Evaluation of Microservice Architecture Designs in an IoT-Context

Abramov Sviatoslav
svyatabram@gmail.com

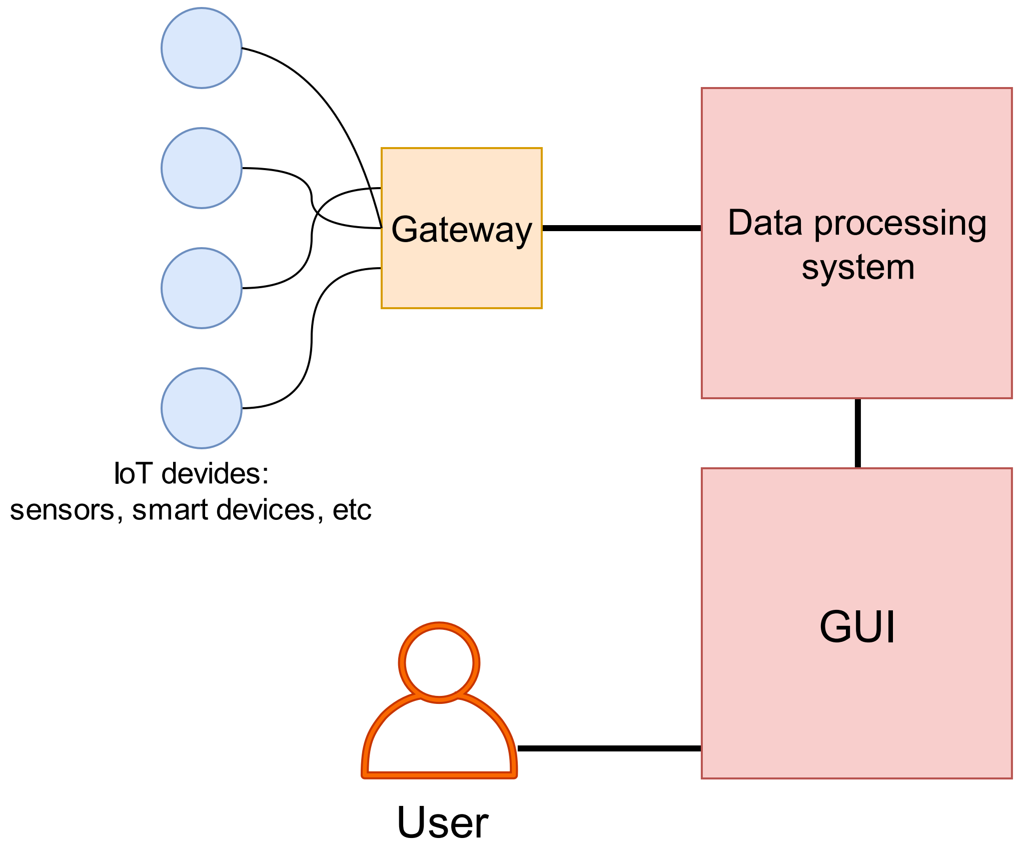
Supervisor: Prof. Dr.-Ing Günter Schäfer

Supervisor: Associate Prof. Dr. Igor Anikin

Content

Introduction

The Internet of Things

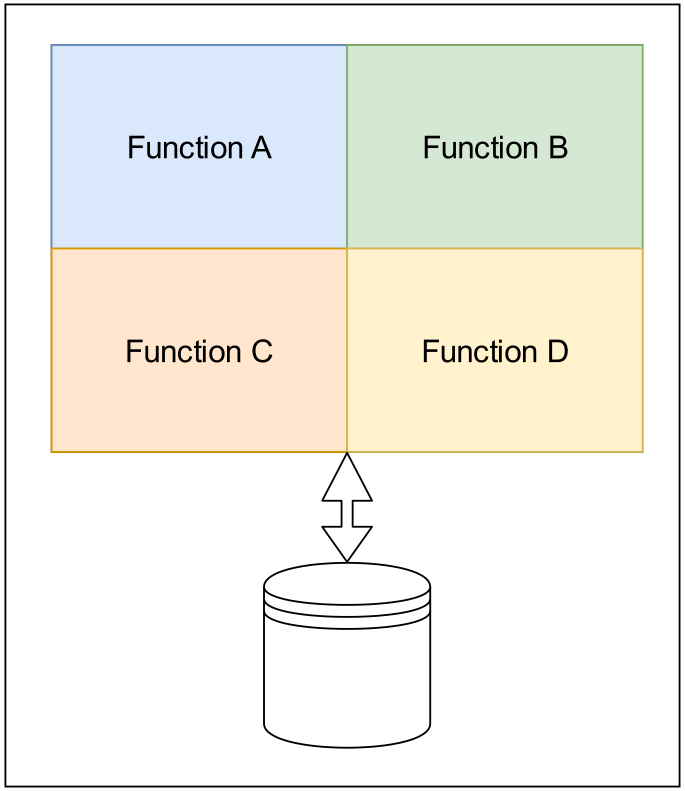


IoT architecture diagram.

IoT architecture elements:

- *Gateway* collects and transports IoT devices data.
- *DPS* is the main data processing system.
- *GUI* represents processed data to end users.

Monolithic architecture



Monolithic architecture example.

Drawbacks:

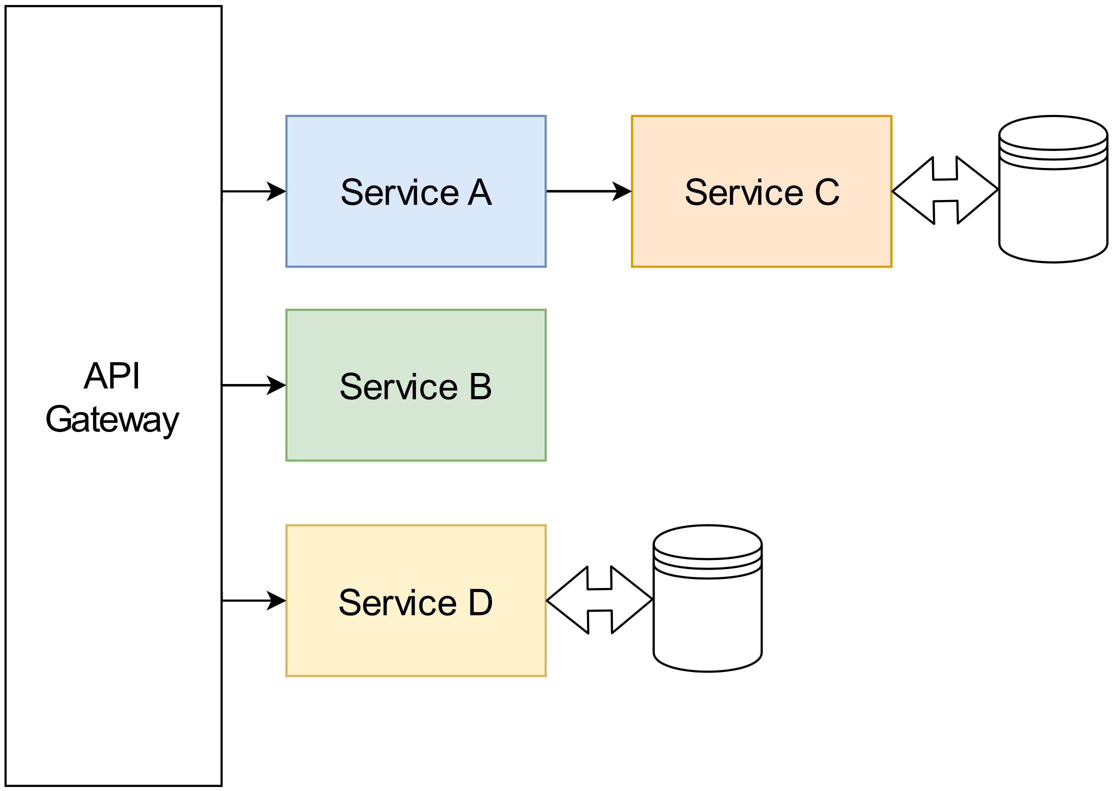
- Can not be horizontally scaled
- Modules share the same memory space
- A module bugs affect the whole application
- Continuous deployment is almost impossible

Advantages:

- Easy to develop and deploy

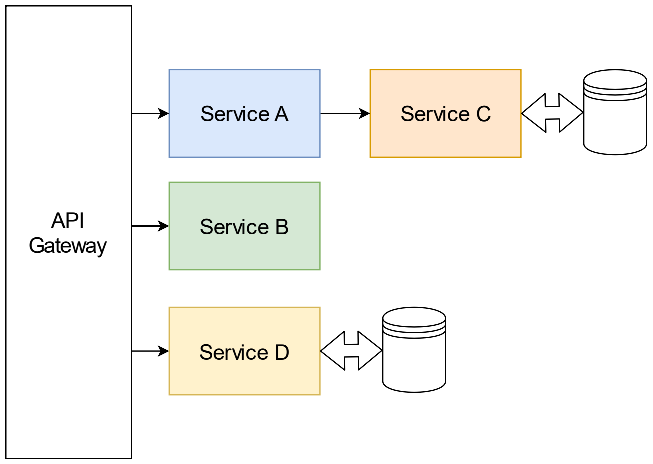
Suitable for lightweight and not multifunctional applications.

