Research Document

Scaling Up: microservices

Plamen Peev

Spring 3 deliverable

Contents

PROBLEM STATEMENT

The micro-service architecture is an approach to developing an application as a set of small independent services. Each of the services is running in its own independent process. Services can communicate with some lightweight mechanisms (usually it is something around HTTP)[1]. Such services can be deployed independently in different programming languages, using their own data models and written in a different approach.

The opposite approach is the monolithic architecture. It is an application in which the controllers, services and data models are combined in a singe platform which is deployed as a united solution. So which is better for scaling?

This question is specific for each project individually. If the application isn't too big, a monolithic one is considerably easier to deploy, and this is its biggest advantage.

But there is one problem with this approach, and it is that the application may be considerably complicated, and therefore hard to understand and maintain. The developers' team can't be much flexible, and new members will be introduced to the project slowly.

Flexibility with the technology stack is also reduced, as it is hard to change it.

This solution's greatest disadvantage is that big data consumption cannot be handled properly, which may result in system overloading.

Micro-services architecture gets our attention in the connection with M2M applications. We declare many times, that in our opinion "no one size fits all" in M2M applications [2]

Therefore, microservices are the natural fit for M2M development. Is this solution however suitable for all projects?

QUESTIONS

Main question:

How can microservices improve project quality and scalability?

Sub questions:

- RQ-1: What are the main practical motivations for using microservices?
- RQ-2: What different types of microservice architectures can be involved?
- RQ-3: What are the existing techniques and tooling used to enable microservices architecture and development?

PLANNING

The first step before doing the research was to define main and sub questions. For this purpose, I did some preliminary research on the general topic: Microservices. I check what other researchers and scholars are discussing. After I noted the questions, I evaluated each with the following helping questions:

Is the question clear? ; Is the question focused? ; Is the question focused?

After defining the sub-questions, I followed the same field pattern from the DOT Framework for all the questions: *Field <-> Library*

For the first 2 questions, I used the following methods: *Problem analysis<-> Community research*

For the last question, I used *Design pattern research* instead of community.

RESULTS

RQ-01: Main practical motivations

Scalability, delegation of responsibilities to independent teams, and the easy support for DevOps also frequently drive adoption, while other motivations were only reported by migration consultants. One interesting observation is that several practitioners reported adopting microservices-based architectures because a lot of other companies are adopting them.

Main motivations: [3]

- *Maintainability*

  Breaking a system into independent and self-deployable services enables developer teams to make changes and test their service independent of other developers, which simplifies dis-tributed development. Breaking the size to microservices improves code readability vastly, therefore making it easier for developers to improve and maintain their assignment easily and independently.

- *DevOps Support*

  The adoption of a DevOps toolchain is supported by microservices. Since each ser-vice can be developed and deployed independently, each team can easily develop, test, and deploy their services independent of other teams.

- *Scalability*

  Scaling monolithic systems requires huge investment in terms of hardware and often fine-tuning of the code.

  On the contrary, for a microservice-based system, each microservice can be deployed on a different server, written in the post appropriate language. This way we prevent deploying the system in one powerful machine.

- *Delegation of Teams Responsibilities*

  Since microservices do not have external dependencies, they can be developed by different teams independently, reducing communication overhead and the need for coordination among teams. Each team owns the code base and can be responsible for the development of each service; it can maintain independent revisions and build environments based on their needs.

RQ-02: What different types of microservice architectures can be involved?

In order to explore the different architecture methods, we first have to explain 2 general points. Microservice states, and microservices architecture primitive needs. After these topics are cleared, we will then look into the communication patterns in microservices architecture

States

There are two types (states) of microservice: “stateless” and “stateful”. [4]

- *Stateful microservices*

  Has saved data in a database that they manage directly. Usually they don’t tend to share databases with other microservices because it leads to harder maintenance and well-defined interfaces. When a stateful service terminates it has to save its state.

- *Stateless microservices*

  Don’t manage data. They only handle requests and return responses. They don’t have any permanent access to data, but rather they are supplied with it in a request, and once the actions are taken, they forget the data. When a stateless service terminates it has nothing to save.

Needs

We can mention the following primitives need for microservices architecture [5]:

1. Request/Response calls with arbitrary structured data

2. Asynchronous events should be flowing in real-time in both directions

3. Requests and responses can flow in any direction,

4. Requests and responses and can be arbitrarily nested. The typical example is a self-registering worker model

5. A message serialization format should be pluggable, so developers may use, for example, JSON, XML, etc.

Communication

Services and clients have different context and goals, so targeting communication through a variety of mechanisms. Depending on the protocol, there are two types of communication [6]:

- *Asynchronous Communication*

  Callers need not have the specific destination of the callee. Handling multiple consumers at a time becomes relatively easy (as services may add up consumers). Moreover, the message queues up if the receiving service is down & proceeds later when they are up. This is particularly important from the perspective of loose coupling, multi-service communication, and coping up with partial server failure. These are determining factors for inclining microservices towards Async communication.

