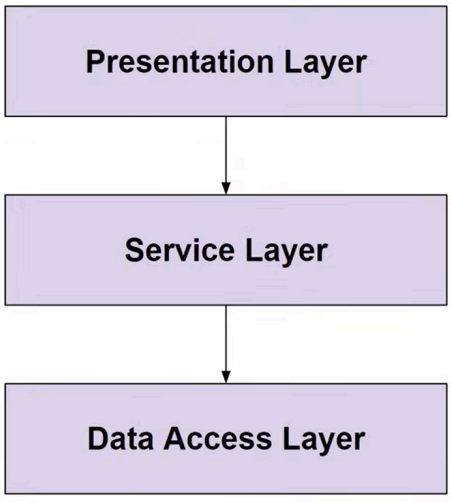
→ *Spring Boot Actuator*

→ *Embedded Server*

→ *Spring Boot DevTools*

Spring Boot Architecture

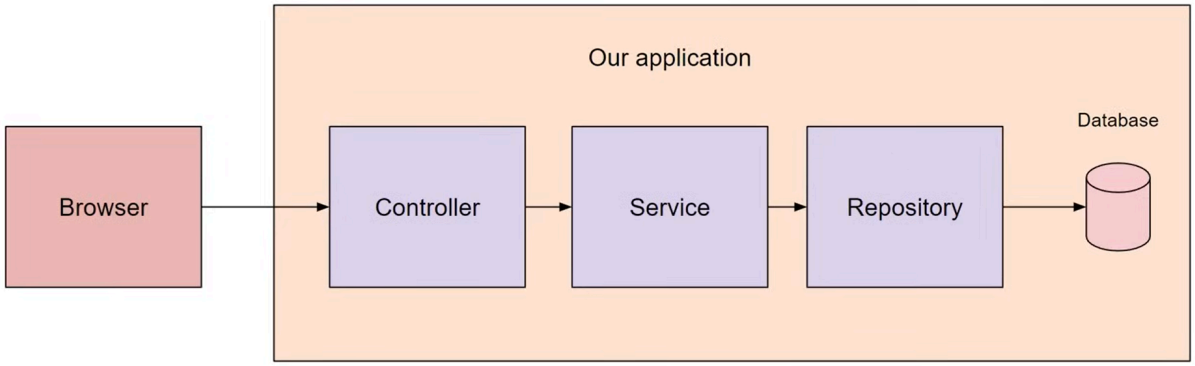
Faisal Memon (EmbarkX)



Presentation layer presents the data and the application features to the user. This is the layer where in all the controller classes exist.

Service layer is where business logic resides in the application. Tasks such as evaluations, decision making, processing of data is done at this layer.

Data access layer is the layer where all the repository classes reside.



Approaching Builds: Gradlevs. Maven

Gradle and Maven fundamentally differ in their approach to builds. Gradle operates based on a graph of task dependencies, with tasks performing the work. Conversely, Maven adopts a fixed, linear model of phases, assigning goals to project phases. These goals, like Gradle's tasks, are the "workhorses."

Performance: Speed and Efficiency

Both Gradle and Maven support parallel execution of multi-module builds. Gradle, however, stands out for its use of incremental builds. It achieves this by checking the status of tasks and skipping any that aren't updated, resulting in shorter build times. Gradle enhances performance with the following features:

- Incremental compilations for Java classes
- Compile avoidance for Java
- APIs for incremental subtasks
- A compiler daemon for faster compiling

Dependency Management: Flexibility and Compatibility

Both Gradle and Maven excel at handling dynamic and transitive dependencies, using third-party dependency caches, and reading POM metadata format. They can also declare library versions through central versioning definition and enforce it. Each can download transitive dependencies from their artifact repositories, Maven from Maven Central, and Gradle from JCenter. Both support the definition of a private company repository. If a project requires multiple dependencies, Maven can download these concurrently.

Despite these similarities, Gradle outperforms Maven in areas like API and implementation dependencies, and concurrent safe caches. Gradle preserves repository metadata with cached dependencies, preventing overwrites when multiple projects use the same cache. It also features a checksum-based cache and synchronizes the cache with the repository. Moreover, Gradle supports IVY Metadata, allowing custom rules for dynamic dependencies and resolving version conflicts, unlike Maven.

