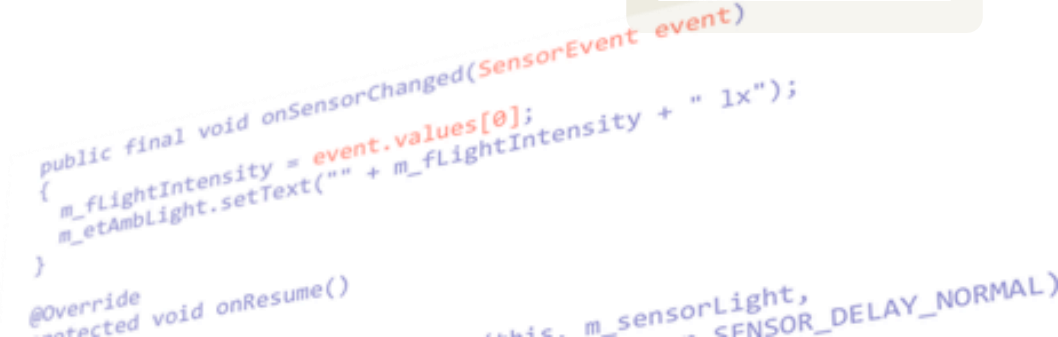
Container vs Serverless architecture design

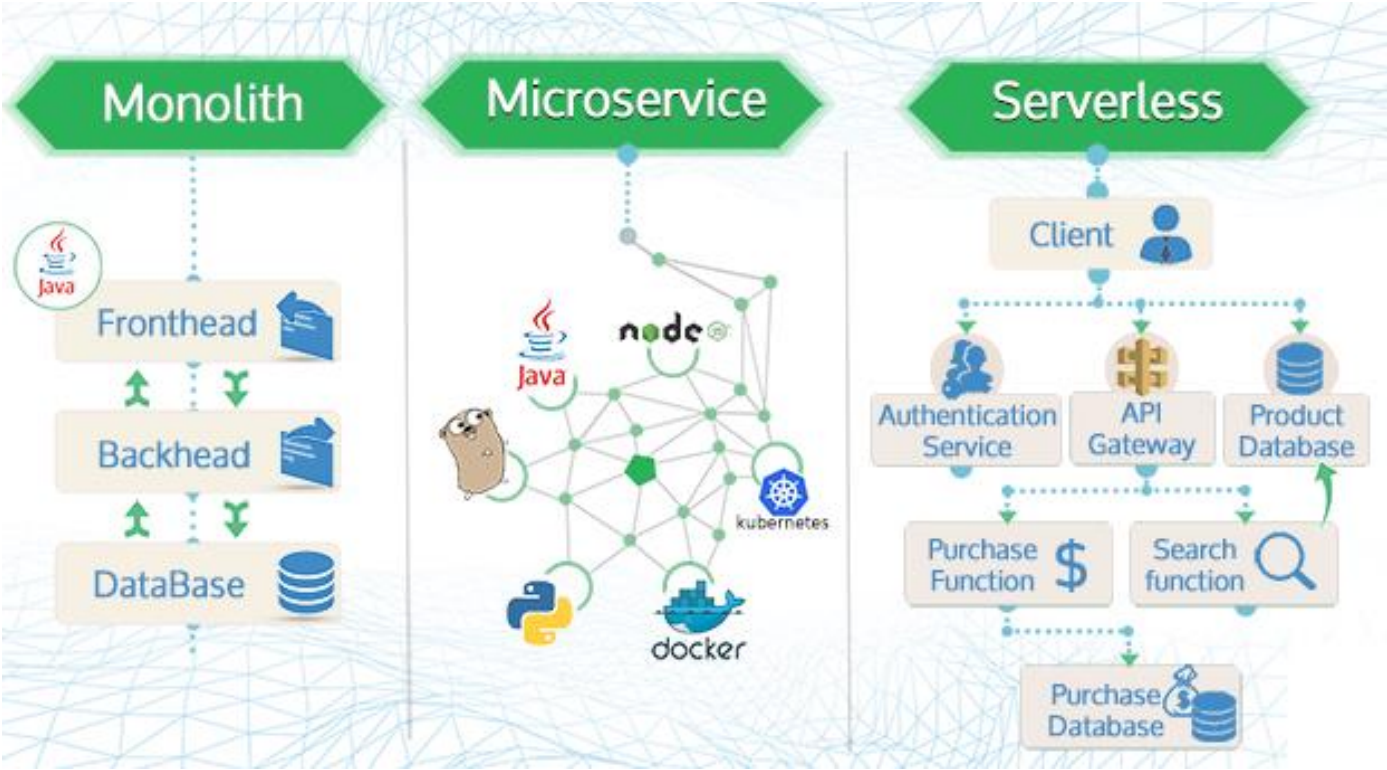
Jiří Šebek

NSS

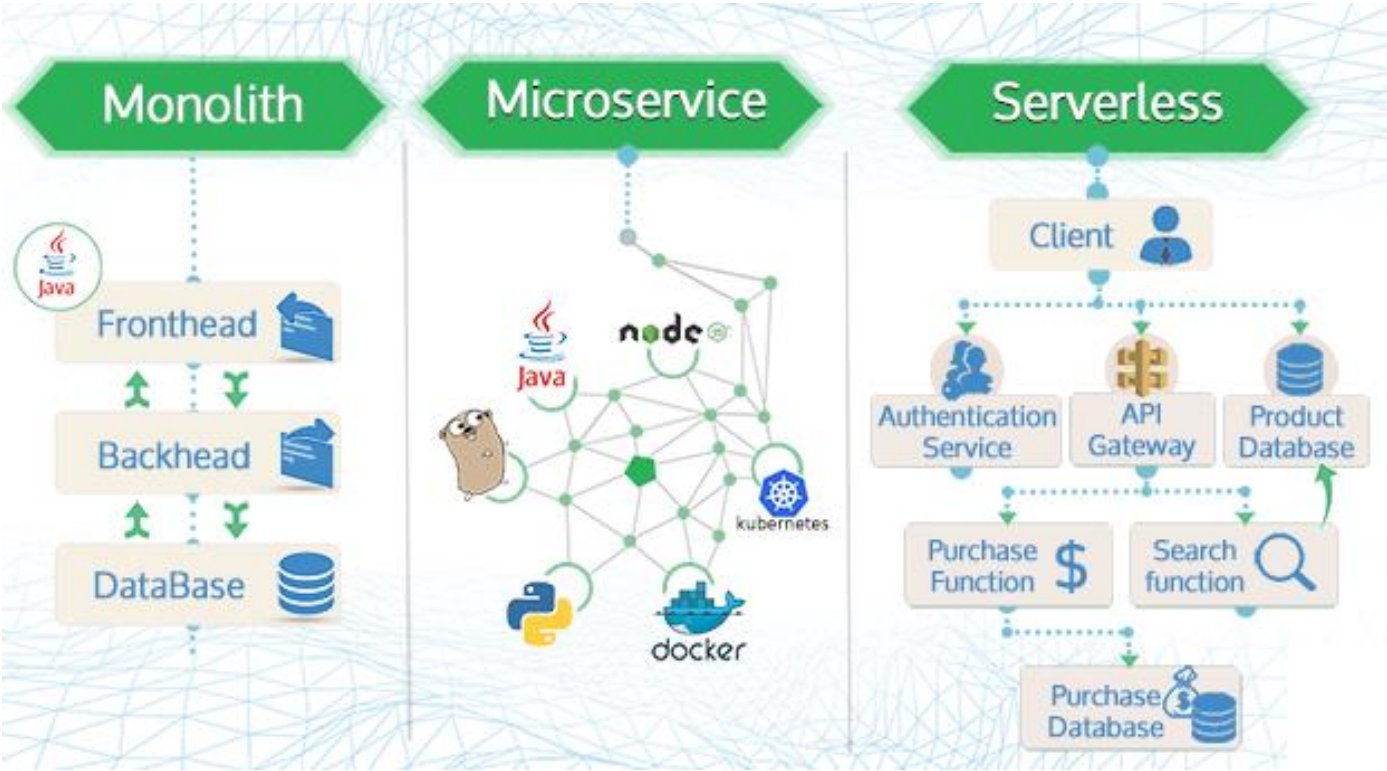


Container	Serverless	Advantage
A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.	Functions are blocks of code, ideally small and single-purpose. They are coordinated & scheduled by a Functions-as-a-service (FaaS) platform such as AWS Lamda, Azure Functions & Google Cloud Functions	With Faas(Serverless), developers never think about infrastructure. No-Ops Model. A container can contain multiple-functions, or just one function.
A container can run an entire application like a Database, DBMS & provide multiple outputs.	