Microservices architecture



Microservices architecture example.

Microservices architecture



Microservices architecture example.

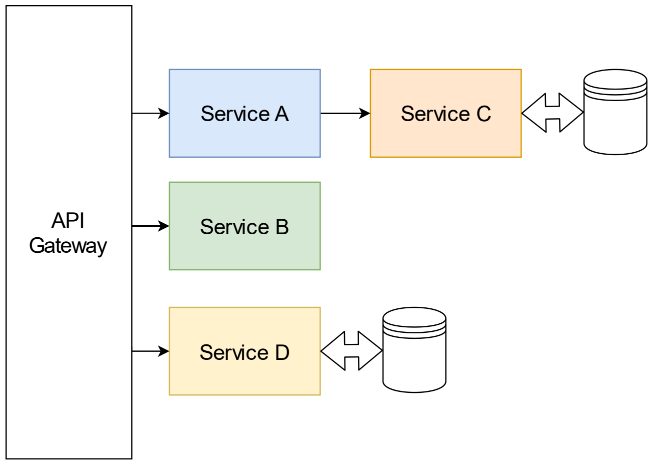
Advantages:

- Functional decomposition
- Each service can be scaled independently
- Services do not share the same memory space
- Each service is developed independently

Drawbacks:

- Complex deploying and testing
- Difficult interconnection process

Microservices architecture



Microservices architecture example.

Advantages:

- Functional decomposition
- Each service can be scaled independently
- Services do not share the same memory space
- Each service is developed independently

Drawbacks:

- Complex deploying and testing
- Difficult interconnection process

Suitable for high-loaded and multifunctional application such as IoT data processing system

The main goal

To find the most performable MSA architecture design in an IoT context.

The main goal

To find the most performable MSA architecture design in an IoT context.

We compared:

► Interconnection methods

► Database management systems

► Load balancing strategies

What we have done

1. Analyzed the most demanded technologies in MSA context.
2. Built the MSA application, satisfying the most common requirements.
3. Developed a load generation, simulating IoT devices.
4. Implemented a monitoring system
5. Processed measurements.

Requirements

Functional requirements

1. Provide connectivity for IoT devices.
2. Transform IoT device data model to the system data model.

Non-functional Requirements

Qualitative

- Testable
- Reproducible
- Deployable

Non-functional Requirements

Qualitative

- Testable
- Reproducible
- Deployable

Quantitative

- Response time
- Scalable

State of the art

Basic articles

M. S. Hatem Hamad and R. Abed, “Performance evaluation of restful web services for mobile devices,” *Computer Engineering Department, Islamic University of Gaza, Palestine, International Arab Journal of e-Technology*, 2010.

In the article advantages of RESTful web services before SOAP web services are shown:

- RESTful web services provide less message size.
- RESTful web services provide less response time.

Basic articles

P. J. Amaral M. and C. D., “Performance evaluation of microservices architectures using containers.,” *IEEE 14th International Symposium on Network Computing and Applications*, 2015.

This article shows Server Virtualization provides performance improvement

- SV increases server throughput
- SV decreases server latency

Basic articles

- J. F. Kunhua Zhu and Y. Li, *Research the performance testing and performance improvement strategy in web application", 2nd international Conference on Education Technology and Computer.* 2010.

This article provides a survey about overall MSA based application testing :

Basic articles

- J. F. Kunhua Zhu and Y. Li, *Research the performance testing and performance improvement strategy in web application", 2nd international Conference on Education Technology and Computer.* 2010.

This article provides a survey about overall MSA based application testing :

- Functional and load testing of web MSA based web applications basis.
- Prediction of an application response time changing with an increasing user load.
- User load impact on application throughput.
- Possible bottlenecks caused by system utilization.

Not answered questions

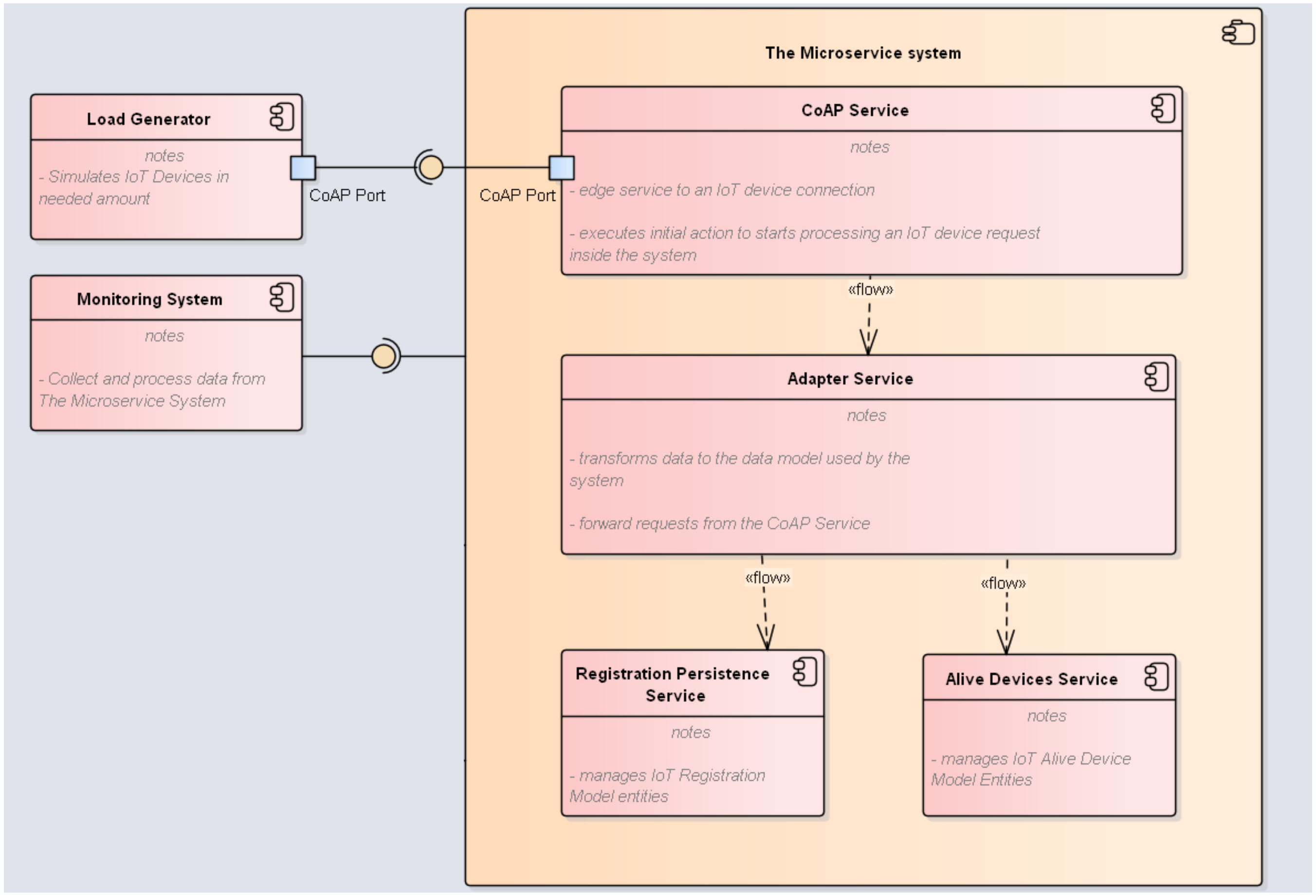
- Which design chooses we have to made to build MSA system in an IoT-device context?
- Which interconnection method fits a certain service functionality better?
- How to use load balancing according to services functionality?

The system overview

The system developing

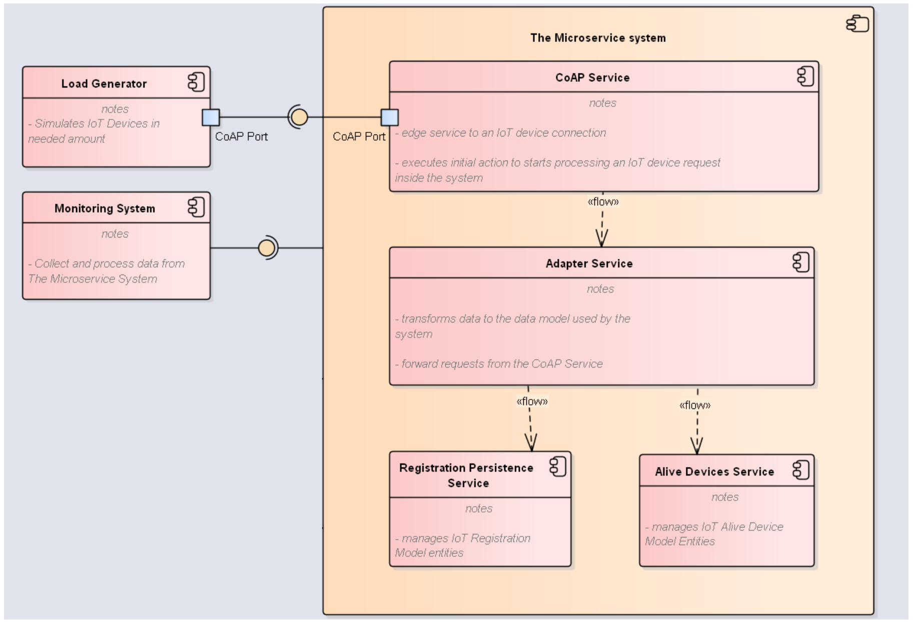
- The MS system must provide connectivity, session management, and data model transformation.
- Response time is assessment criteria.
- The server virtualization and imaging
- Load generator simulates a given amount of IoT devices

The system developing


The Microservice system
Load Generator
notes
- Simulates IoT Devices in needed amount
CoAP Port
CoAP Port
CoAP Service
notes
- edge service to an IoT device connection
- executes initial action to starts processing an IoT device request inside the system
«flow»
Monitoring System
notes
- Collect and process data from The Microservice System
Adapter Service
notes
- transforms data to the data model used by the system
- forward requests from the CoAP Service
«flow»
«flow»
Registration Persistence Service
notes
- manages IoT Registration Model entities
Alive Devices Service
notes
- manages IoT Alive Device Model Entities

The system components diagram.