Latest Posts

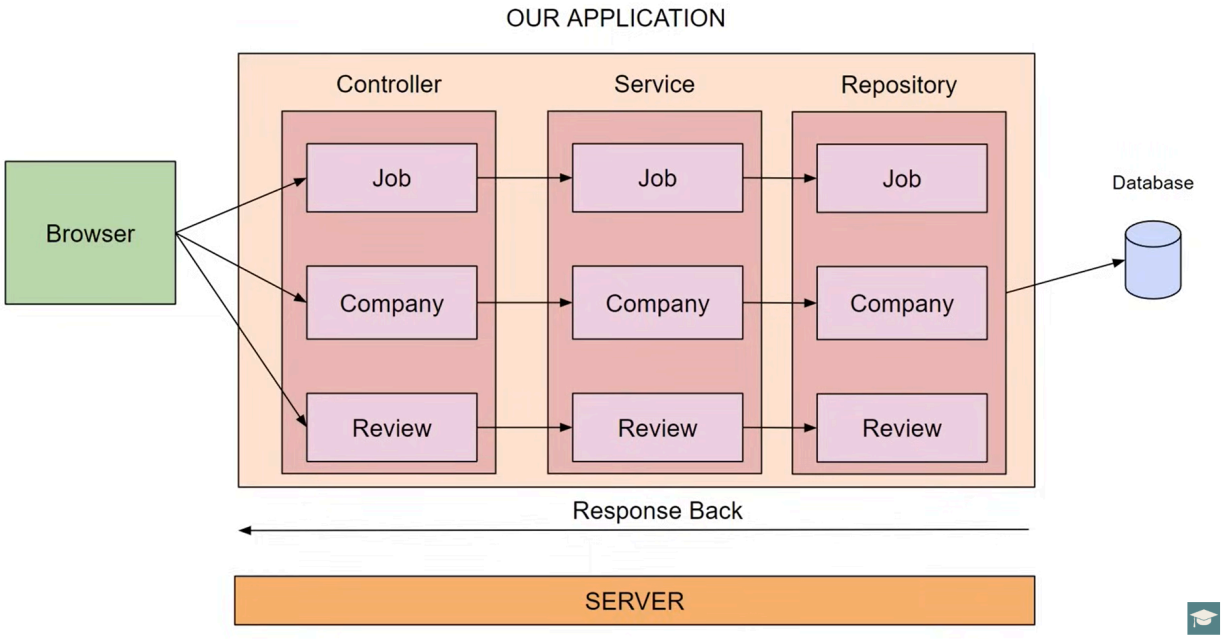

Top Linux Commands: 11 You Need to Know


Developer's Guide to Getting Started with Pandas Profiling


What Is Python Logging? A Complete Introduction


Uses for Java Programming: 10 Reasons to Learn the Language