Functions have a clear and defined purpose with minimal API. Function takes a clearly defined short input & provides a reasonably short output.	Containers are preferred for complex software which can run for several minutes/hours. Functions are preferred for simple applications.
A container can live forever & scale horizontally – multiple containers can be created across servers	Functions are essentially ephemeral – lives for few seconds or minutes & is created on demand Containers can scale on demand supporting both Scale-up & Scale-out mode.	Containers are preferred for webscale apps that span across multiple servers. Functions are preferred mobile/web apps which are invoked by user. Eg: Fetch a PDF file, Apply specific filter on a picture etc.
Containers are billed based on the block of reserved resources + consumption of storage & N/W	Functions are billed on actually consumed resources	Functions an finer, atomic unit of compute: Runtime, Storage & data transferred. This allows for a refined consumption models

Differences



Differences



Java Core Sockets

Java RMI, XML-RPC, REST API, SOAP, CORBA ...

Historical evolution of development

At very starting we used to build monoliths, three-tier applications.

They were very heavy weight.

They are very slow to deploy.

Had trouble testing them.

After monolith, we broke these down into microservices.

They focused on being composable.

We deploy them with Docker containers.

The next step in the evolution.

Functions are small, discrete, reusable pieces of code that we can deploy.

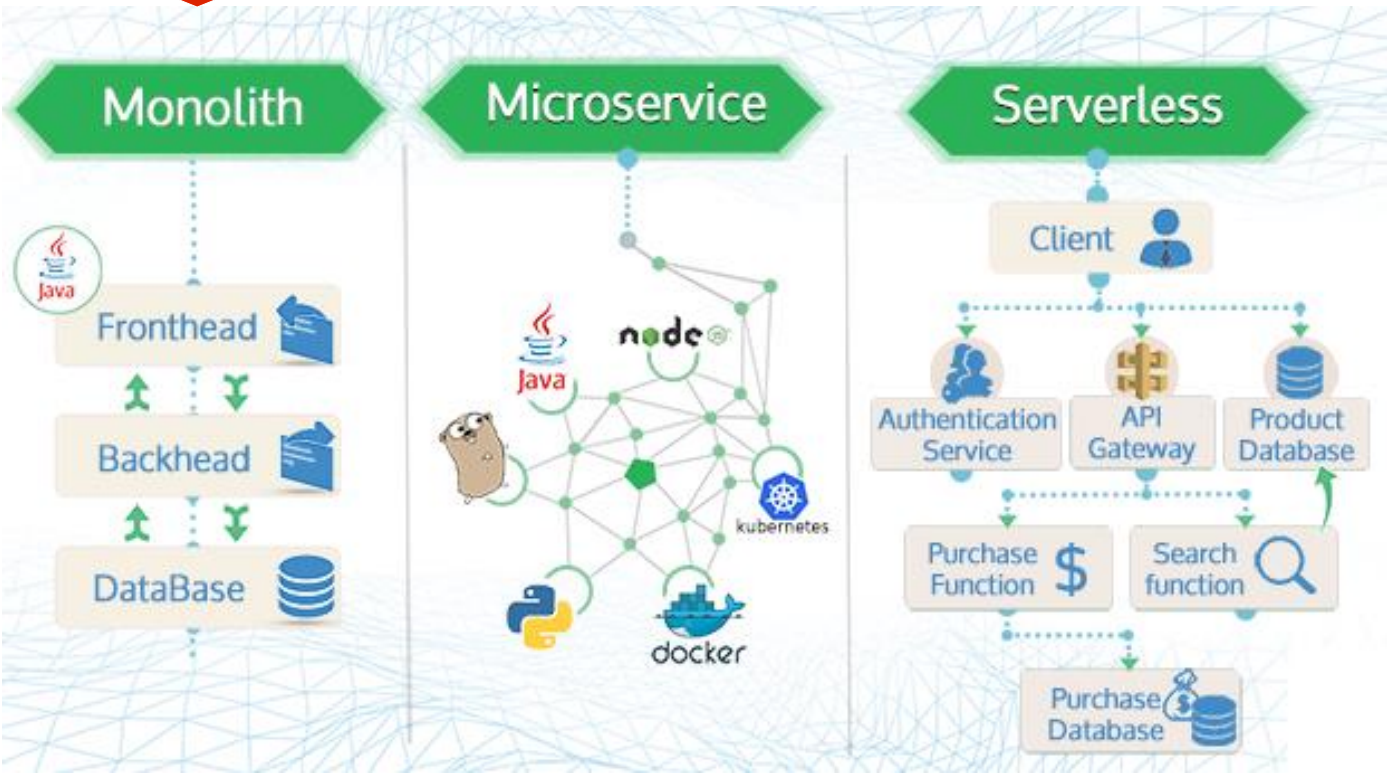
Not stateful.

