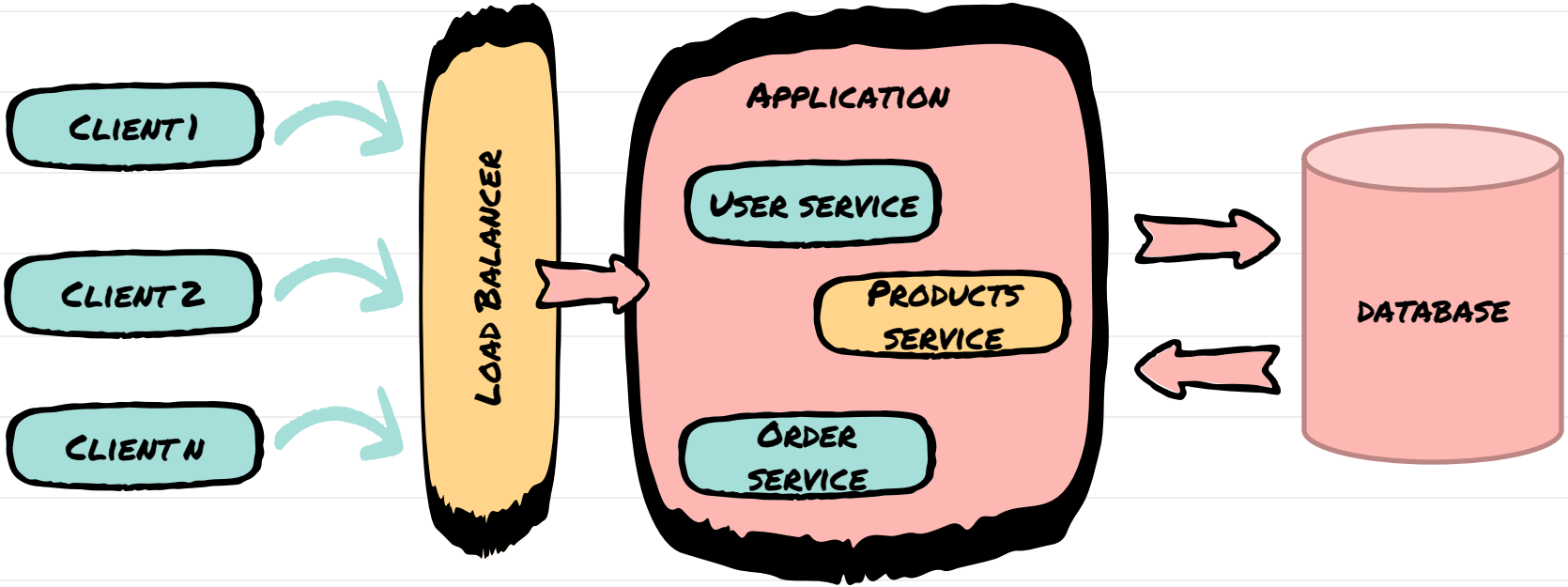
Microservices I

Teaching Faculty: Umur INAN

Prepared by Umur INAN

Monolith architecture



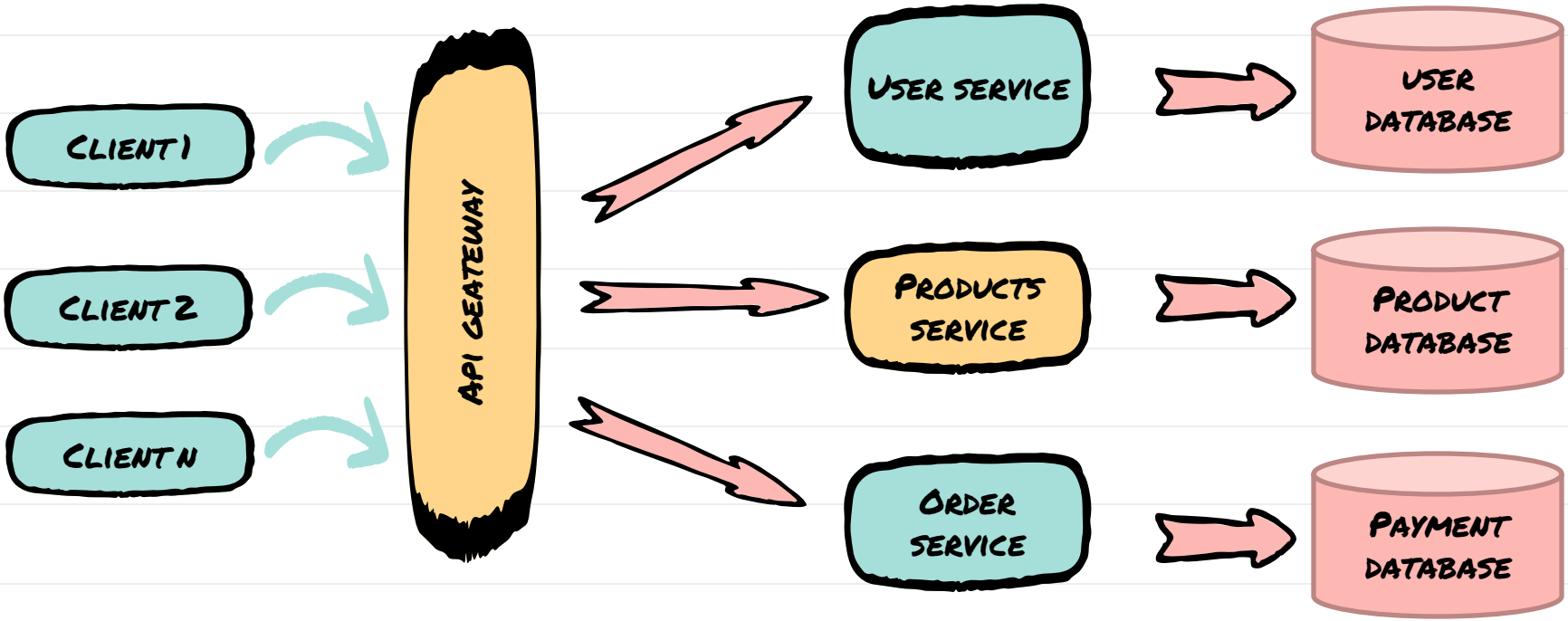
ADVANTAGES

- Simple to develop.
- Simple to deploy.
- Simple to scale horizontally by running multiple copies behind a load balancer.

DISADVANTAGES

- Maintenance
  - If Application is too large and complex to understand entirely, it is challenging to make changes fast and correctly.
- The size of the application can slow down the start-up time.
- You must redeploy the entire application on each update.
- Scale when different modules have conflicting resource requirements.
- Reliability
  - Bug in any module (e.g. memory leak) can potentially bring down the entire process.

Microservices architecture



ADVANTAGES

- Microservices Enables the continuous delivery and deployment of large, complex applications.
- Better testability
  - Services are smaller and faster to test.
- Better deployability.
  - Services can be deployed independently.
- Improved fault isolation.

DISADVANTAGES

- Developers must deal with the additional complexity of creating a distributed system.
- Developers must implement the inter-service communication mechanism.
- Implementing use cases that span multiple services without using distributed transactions is difficult.

MICROSERVICES

- Microservices are an architectural pattern that structures an application as a collection of small, loosely coupled services that operate together to achieve a common goal.

- Because they work independently, they can be added, removed, or upgraded without interfering with other applications.