The system developing



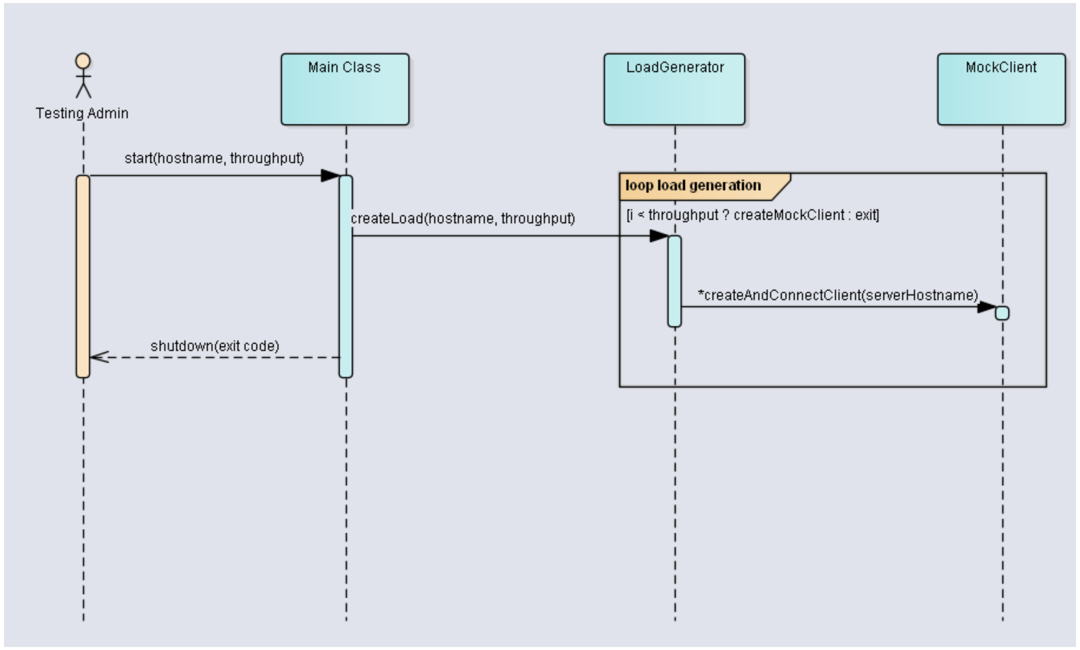
The Microservice system
Load Generator
notes
- Simulates IoT Devices in needed amount
CoAP Port
CoAP Port
CoAP Service
notes
- edge service to an IoT device connection
- executes initial action to starts processing an IoT device request inside the system
«flow»
Monitoring System
notes
- Collect and process data from The Microservice System
Adapter Service
notes
- transforms data to the data model used by the system
- forward requests from the CoAP Service
«flow»
«flow»
Registration Persistence Service
notes
- manages IoT Registration Model entities
Alive Devices Service
notes
- manages IoT Alive Device Model Entities

The system components diagram.

3 service types:

- Edge service – *CoAP service* provides connectivity for IoT devices
- Data transformation service – *Adapter service* generates the system data models for further their processing
- Persistence services – *Registration* and *Alive devices persistence* service ensure system object storing.

Load generation

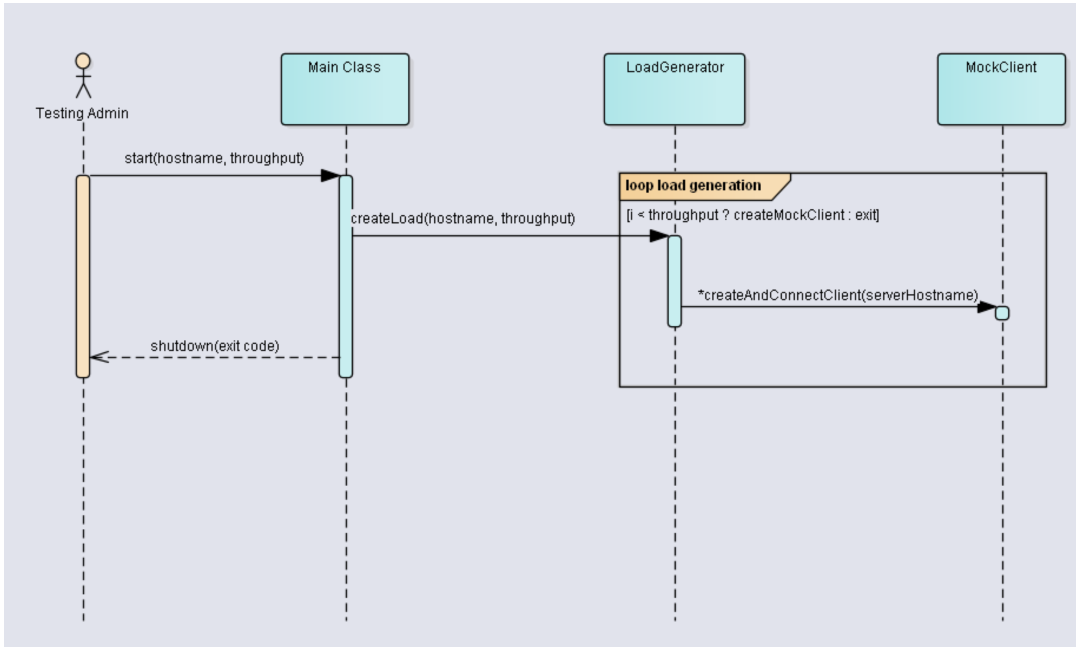

Testing Admin
Main Class
LoadGenerator
MockClient
start(hostname, throughput)
createLoad(hostname, throughput)
loop load generation
[i < throughput ? createMockClient : exit]
*createAndConnectClient(serverHostname)
shutdown(exit code)

Load generation sequence diagram.

For each of the interconnection type we generated load of approximately 50% system maximal throughput.

- HTTP(sync) – 10 devices.
- HTTP(async) – 90 devices.
- gRPC – 170 devices.
- RabbitMQ – 290 devices.

Load generation


Testing Admin
Main Class
LoadGenerator
MockClient
start(hostname, throughput)
createLoad(hostname, throughput)
loop load generation
[i < throughput ? createMockClient : exit]
*createAndConnectClient(serverHostname)
shutdown(exit code)

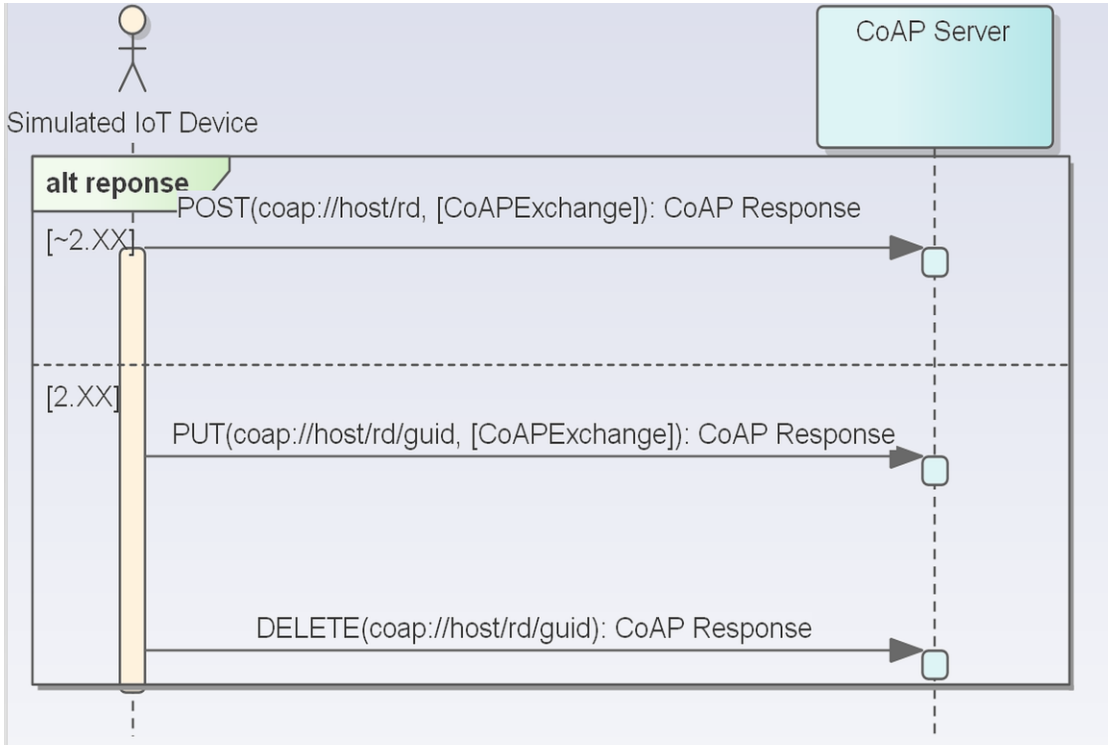
Load generation sequence diagram.