Makes use of our existing services or third party resources.

Executes in milliseconds. Based upon AWS Lambda's default.

Serverless functions do not replace our monolith or microservice, they work best alongside our existing systems building integrations and helping events flow between our ecosystem.

Differences



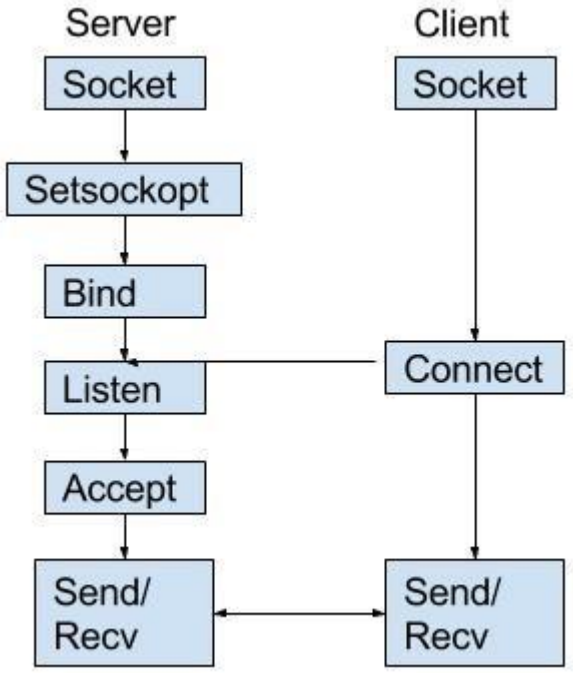
Java Core Sockets

Java RMI, XML-RPC, REST API, SOAP, CORBA ...

Monolitic architecture – Java Core Sockets

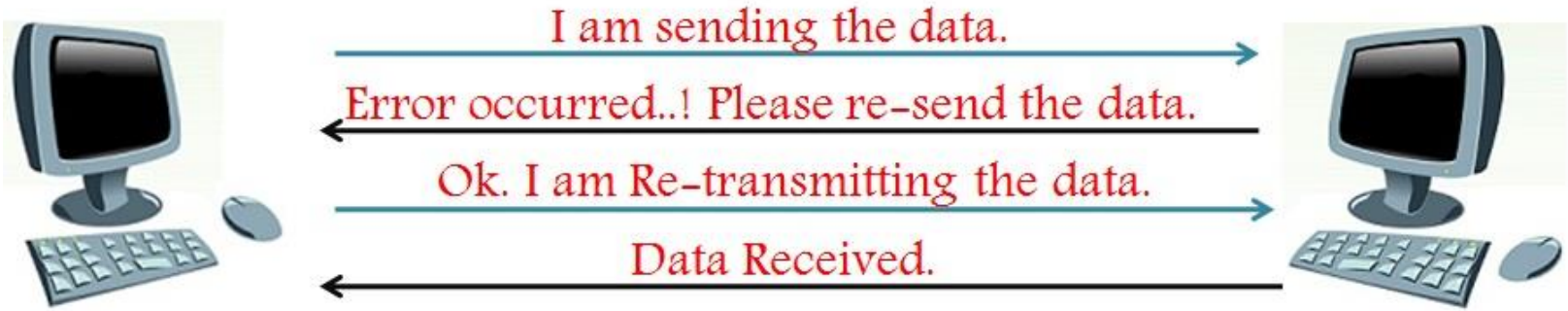
Sockets provide the communication mechanism between two computers using TCP