CAP Theorem

- “Modern” apps require highly available distributed computing that can replicate changes across nodes .
- Cap Theorem States
  - It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees.
    - Consistency
    - Availability
    - Partition Tolerance

CAP Theorem

- Consistency
  - All nodes see the same data at the same time.
  - Achieved by updating several nodes before allowing reads “traditionally” a 2-phase commit.

CAP Theorem

- Availability
  - Every request receives a response about whether it succeeded or failed.
  - Achieved by replicating the data across different machines.

CAP Theorem

- Partition Tolerance
  - Operation continues despite arbitrary partitioning due to network failures.
  - Latency due to re-routing.

Basically Available, Soft state, Eventual consistency

- ACID ensures that at the point in time of the transaction [ACID] compliance is respected
- BASE allows for ACID compliance to be temporarily violated as long as it eventually gets to a compliant end state.

Eventual Consistency:

At some point, all data sources will show the same data

Tools for microservice architecture

- Containers
- Clustering
- Orchestration
- Cloud infrastructure
- API Gateway
- Service Discovery
- Distributed Loggings
- Tracing
- Circuit Breakers
- Load Balancing
- Messaging
- Configuration

Service discovery

- Client-side service discovery allows services to find and communicate with each other without hard-coding hostname and port.

- The only ‘fixed point' in such an architecture consists of a service registry with which each service has to register.

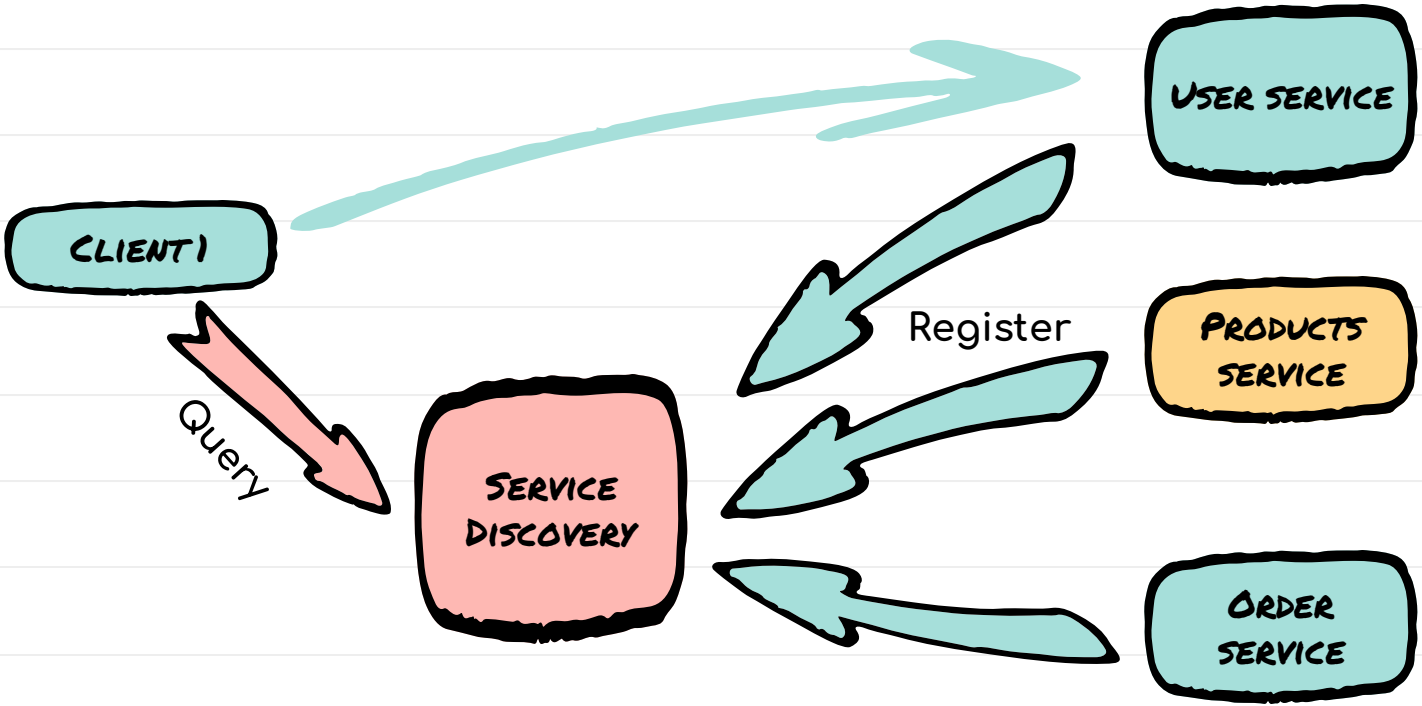
- A drawback is that all clients must implement a certain logic to interact with this fixed point. This assumes an additional network round trip before the actual request.

Service discovery

- Service discovery mechanism uses a central registry to maintain the network locations of all the microservices.

- If for some reason the IP address and the port number of a particular microservice changes, new values will be immediately re-registered in the registry.

Service discovery



Service discovery

- Spring cloud provides
  - Eureka
  - Zookeeper
  - Cloud Foundry
  - Consul

Spring Cloud Eureka

- Eureka is a REST based service which is primarily used for acquiring information about services that you would want to communicate with.
- This REST service is also known as Eureka Server.
- The Services that register in Eureka Server to obtain information about each other are called Eureka Clients.
- Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address.

Eureka server

- Add `spring-cloud-starter-netflix-eureka-server` to the dependencies.
- Enable the Eureka Server in a `@SpringBootApplication` by annotating it with `@EnableEurekaServer`.
- Application.yml

```
server:
  port: 8761
eureka:
  client:
    registerWithEureka: false
    fetchRegistry: false
```

Eureka server

- RegisterWithEureka:
  - Controls whether or not this client registers itself and therefore becomes discoverable.
- FetchRegistry:
  - Controls whether or not this client is going to try to connect to the Eureka server(s) in order to download the information on other services' endpoint.
  - It can do so without registering itself.

Eureka client

- Add `spring-cloud-starter-netflix-eureka-client` to the dependencies.
- Application automatically registers with the Eureka Server.
- Application.yml

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

Eureka client

- When a client registers with Eureka, it provides meta-data about itself such as host, port, health indicator URL, home page, and other details.

- Eureka receives heartbeat messages from each instance belonging to a service.

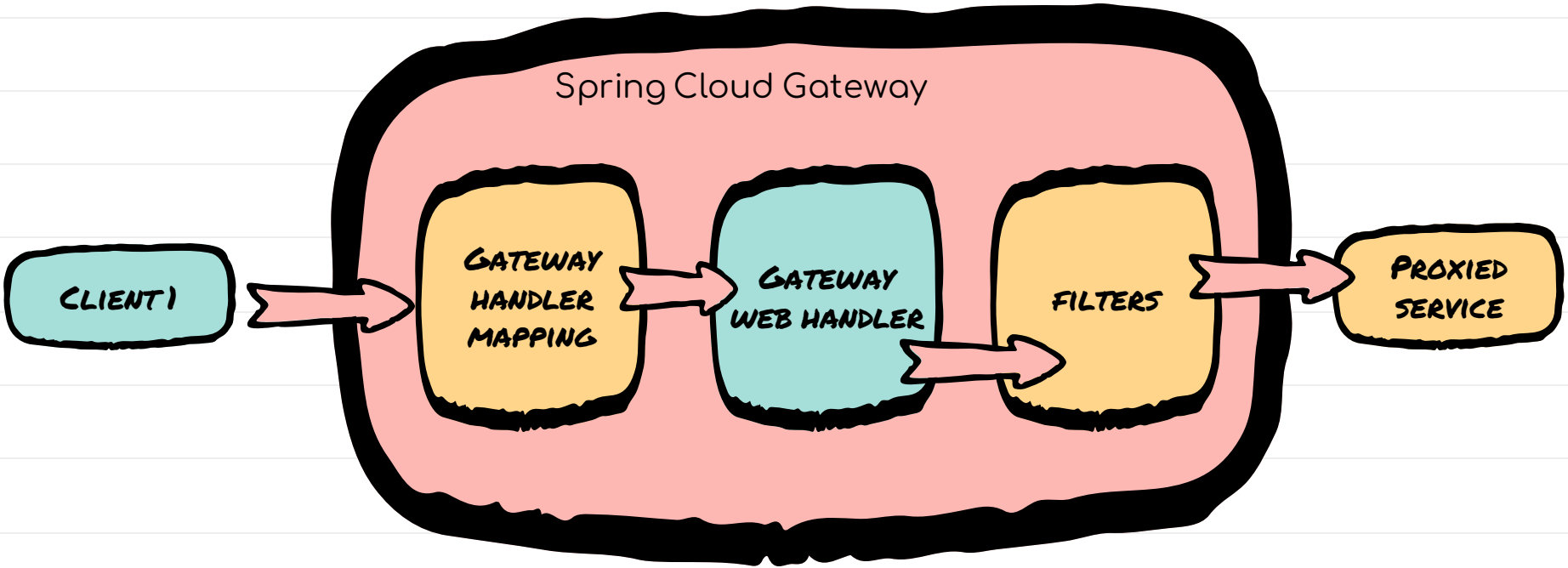
Spring Cloud Gateway

- Provides an API Gateway built on top of the Spring Ecosystem, including Spring 5, Spring Boot 2 and Project Reactor.
- Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as:
  - Security
  - Monitoring/metrics
  - Resiliency.

Spring Cloud Gateway features

- Built on Spring Framework 5, Project Reactor and Spring Boot 2.0
- Able to match routes on any request attribute.
- Predicates and filters are specific to routes.
- Circuit Breaker integration.
- Spring Cloud Discovery Client integration.
- Easy to write Predicates and Filters.
- Request Rate Limiting.
- Path Rewriting.

Spring Cloud Gateway



GLOSSARY

- Route
  - The basic building block of the gateway.
  - It is defined by an ID, a destination URI, a collection of predicates, and a collection of filters.
  - A route is matched if the aggregate predicate is true.

GLOSSARY

- Predicate
  - This is a Java 8 Function Predicate.
  - The input type is a Spring Framework `ServerWebExchange`.
  - This lets you match on anything from the HTTP request, such as headers or parameters.

GLOSSARY

- Filter
  - These are instances of Spring Framework `GatewayFilter` that have been constructed with a specific factory.
  - You can modify requests and responses before or after sending the downstream request.

How It works

- Clients make requests to Spring Cloud Gateway.
- If the Gateway Handler Mapping determines that a request matches a route, it is sent to the Gateway Web Handler.
- This handler runs the request through a filter chain that is specific to the request.

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
```

Spring cloud config

- Provides server and client-side support for externalized configuration in a distributed system.
- With the Config Server you have a central place to manage external properties for applications across all environments.
- The default implementation of the server storage backend uses git so it easily supports labelled versions of configuration environments, as well as being accessible to a wide range of tooling for managing the content.

FEATURES

- Server
  - HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content)
  - Encrypt and decrypt property values (symmetric or asymmetric)
  - Embeddable easily in a Spring Boot application using `@EnableConfigServer`.

FEATURES

- Client
  - Bind to the Config Server and initialize Spring Environment with remote property sources.
  - Encrypt and decrypt property values (symmetric or asymmetric)

Without config server

- It is necessary to update the properties file of each instance when a property is added or changed.

- It is required to relaunch each application so that new properties are taken into account.