Syntax Errors in Python: A Complete Explanation

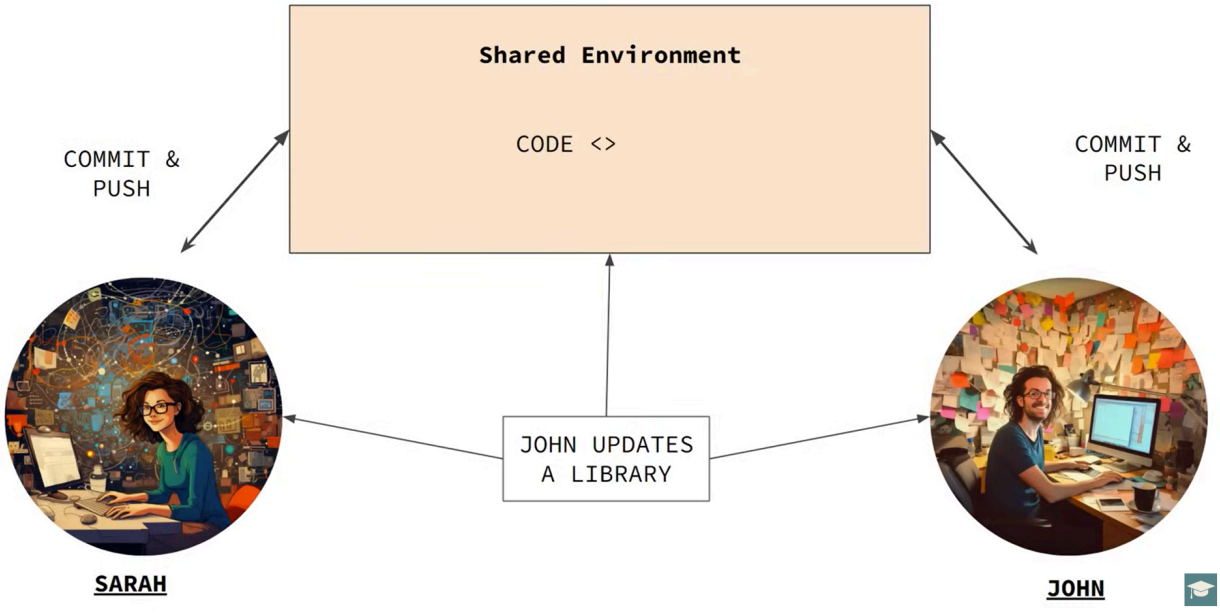


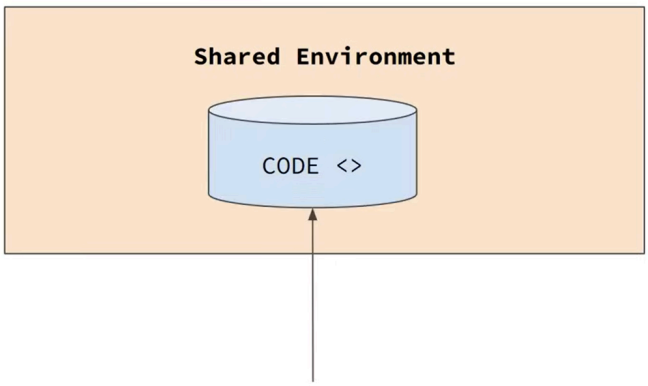
© Faisal Memon | EmbarkX.com

What is it?

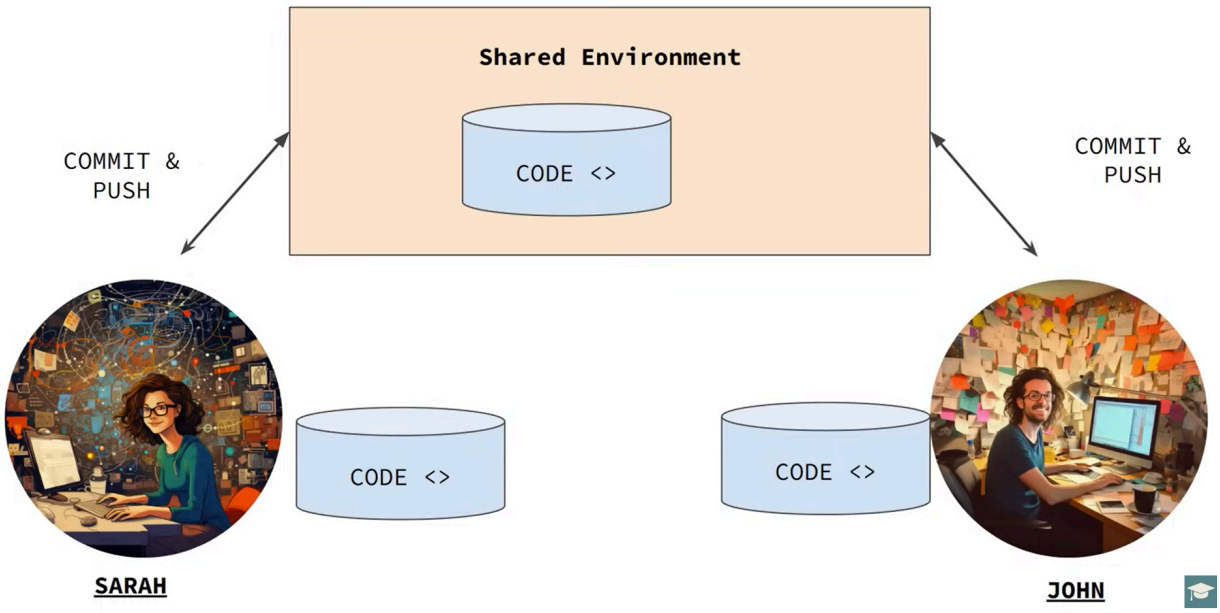
Provides built-in production-ready features to monitor and manage your application