Example in repository

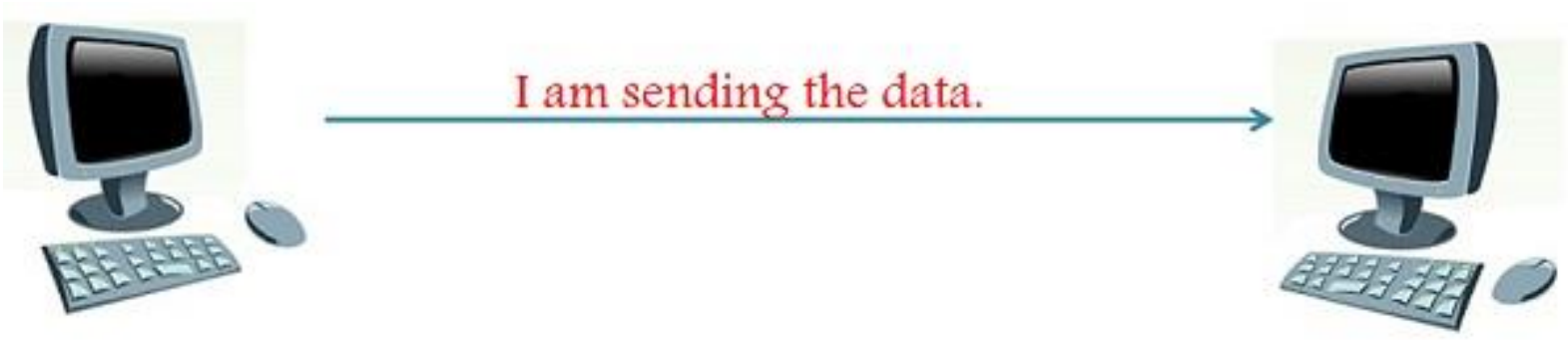


TCP vs UDP

TCP



UDP



TCP vs UDP

TCP is Connection-oriented whereas, UDP is Connectionless protocol.

TCP is highly reliable for transferring useful data as it takes the acknowledgment of information sent. It resends the lost packets if any. Whereas in the case of UDP if the packet is lost it won't request for retransmission and corrupt data is received by the destination computer. So, UDP is an unreliable protocol.

TCP is slower as compared to UDP since TCP establishes the connection before transmitting data, and ensures the proper delivery of packets. On the other hand, UDP does not acknowledge whether the data transmitted is received or not.

Header size of UDP is smaller then TCP and TCP header contains options, padding, checksum, flags, data offset, acknowledgment number, sequence number, source and destination ports, etc.

Both TCP and UDP can check for errors, but only TCP can correct the error since it has both congestion and flow control.

Monolitic architecture – Java Core Sockets

Advantages:

Sockets are flexible and sufficient.

