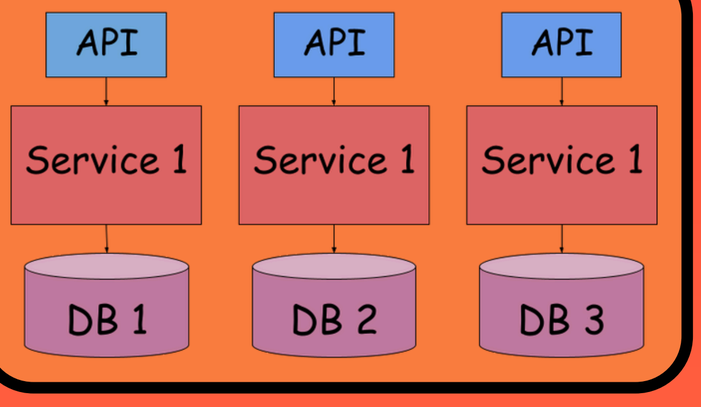
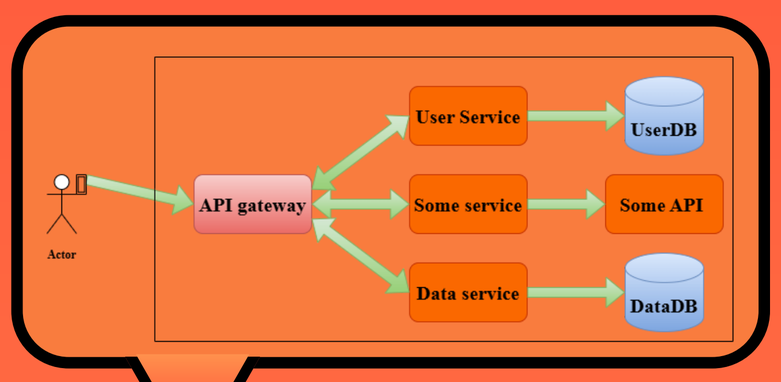
You can get strangled!

(if you're using monolithic architecture)

Database per service

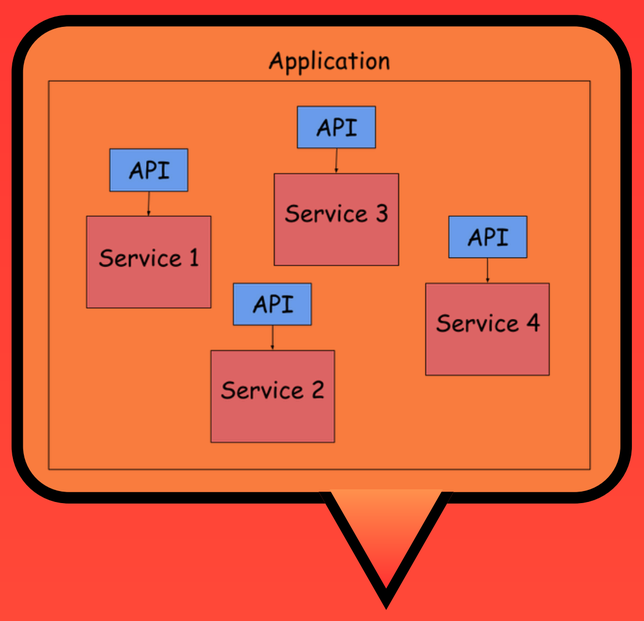
This design pattern divides the applications database into multiple ones where each service has its own database. This allows data that belong to different services to be separated and grouped with data that come from the same service. Microservices are designed to be small and independent from other services, and upphold loose coupling. This allows for an array of benefits.
1. If changes to a database need to be made, the changes won't affect other unrelated services.
2. Services can only access data that is relevant to their services.
3. The microservices can choose the database technology that best suits their needs.
4. A database crash will not affect the data storing abillities of other services. [1]