Endpoint	Purpose
/health	Shows application health information, useful for checking the status of the application, such as database connectivity, disk space, and custom health checks.
/info	Displays arbitrary application information, commonly used to display application version, git commit information, etc.
/metrics	Shows 'metrics' information that allows you to understand the performance and behavior of your running application.
/loggers	Allows you to query and modify the logging level of your application's loggers.
/beans	Provides a complete list of all the Spring beans in your application
/shutdown	Allows your application to be gracefully shut down





Includes the **application code**, its **dependencies**, and the required **environment configuration**

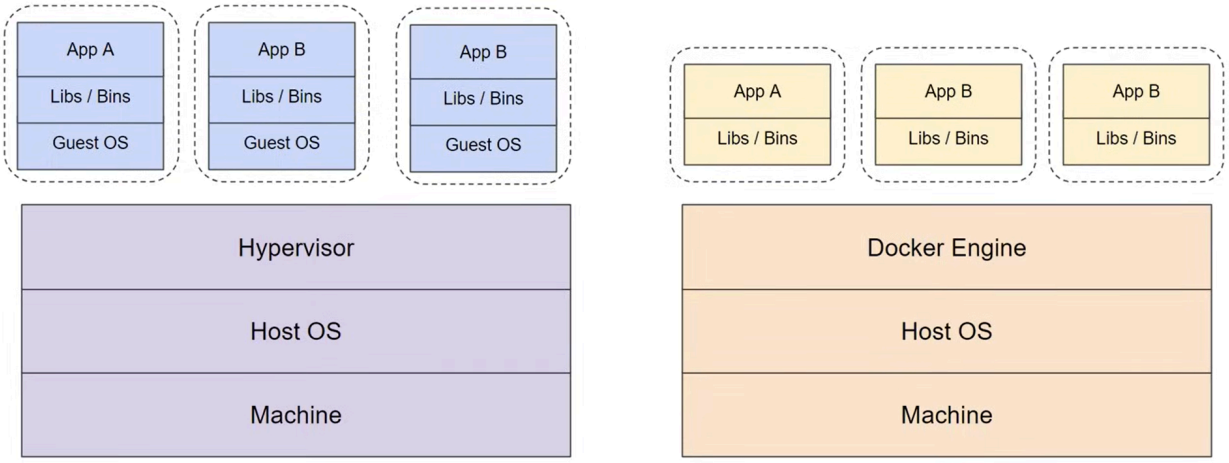


Let's Define Docker

Docker is an open-source platform that allows you to automate the deployment, scaling, and management of applications using containerization