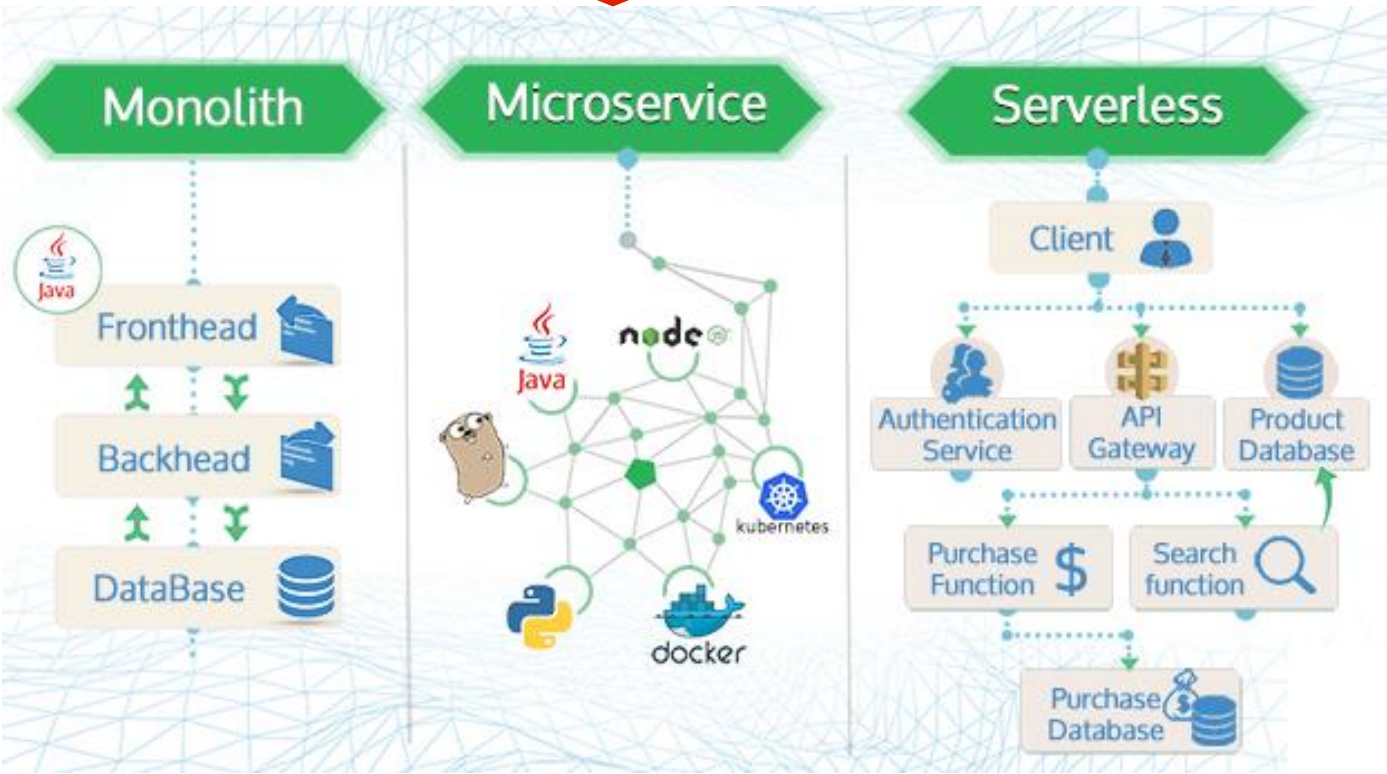
Sockets cause low network traffic. Unlike HTML forms and CGI scripts that generate and transfer whole web pages for each new request, Java applets can send only necessary updated information.

Disadvantages:

Security restrictions are sometimes overbearing.

Socket based communications allows only to send packets of raw data between applications. Both the client-side and server-side have to provide mechanisms to make the data useful in any way.

Differences



Java Core Sockets

Java RMI, XML-RPC, REST API, SOAP, CORBA ...

Java RMI

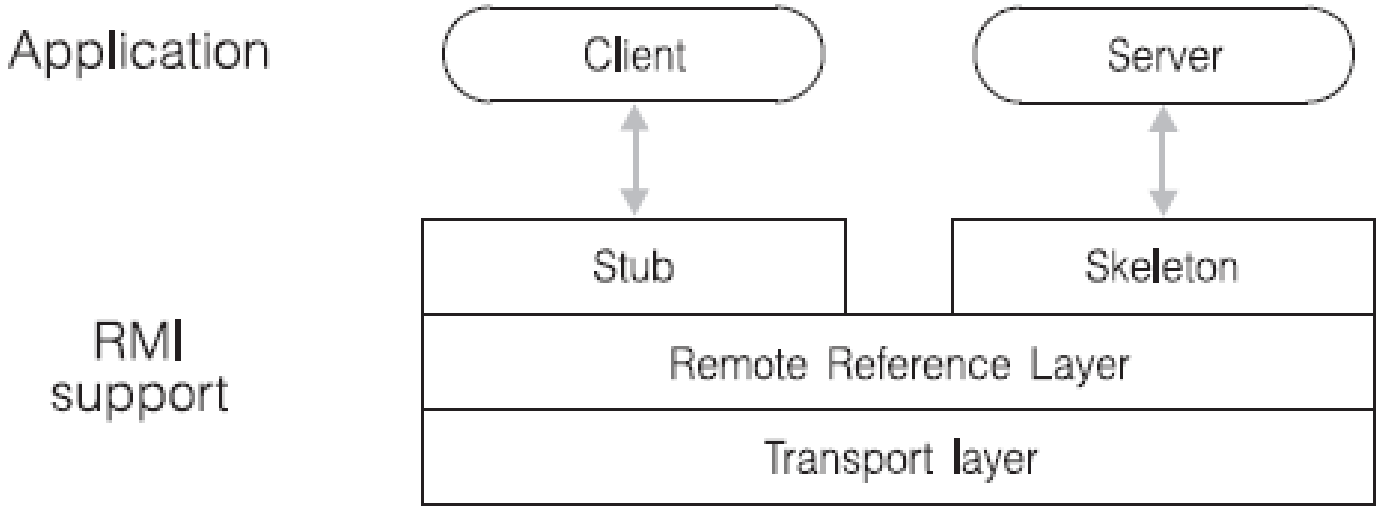
RMI allows us to write distributed objects using Java (easier then with sockets)

Similar to XML-RPC

Difference between RPC and RMI is that RMI involves objects. Instead of calling procedures remotely by use of a proxy function, we instead use a proxy object.