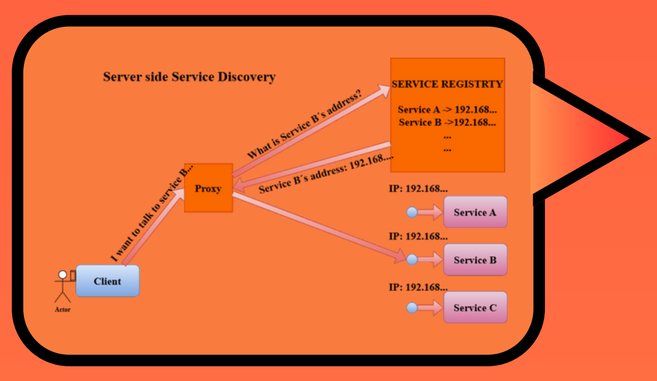
API Gateway

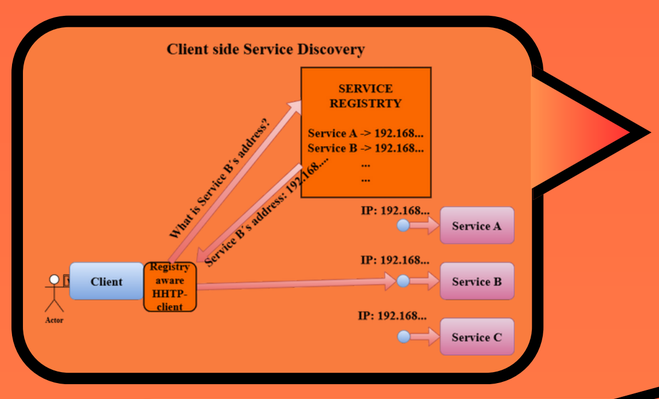
An API Gateway is a design pattern where the application provides a single access point for the client. The API Gateway handles the routing of requests to the services responsible for a specific task. It is a common design pattern for websites. From the client's perspective, there is only one user interface, but behind the API Gateway, multiple services may exist. For instance, when a client sends a request to access user data, the API Gateway forwards the request to the service responsible for managing users. An API Gateway can receive a single request from the client and then send multiple requests to different services. Examples of technologies implementing this pattern include Amazon API Gateway, Microsoft Azure API Management, and 3scale by Red Hat. [2]

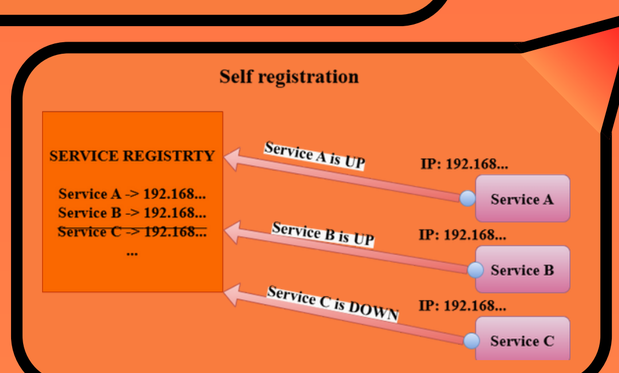


What are microservices?

Microservices are a design pattern that involves dividing the application's components/modules into smaller, independent services that can be implemented individually [5][7]. These modules can then be arranged in various ways depending on the use case. There are several design patterns to use. Examples include strangler, database per service, circuit breaker, API gateway, service discovery, saga pattern, decompose by business capability, decompose by subdomain, single service per host, multiple services per host, and many more [3][4][5]. These can be categorized based on their use case [8]. Design patterns can also be synchronous or asynchronous, where a service sends a request to another service and either waits for a response or it doesn't wait for a response [2]. Since the services are divided and communicate via REST calls, there are good reasons to use asynchronous communication to avoid halting the program when one service is waiting for a response.

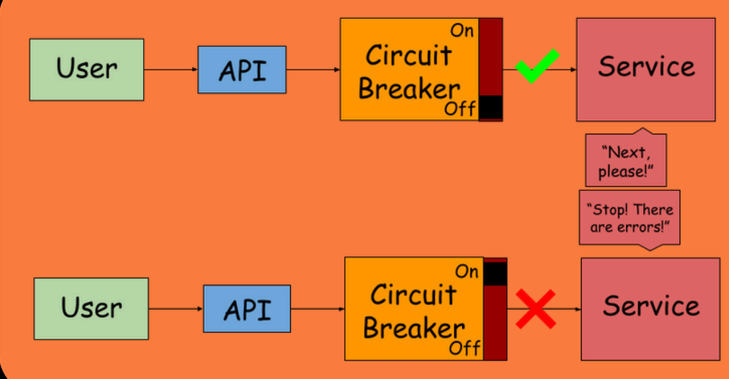






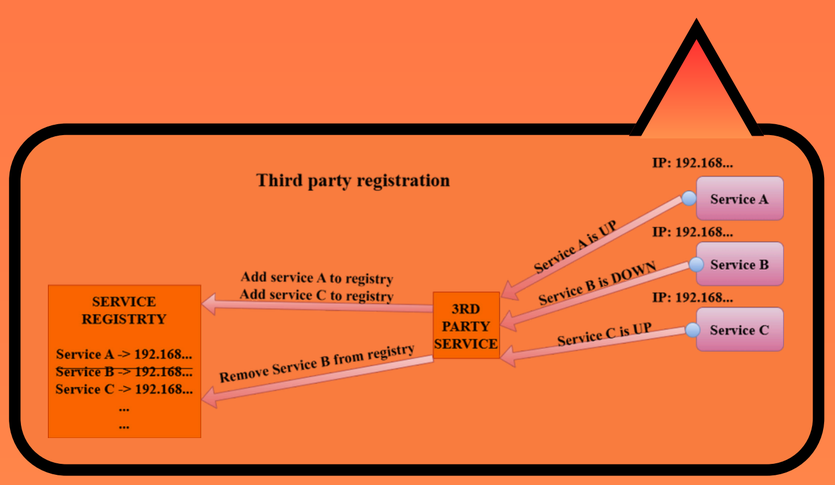
Circuit Breaker

This pattern safeguards a service from being overwhelmed with requests it cannot handle. It monitors whether requests to the service fail or succeed when being handled. If too many fail, the circuit breaker will trip and stop further access to the service, temporarily preventing it from needing to handle more requests. This is where the circuit breaker switches to the open state. This gives the service time to recover or time for the developers to fix potential problems with the service, before it can continue handling requests again. When it is ready, the circuit breaker will switch to the closed state and traffic can continue flooding the service as usual. [6]



Service Discovery

Service discovery refers to the process of locating services within a distributed system. The pattern involves using a service registry, which acts as a directory of addresses for the system's various services. Different Approaches to Service Discovery:
-Client-side service discovery: The client searches the service registry for the address of the desired service.
-Server-side service discovery: A proxy is used between the client and the service registry. The proxy searches the service registry for the service's address and provides it to the client.
There are two ways a service can be registered to the service registry:
1. Third Party Registration: A separate service is used to add or remove service addresses in the service registry, depending on whether the service is up or down.
2. Self-registration: In this approach, the services themselves notify the service registry about their availability. This solution typically uses a keepalive mechanism, where the service sends regular updates to avoid being removed from the service registry. [9]



References:

- [1] Medium/Mehmet Ozkaya. The database per service pattern [Internet]. Design Microservices Architecture with Patterns & Principles; 2021 [updated date; Sep 6]. Available from: https://medium.com/design-microservices-architecture-with-patterns/the-database-per-service-pattern-9d511b882425
- [2] Microservices - A Practical Guide: Wolff E. Second edition. Manning; 2019
- [3] DZone./Rajesh Bhojwani. Microservices Design Patterns: Essential Architecture and Design Guide
- [Internet]. 2024 [updated date; Oct 23] Available from: https://dzone.com/articles/design-patterns-for-microservices
- [4] Microservices.io/Chris Richardson. Microservices architecture patterns [Internet]. Available from: https://microservices.io/patterns/microservices.html
- [5] IONOS. What is Microservice architecture? [Internet]. 2023 [updated date; Jul 13] Available from: https://www.ionos.com/digitalguide/websites/web-development/microservice-architecture/
- [6] DZone/Dileep Kumar Pandiya. The Circuit Breaker Pattern: Fortifying Microservices Architecture [Internet]. 2024 [updated date; Mar 19]. Available from: https://dzone.com/articles/fortifying-the-circuit-breaker-pattern
- [7] Microsoft. Designing microservices: Microservice architecture patterns [Internet]. Available from: https://learn.microsoft.com/en-us/azure/architecture/microservices/design/patterns
- [8] Amazon. AWS. Welcome to Amazon API Gateway [Internet]. Available from: https://docs.aws.amazon.com/de_de/apigateway/latest/developerguide/welcome.html
- [9] Microservices.io/Chris Richardson. Service discovery in microservices [Internet]. Available from: https://microservices.io/tags/service%20discovery

Kungliga Tekniska Högskolan - TIDAA3 - HT2024

Simon Springer - simonspr@kth.se
Björn-Henrik Andersson - bhand@kth.se