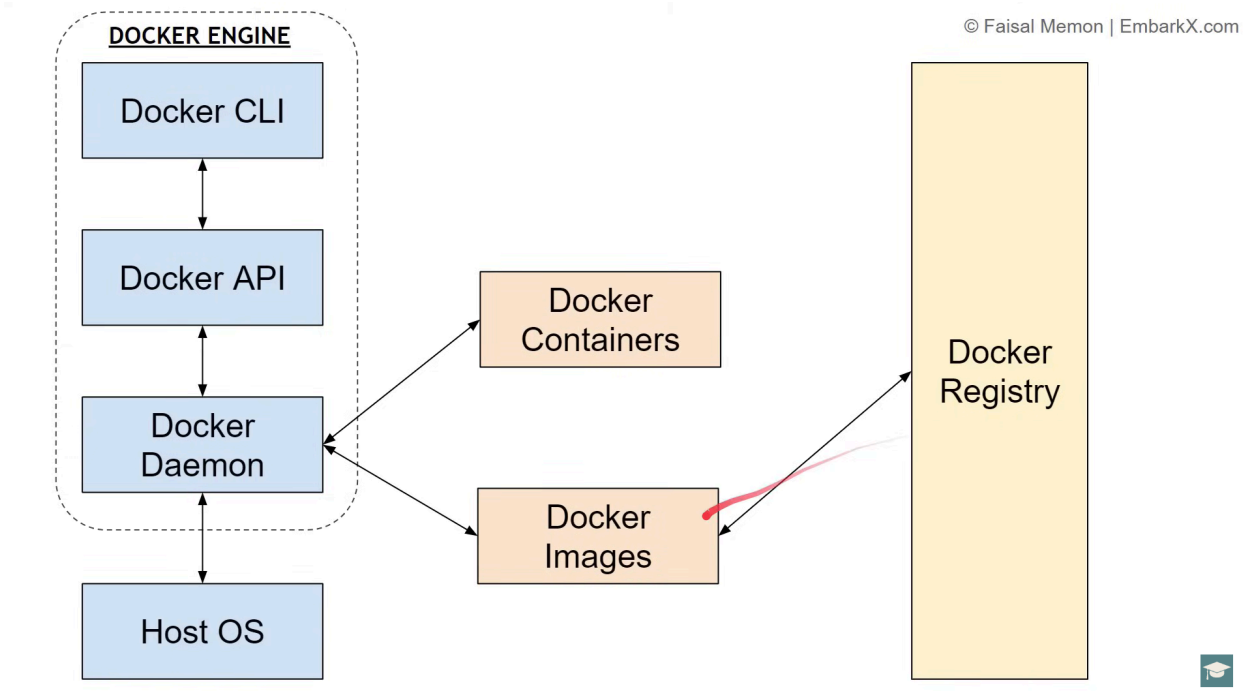
---

Docker over VM's



Parameters	Virtual Machines (VMs)	Docker Containers
Size	Relatively large and resource-intensive	Lightweight and resource-efficient
Startup Time	Longer boot time as full OS needs to start	Almost instant startup as no OS boot required
Resource Utilization	Utilizes more system resources (CPU, memory)	Utilizes fewer system resources
Isolation	Strong isolation between VMs	Isolated, but shares host OS kernel
Portability	Portable, but requires OS compatibility	Highly portable, independent of host OS

Parameters	Virtual Machines (VMs)	Docker Containers
Scalability	Scaling requires provisioning of new VMs	Easy to scale by creating more containers
Ecosystem	VM-specific tools and management frameworks	Docker ecosystem with extensive tooling
Development Workflow	Slower setup and provisioning process	Faster setup and dependency management
Deployment Efficiency	More overhead due to larger VM size	Efficient deployment with smaller container



© Faisal Memon | EmbarkX.com

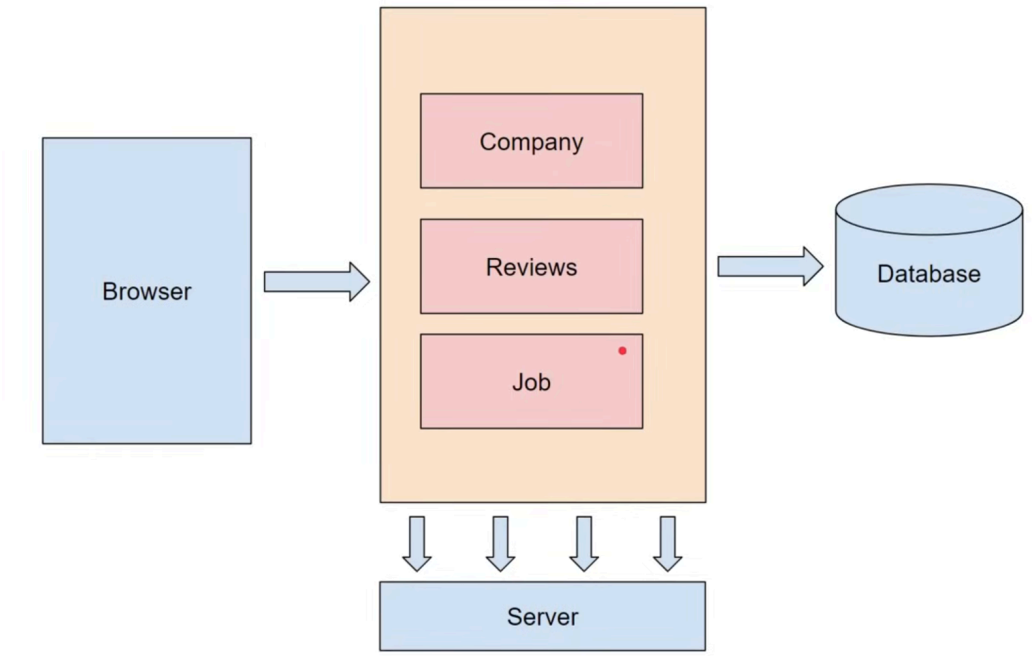
→ ***Images***: *Docker images are templates that define the container and its dependencies.*

→ ***Containers***: *Containers are runtime environments created from Docker images.*

→ ***Docker Engine***: *The Docker Engine is the runtime that runs and manages containers*

→ ***Dockerfile***: *A Dockerfile is a file that contains instructions to build a Docker image.*

→ ***Docker Hub***: *Docker Hub is a cloud-based registry that hosts a vast collection of Docker images*



Problems

→ *Difficult to Implement Changes*

→ *Lack of Scalability*

→ *Long-term Commitment to a Single Technology Stack*

→ *Application Complexity and Its Effect on Development and Deployment*

Problems

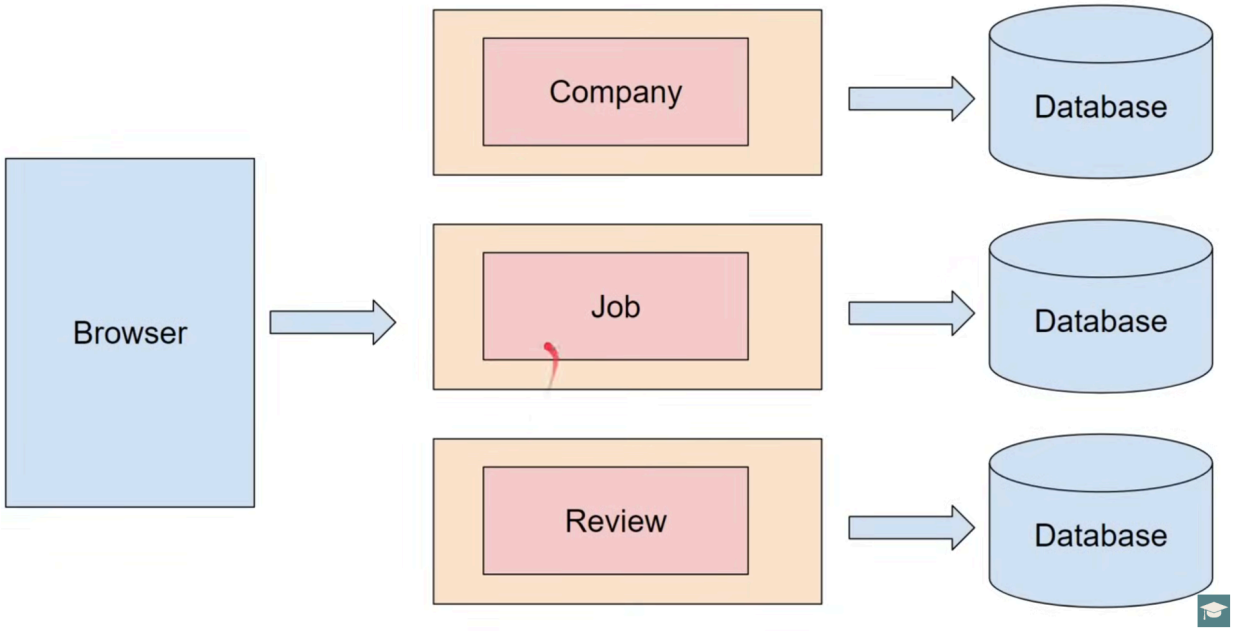
→ *Increased Application Start Time*

→ *Large Project Size*

→ *Deploying for Small Changes*

→ *Team Collaboration and Autonomy*

Microservices structures an application as a collection of small autonomous services



Principles of Microservices

→ *Single Responsibility*

→ *Independence*

→ *Decentralization*

→ *Failure Isolation*

→ *Continuous Delivery/Deployment*

Features and Advantages of RestTemplate

- *Abstraction*
- *Versatility*
- *Conversion*
- *Error Handling*
- *Integration*