Example in repository

Java RMI


Application
Client
Server
Stub
Skeleton
RMI support
Remote Reference Layer
Transport layer

REST API

RMI allows us to write distributed objects using Java (easier then with sockets)

Example in repository

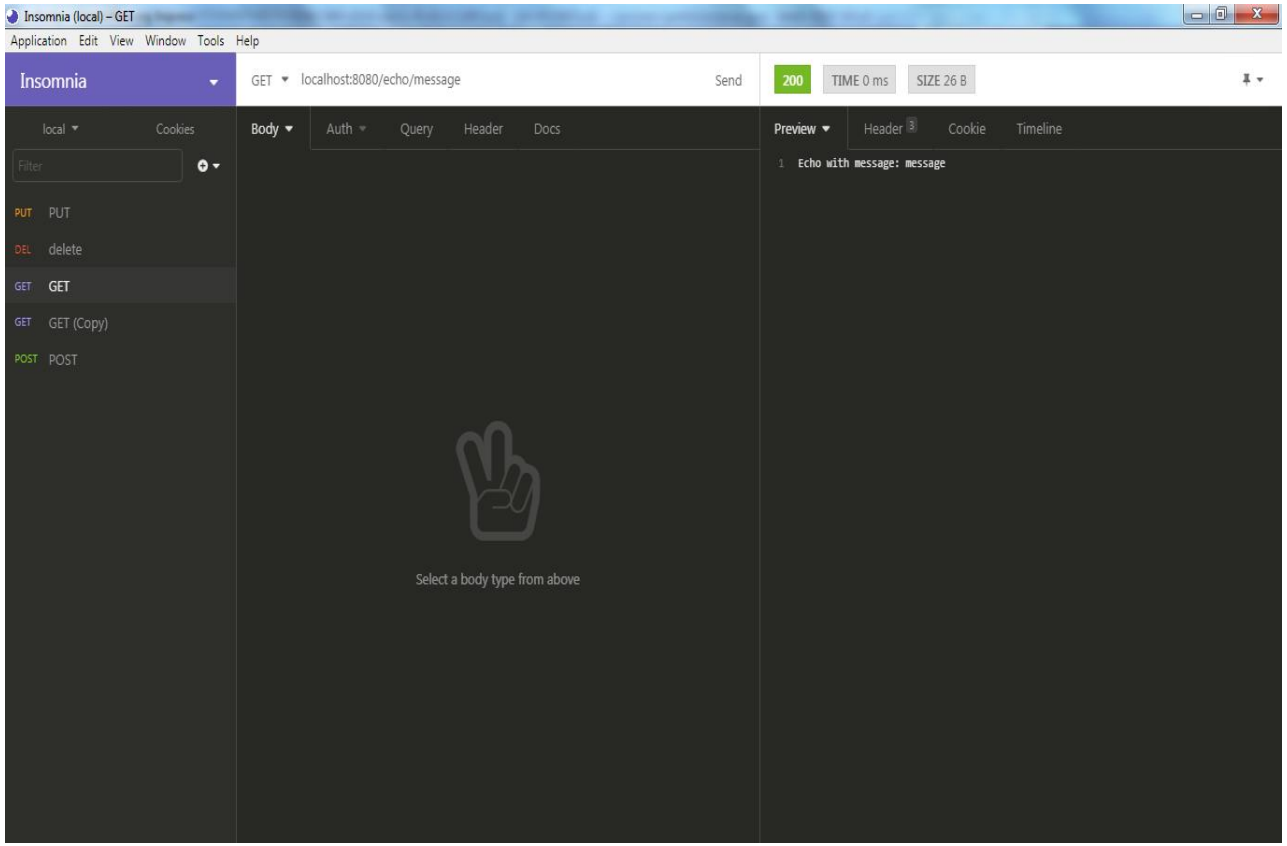
For client-side download one of the software

