Microservices

Service discovery ⇒ find available microservices and their locations.

$$\begin{tikzcd}\text{Service discovery} \arrow[d, dashed, "\text{done using service registry}"] \\ \text{service registry}\end{tikzcd}$$

service registry ⇒ holds the microservice instances and their locations.

$$\begin{tikzcd} & \text{client-side discovery} \\ \text{service discovery} \arrow[ur] \arrow[dr] & \\ & \text{server-side discovery}\end{tikzcd}$$

Client-side discovery } the service consumer is responsible for determining network locations of available instances and load balancing requests between them.

Client queries the service registry, then client uses a load-balancing algorithm to choose one of the available service instances & performs a request.

Server-side discovery } the client ~~queries~~ make a request to a service via a load balancer that acts as an orchestrator.
Load balancer queries the service registry and routes each request to an available service instance.

Dependability factors

(1) Availability ==> system ready to be used immediately.

(2) Reliability ==> system can run continously without failure.

(3) Safety ==> if system fails, nothing ~~catasphore~~ catastrophic will happen.

(4) Maintainability ==> when a system fails, it can be repaired easily and quickly.

Redundancy

Redundancy --- addition of information, resources, time beyond what is needed for normal system operation.

- → Software redundancy ⇒ addition of extra software, beyond what is needed to perform a given function, to detect and possibly tolerate faults.
- → Hardware redundancy ⇒ addition of extra hardware, usually for detecting and tolerating faults.
- → Information redundancy ⇒ addition of extra information beyond that required to implement a given function. (Ex; error deectection codes)
- → Time redundancy ⇒ uses additional time to perform the functions of a system such that fault detection and fault tolerance can be achieved.

Java RMI blocking

synchronous communication method, where the client send a request to the server and waits (blocks) for the server to respond before continuing execution.

Appropriate use cases

Authentication Services :
Logging into an online banking application where the client needs to wait for confirmation before proceeding.

Data Retrieval :
when a client needs to fetch data ~~that~~ that is crutial for the next step in the workflow, such as retrieving user profile after login.

Problems solved

- Ensures sequential execution where the next steps depend on the result of previous remote method call.
- Simplifies client-side logic by handling responses in a linear & predictable manner.

Java RMI with Async. Callback Functions

allows a client to invoke a remote method without waiting for the result. The server processes the request & calls back a client-side method upon completion.

use cases

Event-driven systems :
Receiving alerts from a remote health monitoring system when heart rate exceeds a particular value.

Notification services :
Sending notifications for various async. events like email/SMS alerts.

Problems solved

- Improves responsiveness by allowing the client to continue other tasks while waiting for the server to complete the proceeding.
- Suitable for real-time notification systems where immediate action is required upon certain conditions.

Java RMI-based Polling

Java RMI-based polling involves the client periodically checking with the server to get updates. The client sends requests at regular intervals to get the current state or data from the server.

use cases

Periodic data retrieval :
checking the heart rate using a remote health monitoring system every 5 minutes.

Monitoring system :
Regularly polling for status updates in network/application monitoring tools

Problems solved

- Ensures that the client remains updated with latest info. at specified intervals.
- useful when continuous realtime updates are not necessary, and periodic updates are sufficient.

Socket Programming

allows for low-level network communication between 2 systems over TCP/IP or UDP. It provides a direct means of sending & receiving data packets between client & server.

use cases

Real-time systems :
Fire Alarm systems that need to send/receive data with low latency and high reliability.

custom protocols :
Applications requiring bespoke communication protocols for specific performance or security needs.

problems solved

- Provides high control over the network communication process, enabling fine-tuning for performance and reliability.
- Suitable for applications requiring direct and efficient data exchange.

Java RMI framework --→ ② components

Stub ---- the stub hides the complexity of remote method invocation from the client.
It provides the same method signatures as the remote object, allowing the client to call methods on the stub, as if it were the actual remote object.

Skeleton ---- the skelton acts as the intermediary on the server side that receives method invocations from the stub, process them, & return results.
It ensures that the remote object's methods are executed correctly and results are communicated back to the client.