For each of the interconnection type we generated load of approximately 50% system maximal throughput.

- HTTP(sync) – 10 devices.
- HTTP(async) – 90 devices.
- gRPC – 170 devices.
- RabbitMQ – 290 devices.

We repeat every test run 30 times to ensure more accurate results.

Silmulated IoT device

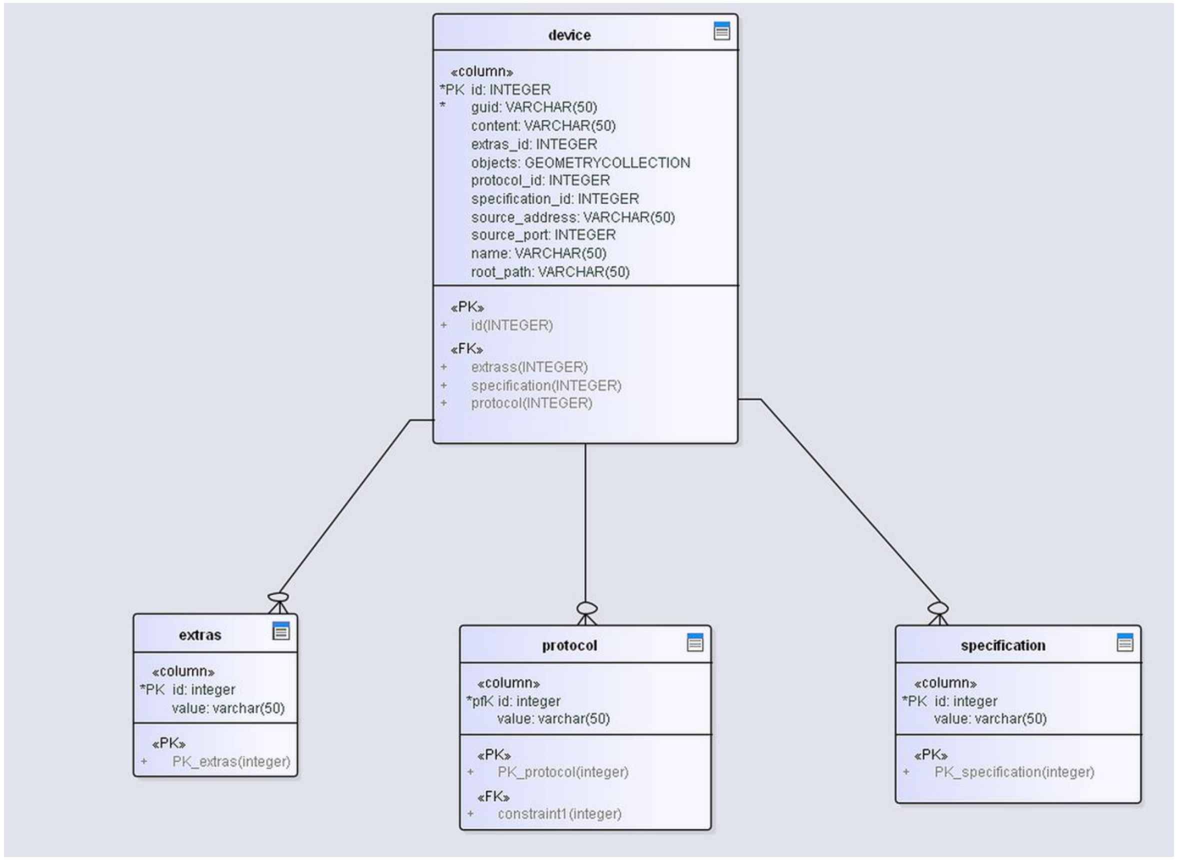

Simulated IoT Device
CoAP Server
alt reponse
[~2.XX]
POST(coap://host/rd, [CoAPExchange]): CoAP Response
[2.XX]
PUT(coap://host/rd/guid, [CoAPExchange]): CoAP Response
DELETE(coap://host/rd/guid): CoAP Response

Simulated IoT device lifecycle SD.

Requests:

- POST – to save IoT device on the given host
- PUT – to update information about IoT device on the given host
- DELETE – to delete information about IoT device on the given host

Registration data model

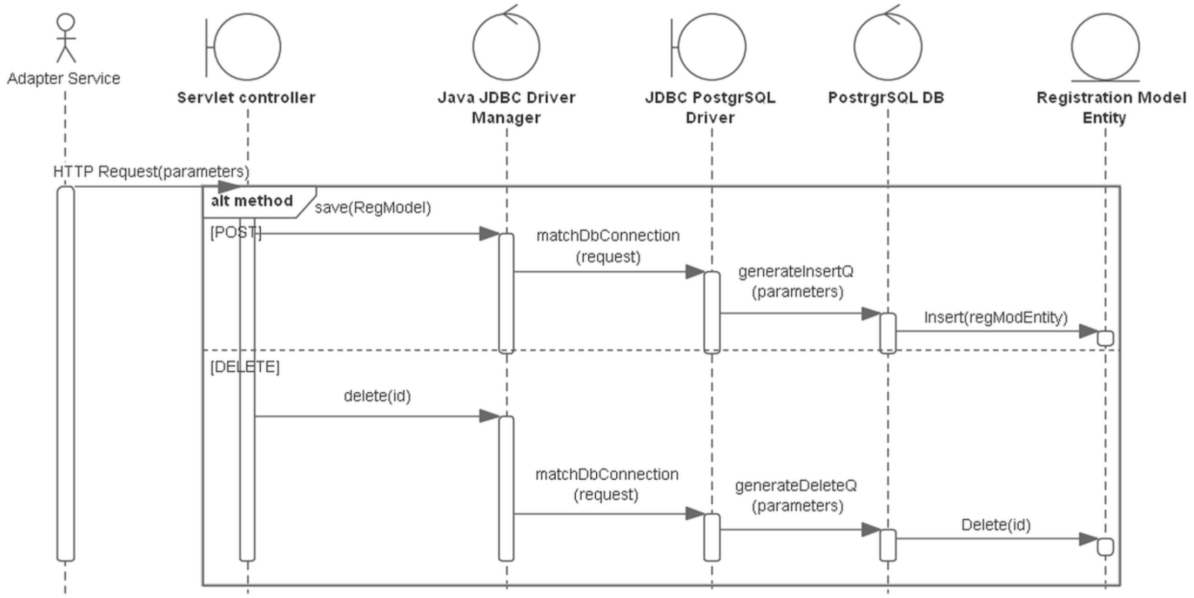


Registration data model diagram.

- Contains a device description.

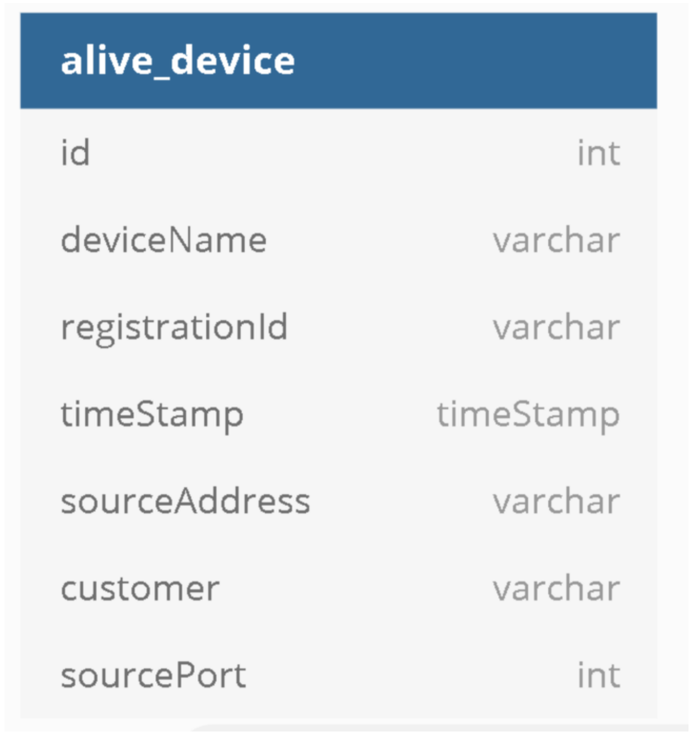
Registration persistence service

- Processes device registration in the system.
- Persists information about connected devices to the system.



Registration sequence diagram.

Alive devices model



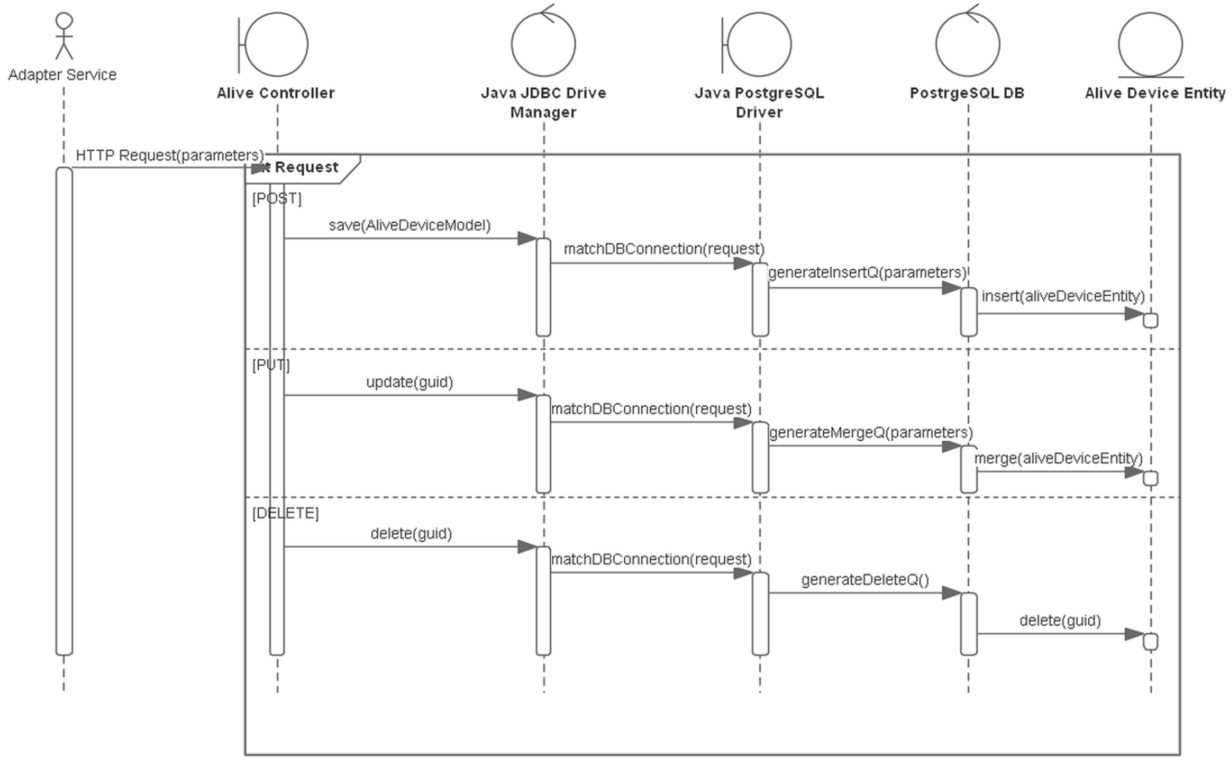
alive_device	
id	int
deviceName	varchar
registrationId	varchar
timeStamp	timeStamp
sourceAddress	varchar
customer	varchar
sourcePort	int

Alive device model diagram.

- Contains information about IoT devices current activity.

Alive devices service

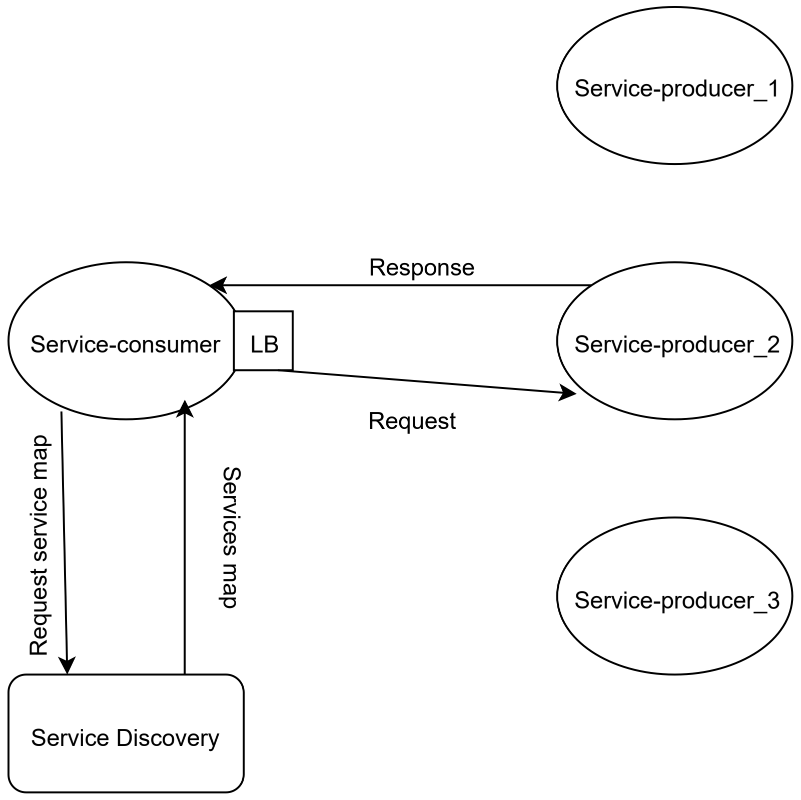
- Ensure alive device entity management



Alive devices service SD.

Client-side LB

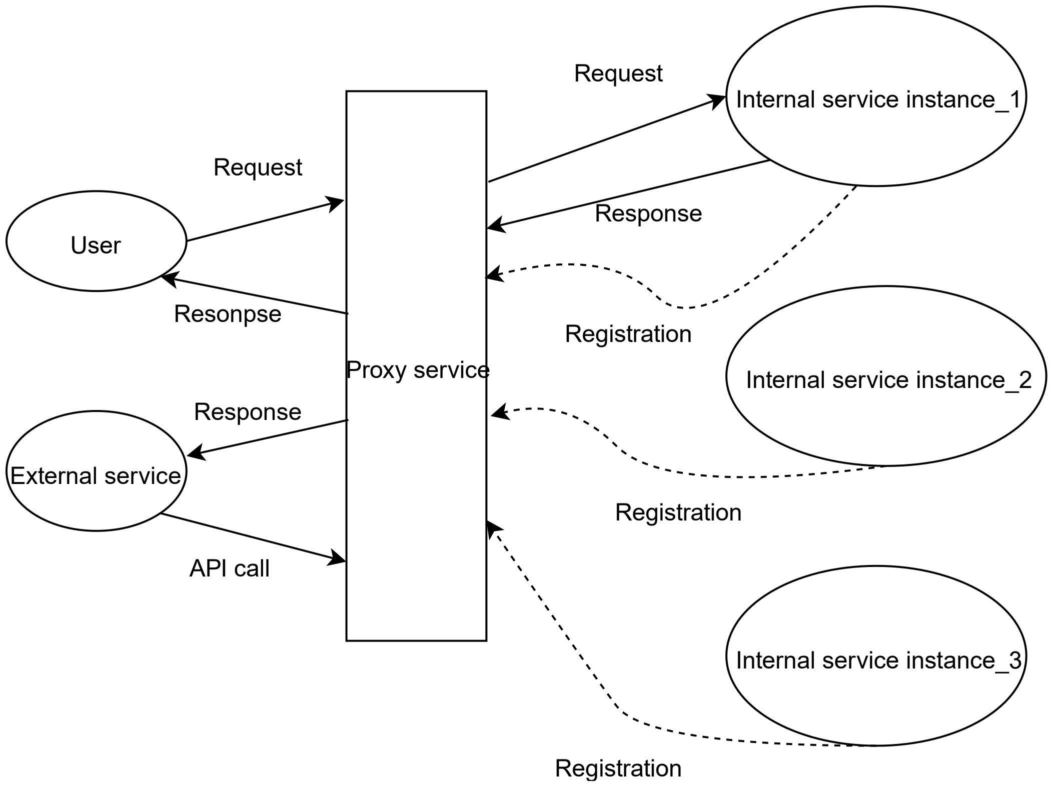
- Each service knows about other service instances.
- Every service instance must connect to service discovery.



Client-side LB diagram.

Server-side LB

- Each service instance knows only URI of requesting service.
- Every service instance has to connect to proxy/gateway service.

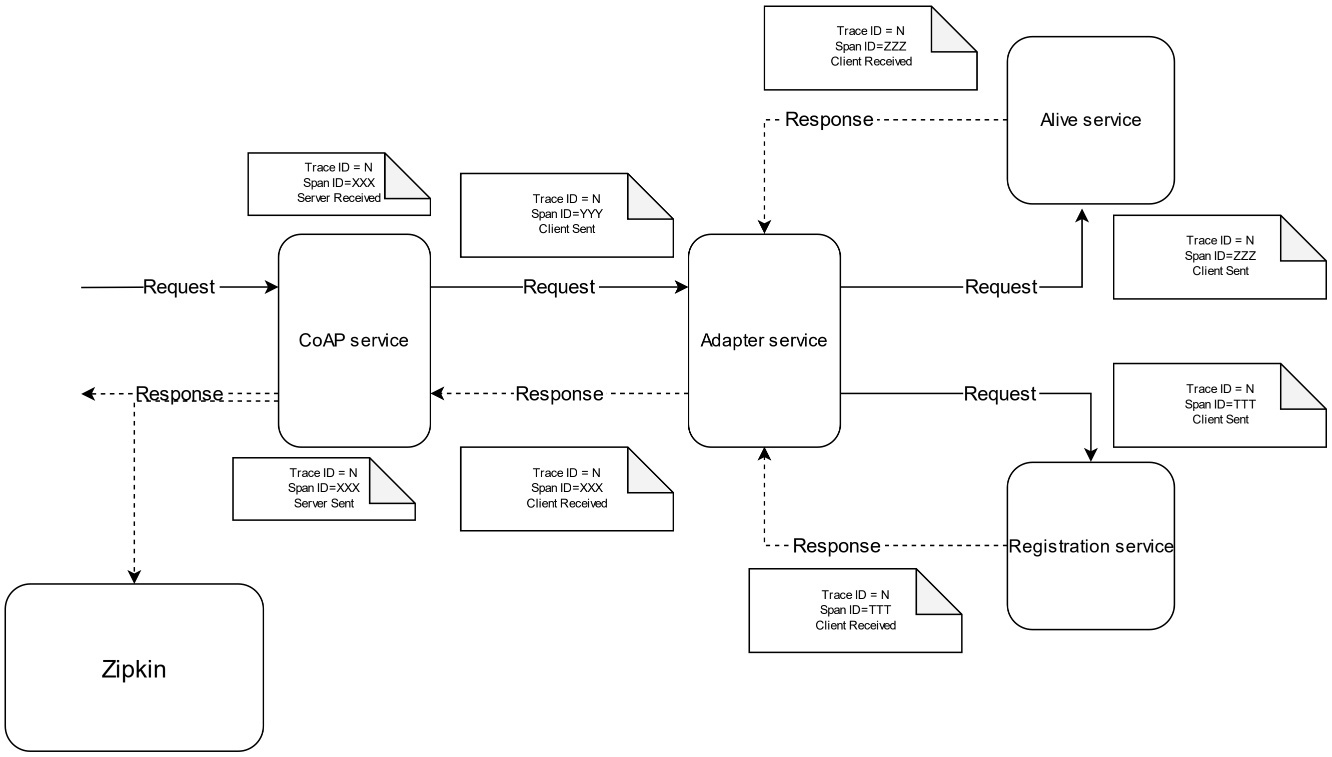


Server-side LB diagram.

Measuring system

- To measure response time we used *distributed tracing* system.
- We stored trace logs in *document-oriented* DB.
- We used Elasticsearch engine to retrieve and analyze logs

Measuring system

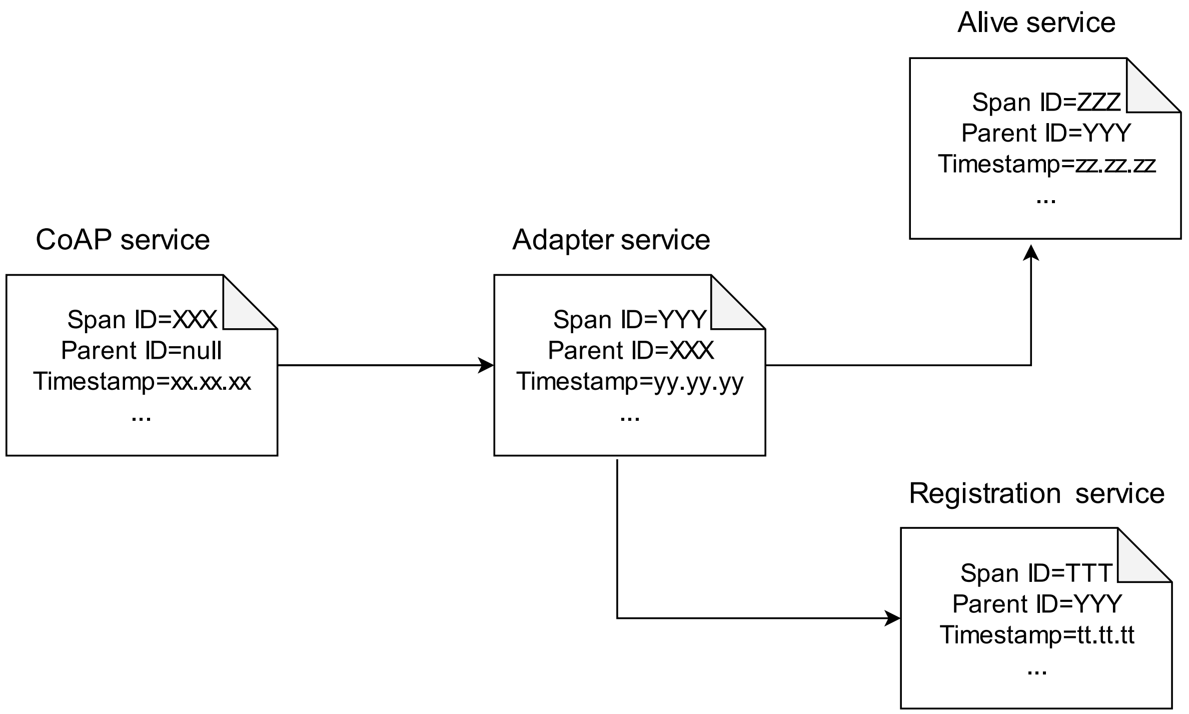


Measuring system diagram.

Measuring system

Each span contains:

- Its ID
- Trace ID
- Parent ID
- Timestamp
- Duration
- Endpoint info
- Custom Tags

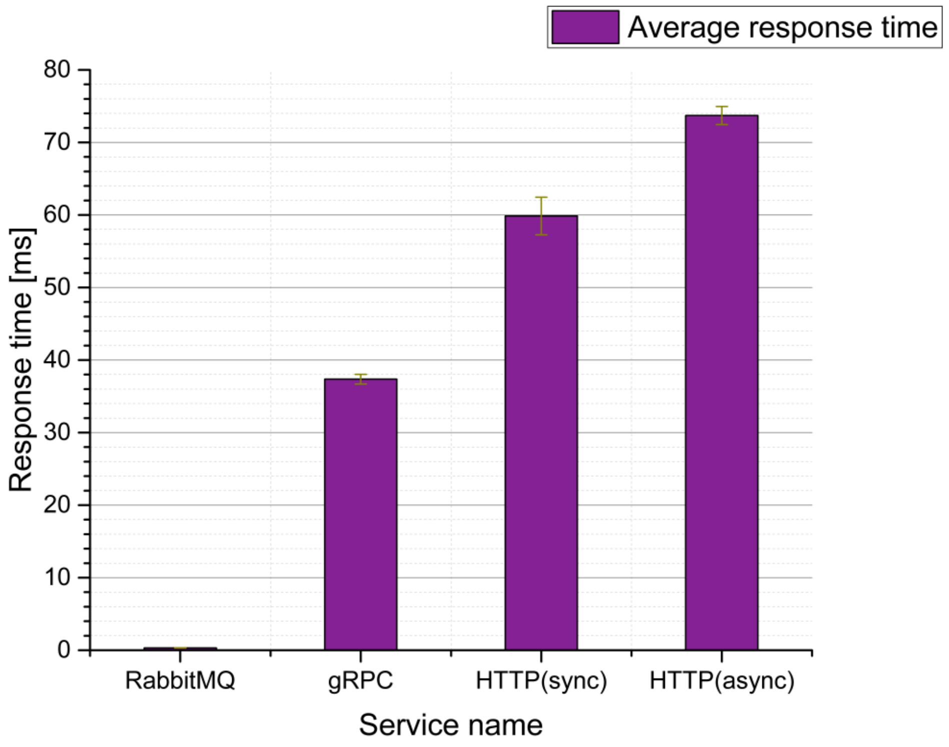


Span child-parent relationship diagram.

Results

Interconnection comparison

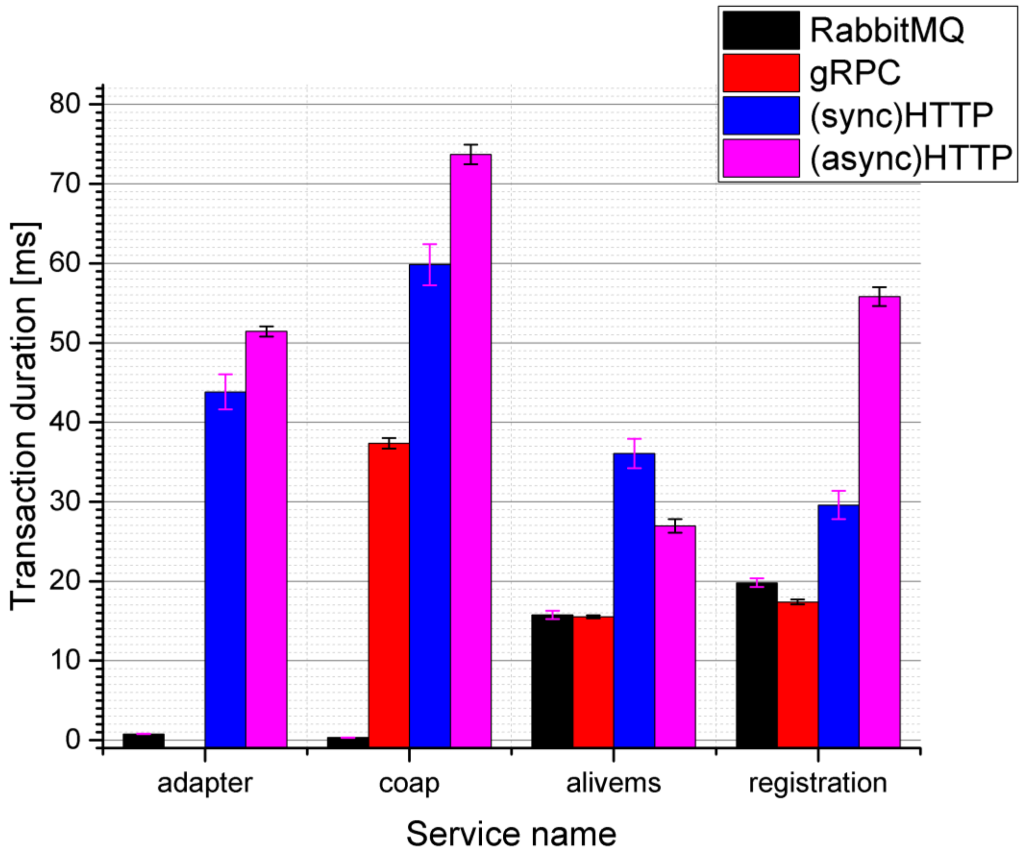
- RabbitMQ provides the lowest response time of less than 1 millisecond.
- Async. HTTP provides the highest response time, a bit higher 70 milliseconds.



Transaction duration by service

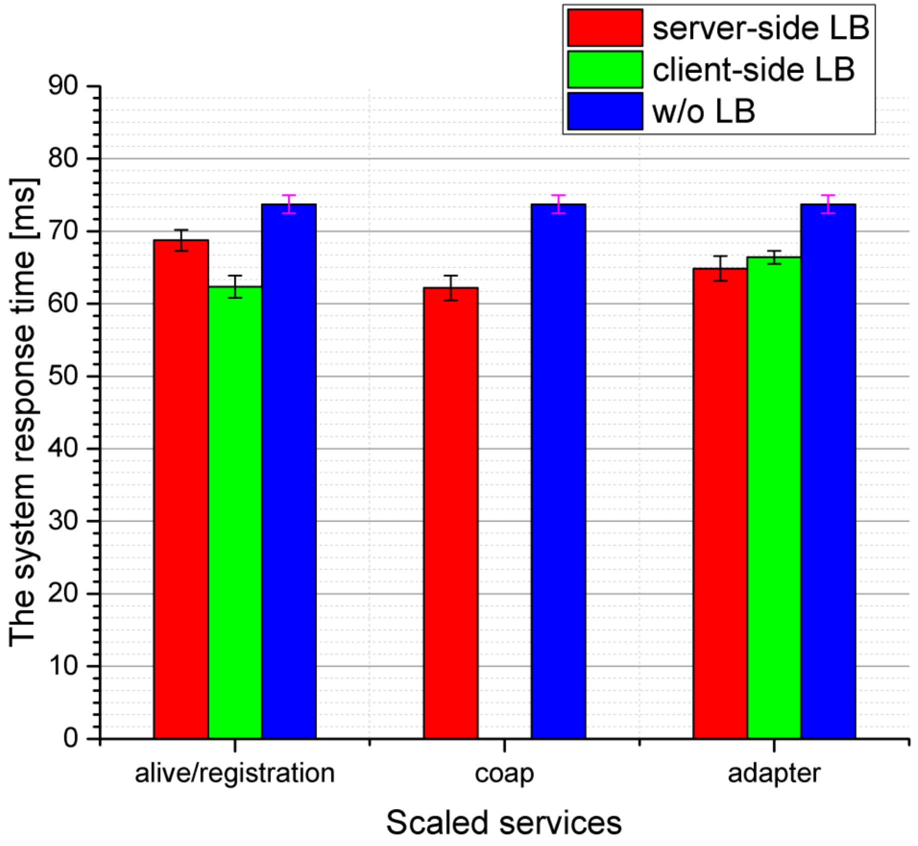
- gRPC interconnection provides lowest transaction duration time of the persistence services.

- RabbitMQ provides the lowest transaction duration time of the non-persistence services.



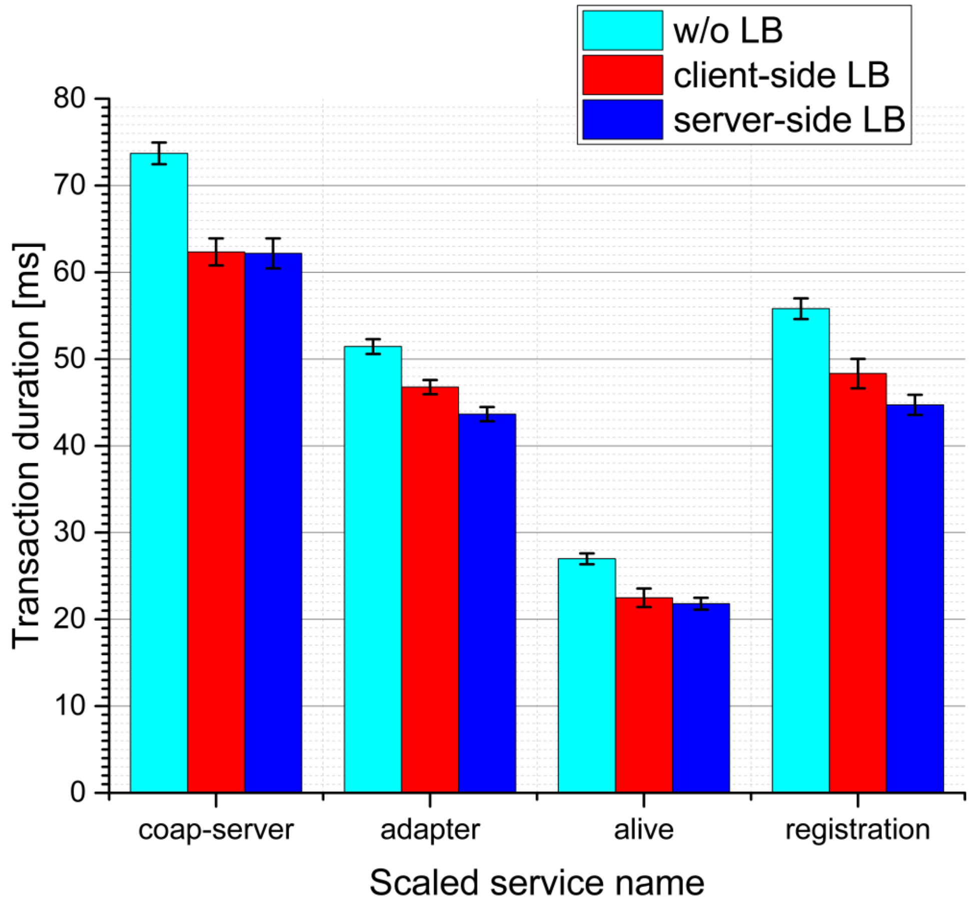
Load balancing strategies comparison

- Client-side LB strategy provides about 10% less a system response time, scaling persistency services.
- Server-side LB strategy improves the system response time by almost the same amount but by scaling edge service.



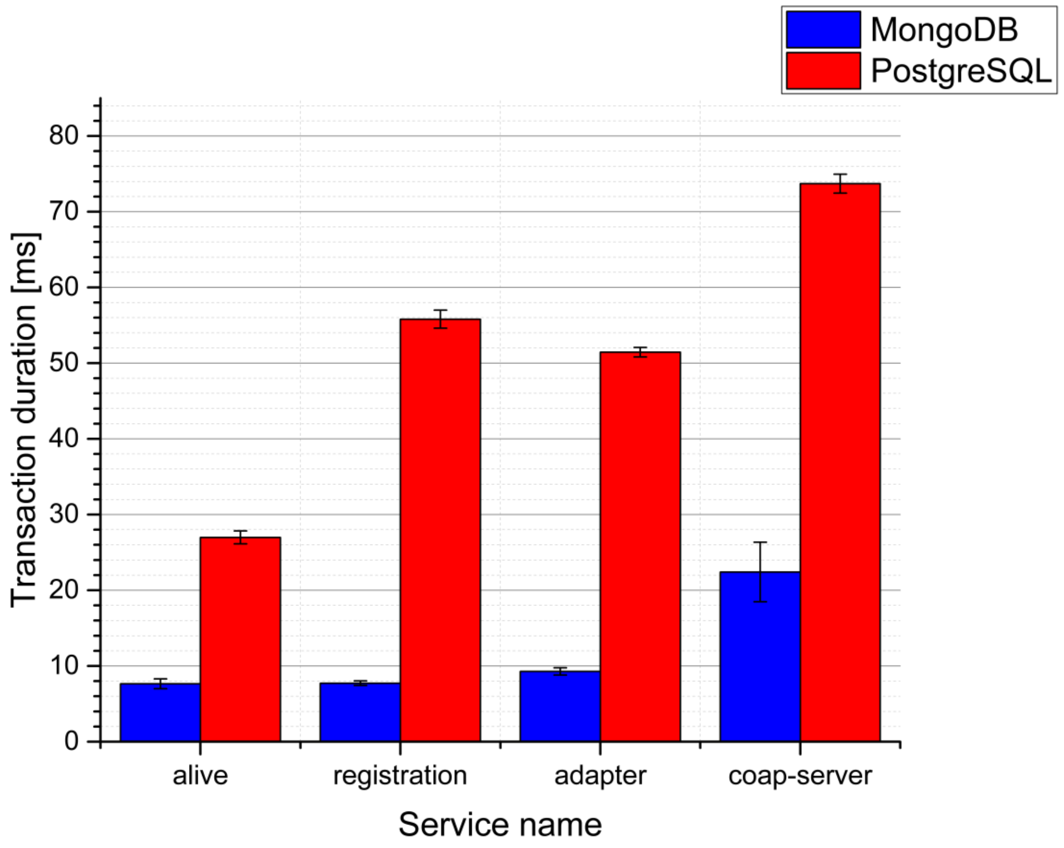
LB Impact on services

- Server-side LB provides slightly faster response time of each service.