- Insomnia https://insomnia.rest/

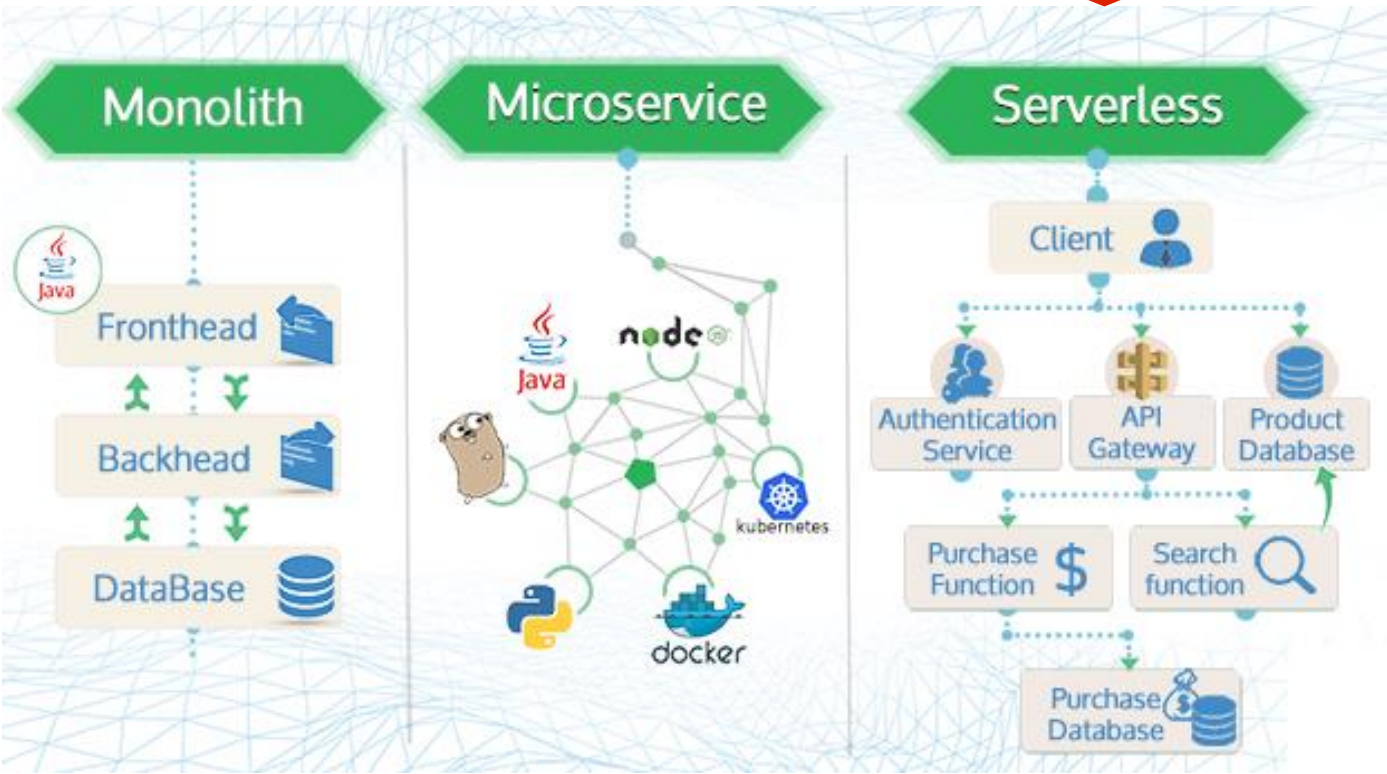
- Postman https://www.getpostman.com/

REST API

Insomnia



Differences



Java Core Sockets

Java RMI, XML-RPC, REST API, SOAP, CORBA ...

Serverless architecture

+ No server management
+ Pay-as-you-go
+ Inherent scalability
+ Quick deployments and updates
+ Code run closer to the client
- Complicated testing and debugging
- Not build for long running operations
- Performace
- Vendor lock-in

https://www.cloudflare.com/learning/serverless/why-use-serverless/

Azure Functions

Complicated testing and debugging

- local development and testing
  https://docs.microsoft.com/en-us/azure/azure-functions/functions-develop-local
- Visual Studio Code & Azure Function extension

Not build for long running operations

- Durable Functions
  https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview