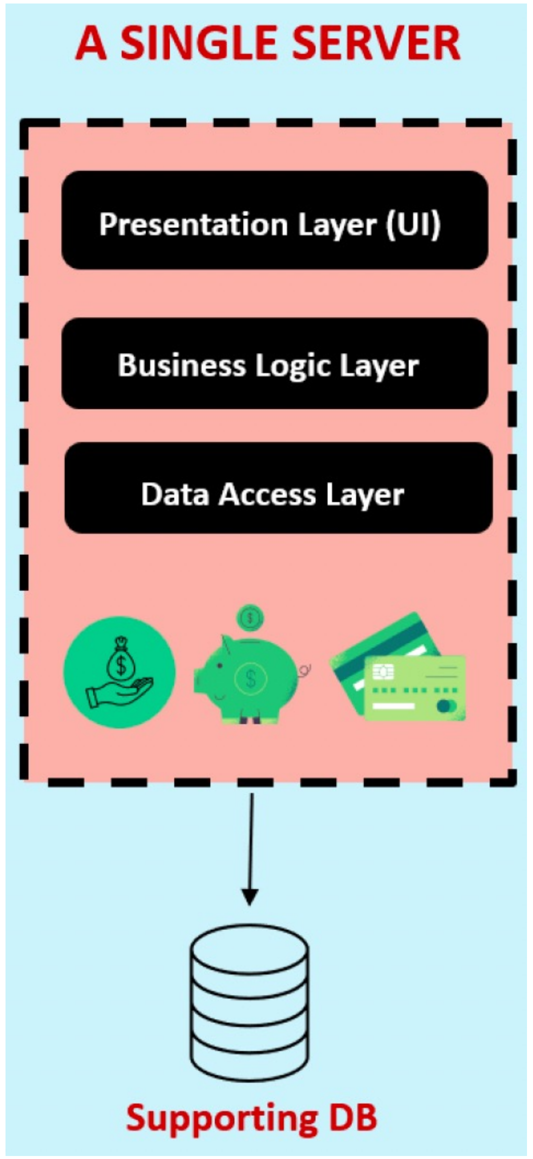
Microservices

The Monolith

- In a monolithic approach, developers work with single codebase, which is then packaged as a unified unit, such as an EAR/WAR file and deployed onto a single web/application server.
- Additionally, the entire application is supported by a single database



The Monolith

- Back a decade, all the applications used to be deployed as a single unit where all functionality deployed together inside a single server, we call this architecture approach as Monolithic.
- We have various forms of Monolithic with the names like Single Process Monolith, Modular Monolith, Distributed Monolith etc

Pros

- Simpler development and deployment for smaller team and application
- Fewer cross cutting concerns
- Better Performance due to no network latency

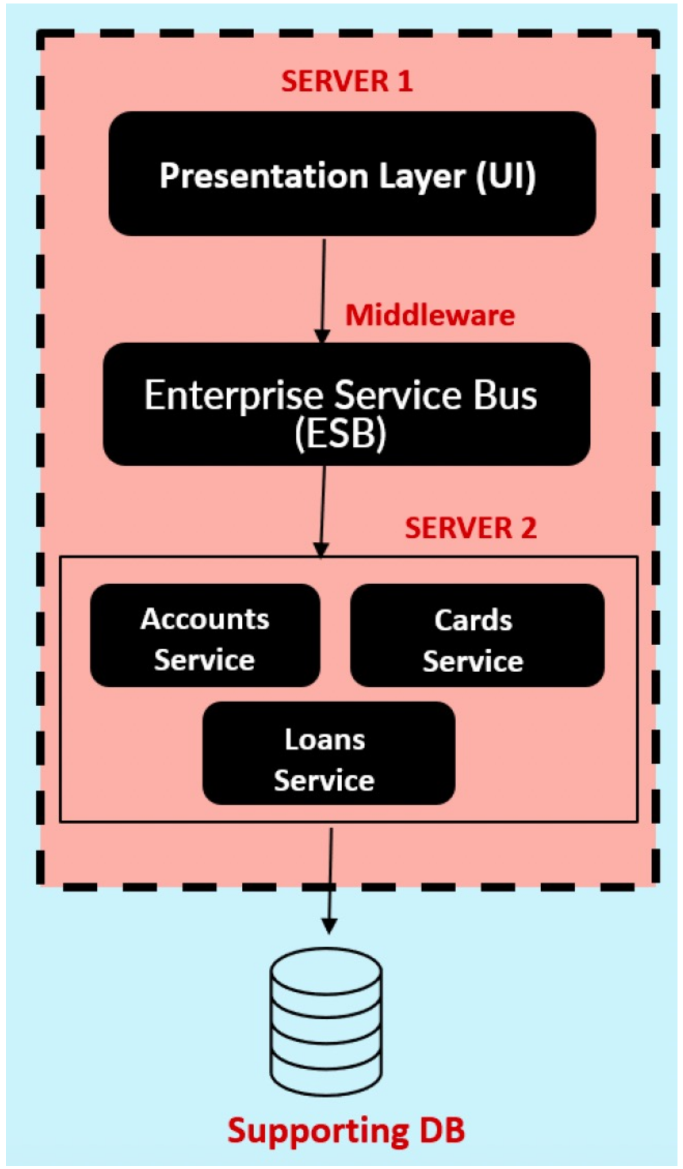
Cons

- Difficult to adopt new technology
- Limited Agility
- Single codebase and difficult to maintain
- Not fault tolerance
- Tiny updates need full deployment

The SOA

(Service Oriented Architecture)

- It is an architectural style that focus on organizing software system as a collection of loosely coupled, interoperable services.
- It provides a way to design and develop large scale applications by decomposing them into smaller, modular services that can be independently developed, managed and deployed.



The SOA

Pros

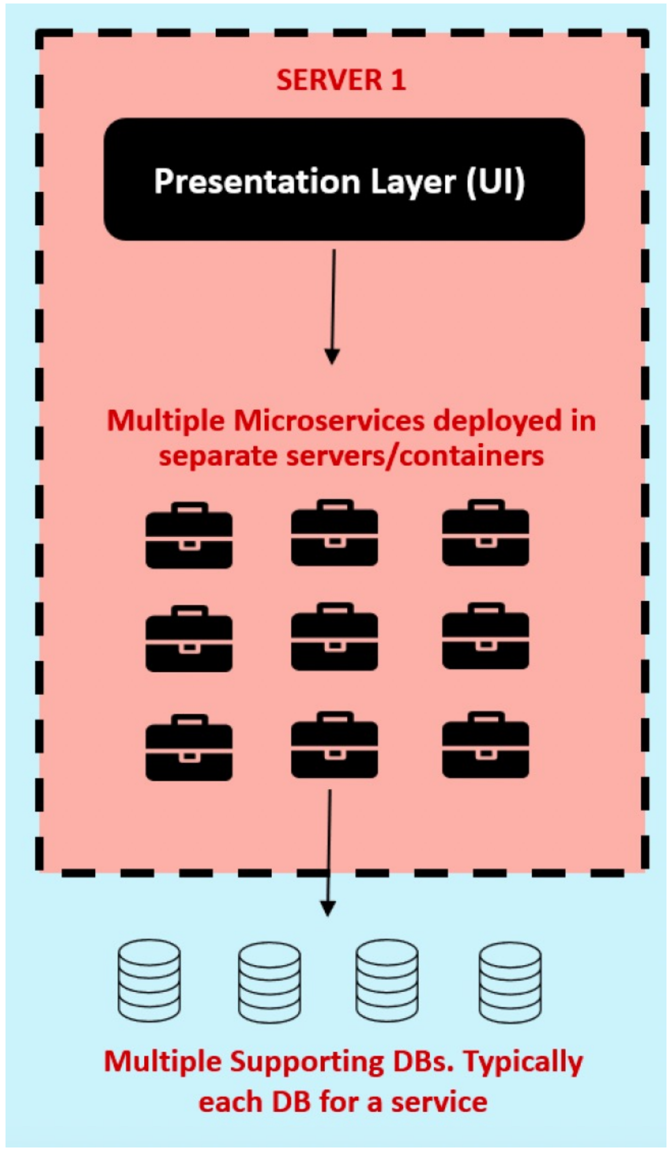
- Reusability of Services
- Better maintainability
- Higher reliability
- Parallel development

Cons

- Complex management due to communication protocol e.g. SOAP
- High investment costs due to vendor in middleware
- Extra overload

The Great Microservices

- Microservices are independently releasable services that are modeled around a business domain.
- A service encapsulate functionality and make it accessible to other services via networks


SERVER 1
Presentation Layer (UI)
Multiple Microservices deployed in separate servers/containers
Multiple Supporting DBs. Typically each DB for a service

The Great Microservices

Pros	Cons
• Easy to develop, test and deploy	• Complexity
• Increased agility	• Infrastructure overhead
• Ability to scale horizontally	• Security concerns
• Parallel development	
• Model around a business domain	

“ Microservice is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanism, built around business capabilities and independently deployable by fully automated deployment machinery ”

- From article by James Lewis and Martin Fowler’s

SpringBoot for Microservices

WHY SPRINGBOOT FOR MICROSERVICES?

SpringBoot is a framework that simplifies the development and deployment of Java applications, including microservices. With SpringBoot we can create self-contained, executable JAR files instead of traditional WAR / EAR files.

JAR Files contain all the dependencies and configuration required to run the microservice. The approach eliminate the need of external web server or application server.

WHY SPRINGBOOT IS THE BEST FRAMEWORK TO BUILD MICROSERVICE?

Provides a range of built- in feature and integrations such as auto-configuration, dependency injection and support for various cloud platforms

Provides an embedded Tomcat, Jetty or Undertow server, which can run microservice directly without the need for a separate server installation

In-built support for production ready features such as metrics, health check and externalized configuration

We can quickly bootstrap a microservice project and start coding with a range of stater dependencies such as database, queues etc

Well-suited for cloud native development. It integrates smoothly with cloud platforms like Kubernetes, provide support for containerization, seamless deployment etc

Different Annotations and Classes to build REST Services

@RestController – can be used to put on top of a call. This will expose your methods as REST APIs

ResponseEntity<T> - allow developer to send response body, status and headers on the Http response

RequestEntity<T> Allow developers to receive the request body, header in an Http request

@ControllerAdvice – is used to mark the class as REST Controller Advice, Along with @ExceptionHandler, this can be used to handle exceptions globally in App

@RequestHeader and @RequestBody – is used to receive the request header and body individually

Summary of steps to build Microservices

Build empty SpringBoot App

- First we create empty SpringBoot App with the required stater dependencies related to Web, Actuator, JPA, DevTools, Validation, H2DB, Lombok, SpringDoc Open API etc

Build DB related logic, entities and DTO

- Create required DB tables schema, establish connection details with H2 DB
- Create JPA entities, repositories
- Create DTO Classes and Mapper logic

Build business logic

- Create REST API supporting CRUD operation with the help of various annotations like @GetMapping, @PostMapping, @PutMapping, @DeleteMapping etc

Build global exception handling logic

- Build global exception handling logic using @ControllerAdvice and @ExceptionHandler
- Also create custom business exceptions

Summary of steps to build Microservices

Perform data validations on the input

- Perform validation on the input data using annotation present inside Jakarta.validation package.
- These annotations are like @NotEmpty, @Size, @Email, @Patter, @Validated, @Valid etc

Perform auditing using Spring Data JPA

- With the help of annotations like @CreatedAt, @CreatedBy, @LastModifiedAt, @LastModifiedBy, @EntityListeners & @EnableJPAAutditing
- Implement logic to populate audit fields in DB

Documenting REST API

- With the help of OpenAPI specifications, Swagger, SpringDoc library
- In the same process, use annotations like @Schema, @Tag, @Operation, @ApiResponse etc

Cloud Native Application

“Cloud native technologies empowered organizationsto build and run scalable application in modern, dynamic environment such as public, private and hybrid cloud. Containers, Service meshes, microservices, immutable infrastructure and declarative API exemplify this approach.”

- The Cloud Native Computing Foundation Definition

Cloud Native Applications – in simple words

Cloud native applications are software applications design specifically to leverage cloud computing principles and take full advantage of cloud-native technologies and services

These application are build and optimized to run in cloud environments, utilizing the scalability, and flexibility offered by the cloud.

Important Characteristics Of Cloud Native Applications

Microservices

Often build using a microservice architecture, where the application is broken down into smaller, loosely coupled services that can be developed, deployed and scale independently

Containers

Typically packaged and deployed using containers, such as docker containers.

Scalability and Elasticity

Designed to scale horizontally, allowing them to handle increased loads by adding more instances of the services. They can also automatically scale up and down based on demand. *(thanks to K8s)*

Important Characteristics Of Cloud Native Applications

DevOps Practice

Embrace DevOps principles, promoting collaboration between development and operations teams

Resilience And Fault-tolerance

Design to be resilient in the face of failure. They utilized techniques such as distributed architecture, load balancing and automated failure discovery to ensure high availability and fault tolerance

Cloud-native Services

Take advantage of cloud-native services provided by cloud platform such as managed databases, messaging queues, caching systems, and identity services

Development Principles Of Cloud Native
(12 Factors And Beyond)

- On Codebase, One Application
- API First
- Dependency Management
- Design, Build, Release, Run
- Configuration, Credentials And Code
- Logs
- Disposibility
- Backing Service
- Environment Parity
- Administrative Processes
- Port Binding
- Stateless Processes
- Concurrency
- Telemetry
- Authentication And Authorization

Spring Cloud

What is Spring Cloud?

Spring cloud provides frameworks for developers to quickly build some of the common pattern of microservices.

- Spring Cloud Config
- Service Registration and Discovery
- Routing and Tracing
- Load Balancing
- Spring Cloud Security
- Distributed tracing and messaging

Spring Cloud Config

Externalized Configurations

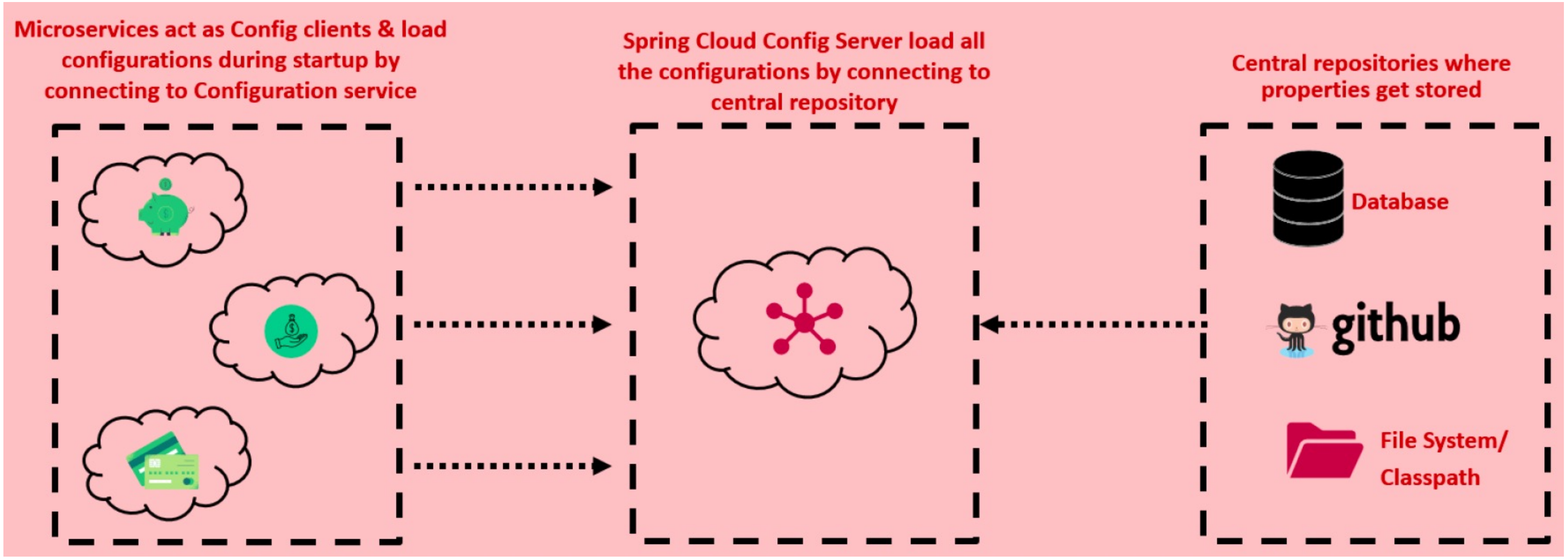
Spring Cloud Config

Spring cloud config provides server and client side support to externalized configuration in a distributed environment. With the config server you have a central place to manage external properties fo applications across all environments.

Centralized configuration resolve around two main elements -

- A data store designed to handle configuration data, ensuring durability, version management and potentially access control
- A server that oversees the configuration data within the data store, facilitating its management and distribution to multiple applications

Spring Cloud Config - Explanation


Microservices act as Config clients & load configurations during startup by connecting to Configuration service
Spring Cloud Config Server load all the configurations by connecting to central repository
Central repositories where properties get stored
Database
github
File System/ Classpath

Service Registration & Service Discovery

Service Registration & Service Discovery

Service Discovery involves in tracking and storing information about all the running instances of services in a Service Registry.

Whenever a new instance is created, it is registered with the registry, and when it is terminated, it should be appropriately removed from the registry automatically.

The registry acknowledges that multiple instances of same application can be active simultaneously.

When an application needs to communicate with a backing service, it performs a lookup in the registry to determine the IP address to connect to.

Client side service discovery and server-side service discovery are distinct approaches that address the service discovery problem in different contexts.

Service Registration & Service Discovery

Service discovery and registration is a way for application and microservices to locate each other on a network. This involves

- A central server that maintains a global view of addresses
- Microservices that connect to the central server to register their address when they start & ready
- Microservices need to send their heartbeats at regular interval to central server about their health
- Microservices that connect to the central server to deregister their address when they are about to shut down.

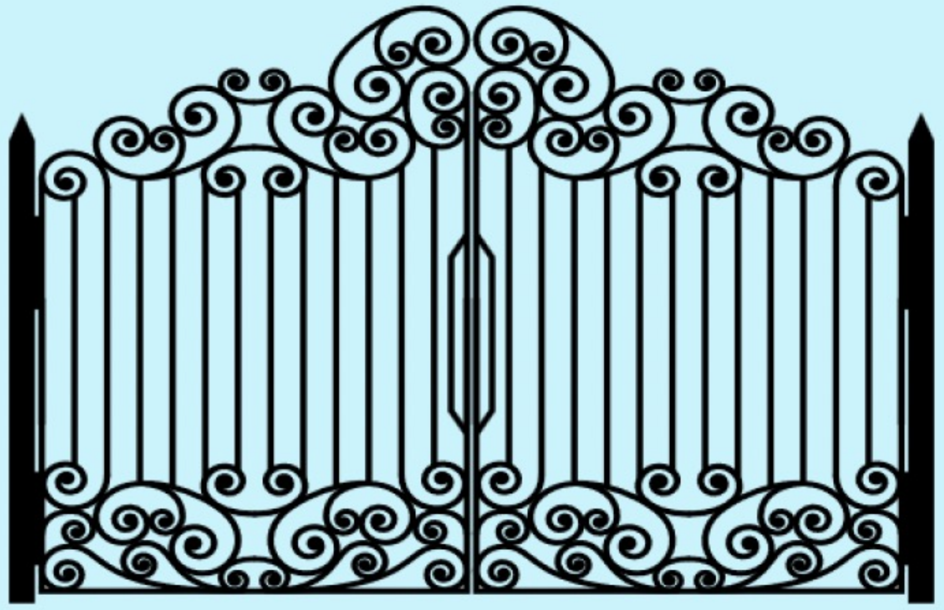
Spring Cloud Gateway

Spring Cloud Gateway

Spring Cloud Gateway is a library for building an API gateway, so it looks like any other SpringBoot application

Spring Cloud Gateway streamlines the creation of edge services by emphasizing ease and efficiency

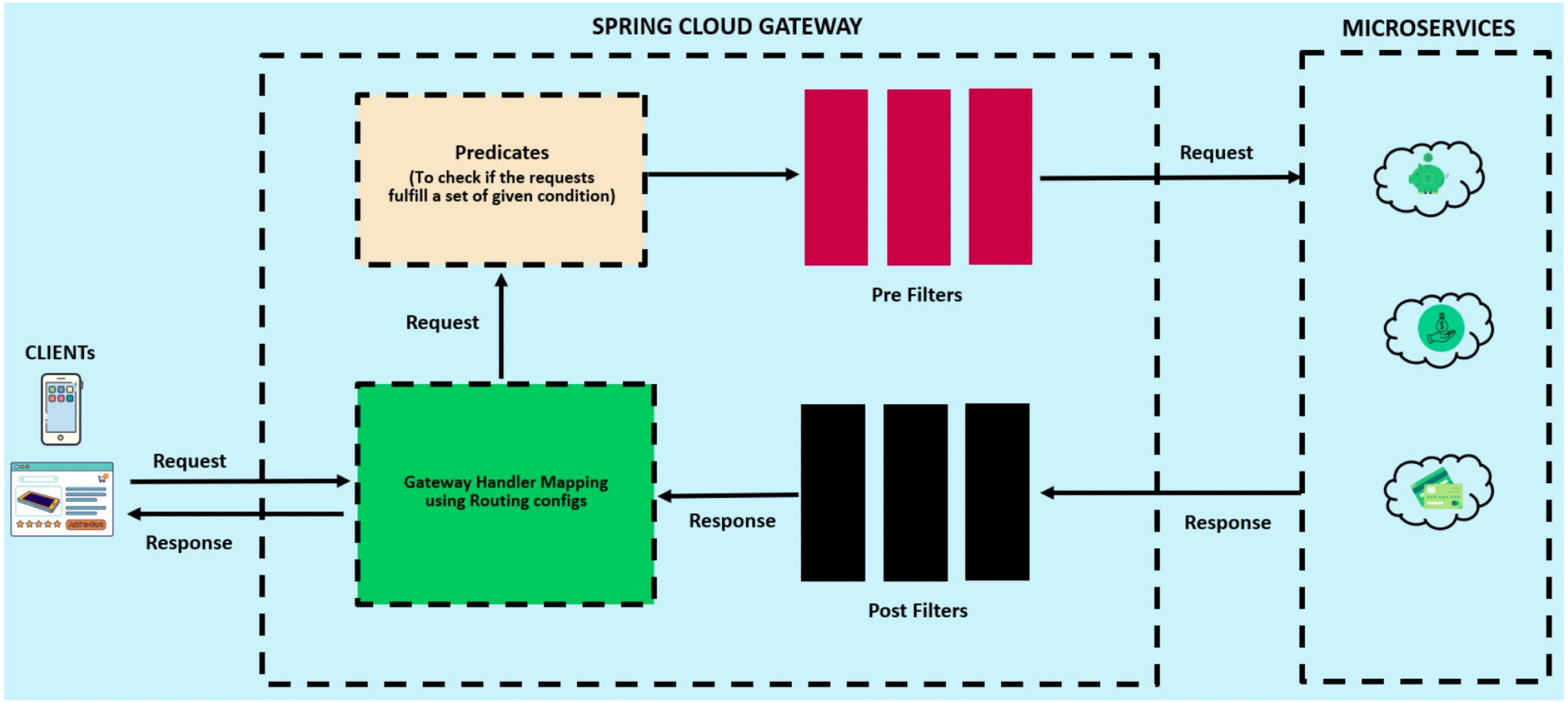
The service gateway sits as the gatekeeper to all inbound traffic to microservices calls within the application



The service gateway sits between all calls from the client to the individual services & acts as a central Policy Enforcement Point (PEP) like below,

- Routing (Both Static & Dynamic)
- Security (Authentication & Authorization)
- Logging, Auditing and Metrics collection

Spring Cloud Gateway - Internal Architecture



Resiliency in Microservices

Resiliency using Resilience4J

Resilience4J is a light-weight fault tolerance designed for functional programming. It offers various patterns for increasing fault tolerance due to network problem or failure of any of the multiple microservices

Circuit Breaker – used to stop making requests when a service invoked is falling

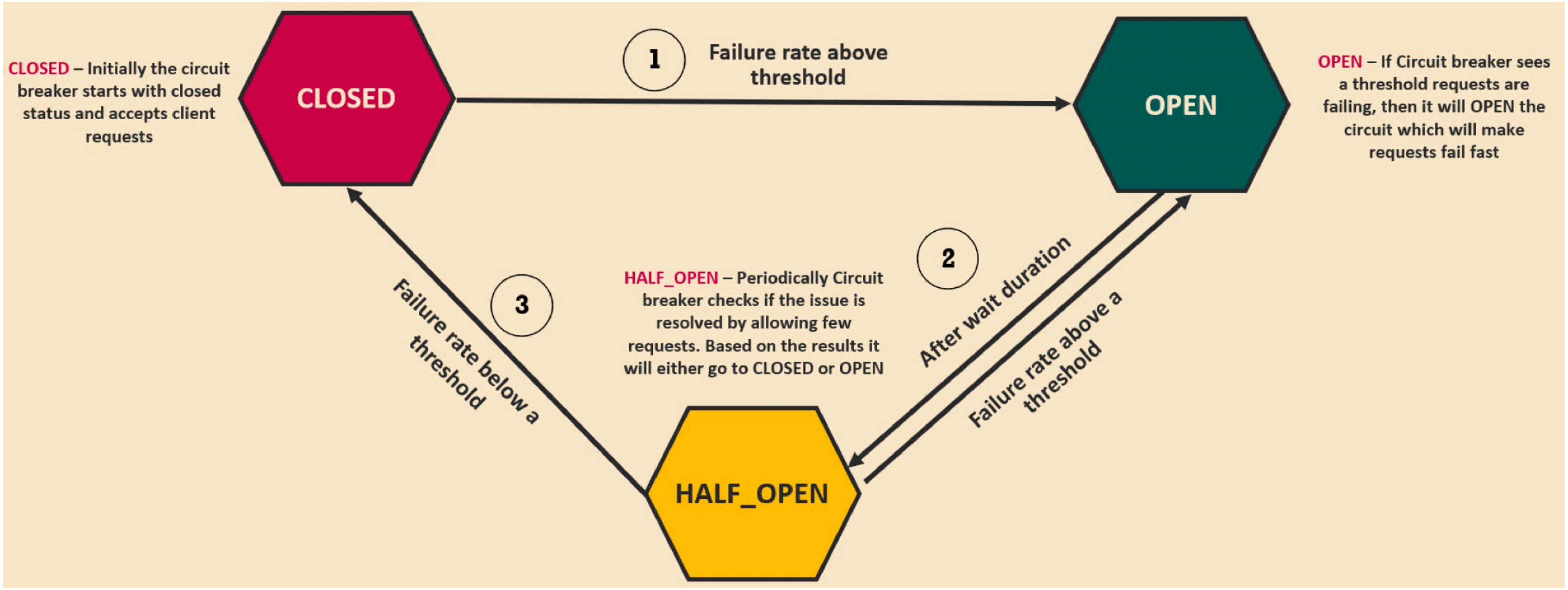
Fallback - Alternate path to falling requests

Retry – used to make retries when a service is temporarily available

Rate Limit – limit the number of calls that a service receives in a time

Bulkhead – limits the number of outgoing concurrent requests to a service to avoid overloading.

Circuit Breaker Pattern


CLOSED – Initially the circuit breaker starts with closed status and accepts client requests
CLOSED
1
Failure rate above threshold
OPEN
OPEN – If Circuit breaker sees a threshold requests are failing, then it will OPEN the circuit which will make requests fail fast
2
After wait duration
Failure rate above a threshold
3
Failure rate below a threshold
HALF_OPEN – Periodically Circuit breaker checks if the issue is resolved by allowing few requests. Based on the results it will either go to CLOSED or OPEN
HALF_OPEN

Deployment, Portability and Scalability of Microservices

Challenge – Deployment, Portability and Scalability

Deployment

How to we deploy all the tiny 100s of microservices with less efforts, configuration and cost?

Portability

How do we move our 100s of microservices across environments with less configuration and cost?

Scalability

How do we scale our application based on the demand on the fly with minimum efforts and cost?

Containerizing Microservices

- Container offers a self-contained and isolated environment for application including all necessary dependencies
- By containerizing the application, it becomes portable and can run seamlessly in any cloud environment

Docker is an open-source platform that provides the ability to package and run the application in a loosely isolated environment called a Container

What are Containers and Docker?

Container

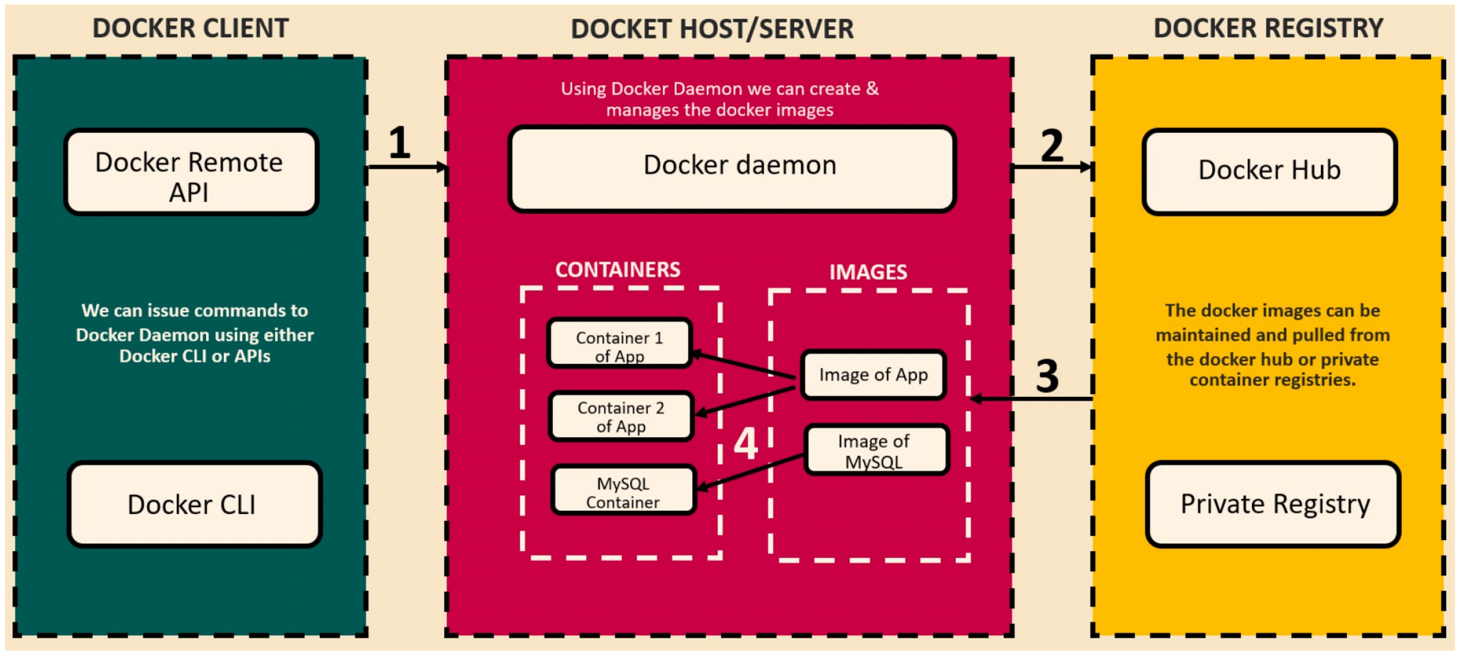
- A container is a loosely isolated environment that allow us to build and run software packages.
- These software packages includes all the dependencies to run application quickly and reliably on any computing environment. We call these packages as container images.

Docker

- Docker is an open source platform that enabled developers to automate the deployment, scaling and management of application using containerization.
- Containers are lightweight isolated environments that encapsulate an application along with its dependencies, libraries and runtime components.

Docker Architecture

1. Instruction from Docker client to server to run a container
2. Docker server finds the image in registry, if not found locally
3. Docker server pulls the image from registry into local
4. Docker server creates a running container from the image



DOCKER CLIENT
Docker Remote API
We can issue commands to Docker Daemon using either Docker CLI or APIs
Docker CLI
1
DOCKET HOST/SERVER
Using Docker Daemon we can create & manages the docker images
Docker daemon
CONTAINERS
Container 1 of App
Container 2 of App
MySQL Container
4
IMAGES
Image of App
Image of MySQL
2
3
DOCKER REGISTRY
Docker Hub
The docker images can be maintained and pulled from the docker hub or private container registries.
Private Registry

Docker Compose

What is Docker Compose?

- It is a tool provided by Docker that allow us to define and manage multi container applications.
- It uses YAML file to describe the services, networks and volume required for your application.
- Using this, we can easily specify the configuration and relationships between different containers
- Simple to setup and manage complex application environment

Advantages of Docker Compose

- By using single command, you can create and start all the containers defined in your docker compose file.
- Docker compose handles the orchestration and networking aspects, ensuring the containers can communicate with each other as specified in configuration.
- It also provides options for scaling services, controlling dependencies and manging the application life-cycles.

Important Docker Commands

Command	Description
docker images	to list all the docker images present in the docker server
docker image inspect [image_id]	to display detailed information about an image
docker image rm [image_id]	to remove image for the given image id
docker build . –t [image_name]	to generate docker image based on a docker file
docker run –p [host_port]:[container_port] [image_name]	to start docker container based on the given image
docker ps	to show all running container
docker ps –a	to show all running and stopped container
docker container start [container_id]	to start the stopped container
docker container pause [container_id]	to pause all process running in a container
docker container unpause [container_id]	to resume/unpause within container

Important Docker Commands

Command	Description
docker container stop [container_id]	to stop running container
docker container kill [container_id]	to kill running container instantly
docker container inspect [container_id]	to inspect all the details for the running container
docker container logs [container_id]	to fetch the logs of given container
docker rm [container_id]	to remove container based on container_id
docker container prune	to removed all stopped containers
docker image push [container_registry/username:tag]	to push an image to the container registry
docker image pull [container_registry/username:tag]	to pull an image from a container registry
docker system prune	remove stopped containers, dangling images, unused networks, volumes and cache
docker compose up | down	to start or remove containers for service defined I compose file

Thank You

Now you have solid foundation for the journey of Microservices