DBMS comparison

- MongoDB requires significantly less time to execute database operations.
- To process a write DB operation, it needs 8 milliseconds, approximately.



Conclusion

Conclusion

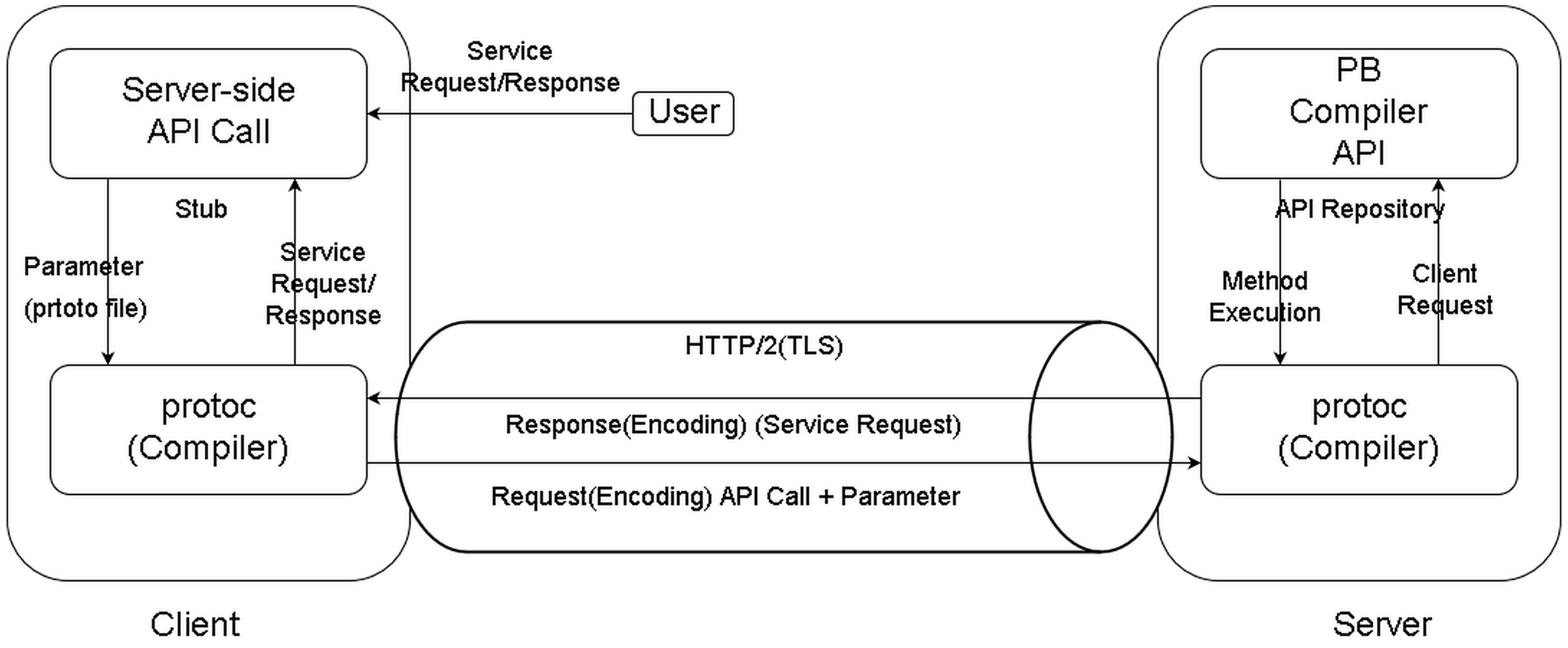
- Connect non-persistence services (CoAP/Adapter) via RabbitMQ and persistence services(Alive/Registration) via gRPC.
- Document-oriented DBMS can ensure the lowest transaction duration of persistence services.
- Load balancing strategy choice depends on which service has to be scaled. Also, server-side one provides a little faster response time.

Future work

- To test the microservice system with some additional requirements.
- Improve the load generator.
- Find out which caching strategy can fit the system in an IoT-contex.
- Prove our research for production usage.

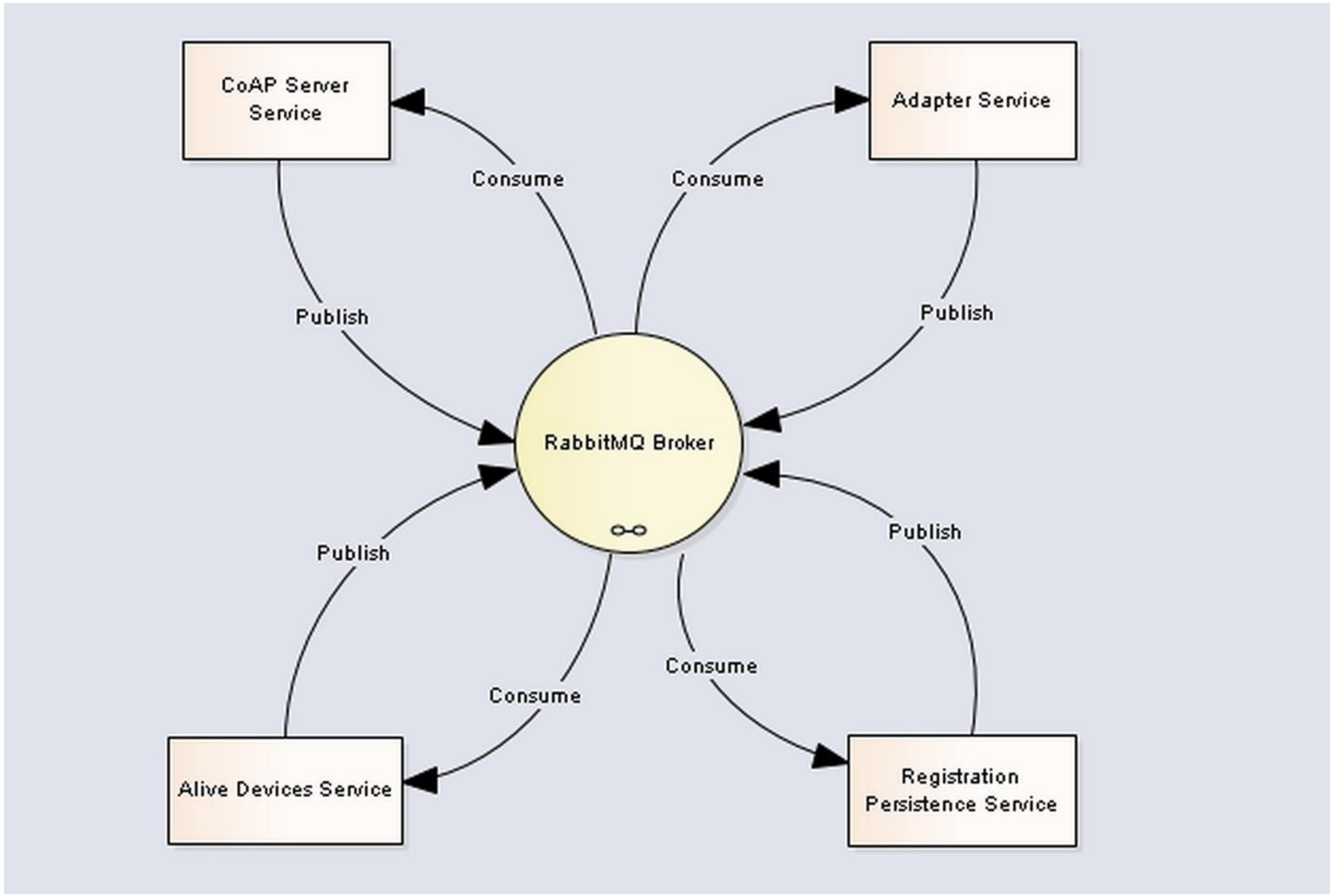
Thank you for your attention

gRPC



GRPC workflow diagram.

RabbitMQ



RabbitMQ dataflow.