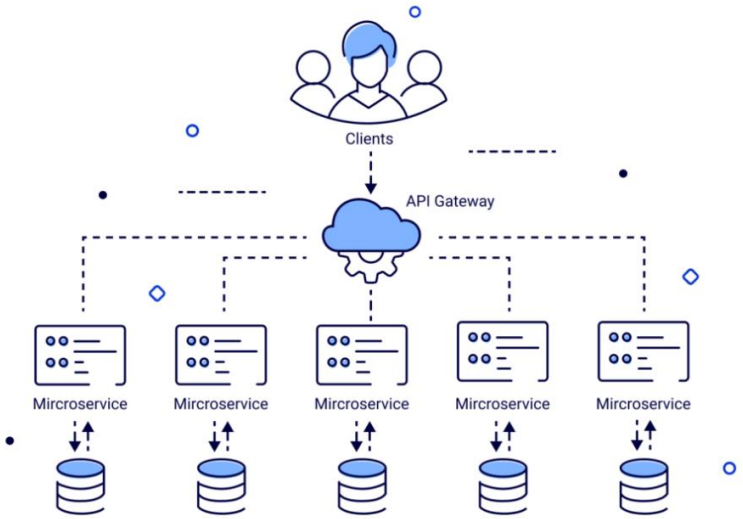
- *Synchronous Communication*

  a predefined source service address required, where exactly to send the request, and BOTH the service (caller and callee) should be up and running at the moment. Though Protocol may be synchronous, I/O operation can be asynchronous where the client need not necessarily wait for the response. This is a difference in I/O and Protocol. The common request-response approach common to web API includes REST, GraphQL, and gRPC.

RQ-03: What are the existing techniques and patterns used to enable microservices architecture and development?

Microservice Architecture

Overview



Applications running on microservices architecture are deconstructed and separated into smaller units (microservices) that the client can access using an API gateway. These microservices are each independently deployable but can communicate with one another when necessary. Microservices architecture focuses on classifying the otherwise large, bulky applications. Each microservice is designed to address an application's particular aspect and function, such as logging, data search, and more. Multiple such microservices come together to form one efficient application. The client can use the user interface to generate requests. At the same time, one or more microservices are commissioned through the API gateway to perform the requested task. As a result, even larger complex problems that require a combination of microservices can be solved relatively easily.[7]

Design patterns

The microservices design patterns can be divided into five main pattern types. Each contains many patterns.

We will look at the most used patterns per pattern type, as explaining all the patterns is out of this research's topic.

- Decomposition patterns
  - Decompose by Business Capability

    Microservices is all about making services loosely coupled, applying the single responsibility principle. It decomposes by business capability. Define services corresponding to business capabilities. A business capability is a concept from business architecture modeling[8]
  - Decompose by Subdomain

    Define services corresponding to Domain-Driven Design (DDD) subdomains. DDD refers to the application's problem space — the business — as the domain. A domain is consists of multiple subdomains. Each subdomain corresponds to a different part of the business.
  - Decompose by Transactions

    You can decompose services over the transactions. Then there will be multiple transactions in the system. One of the important participants in a distributed transaction is the transaction coordinator. [9]
- Integration patterns
  - API Gateway Pattern

    The pattern provides a reverse proxy to redirect or route requests to your internal microservices endpoints. An API gateway provides a single endpoint or URL for the client applications, and it internally maps the requests to internal microservices. A layer of abstraction is provided by hiding certain implementation details. [10]

- Aggregator Pattern

  Implementing a service that receives a request, then makes requests of multiple services, combines the results and responds to the initiating request.

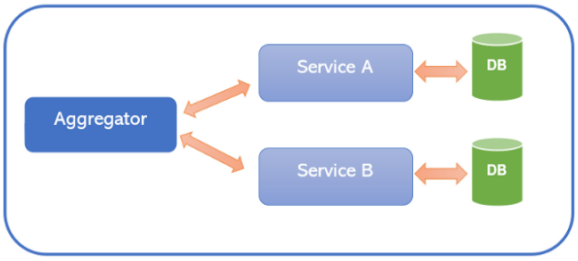


Image: Representation of Aggregator Pattern

- Database patterns
  - Database per Service

    Designing one database per service; it is private to that service only. It should be accessed by the microservice API only. It cannot be accessed by other services directly.

- Observability patterns
  - Log Aggregation [11]

    Consider a use case where an application consists of multiple services. Requests often span multiple service instances. Each service instance generates a log file in a standardized format. We need a centralized logging service that aggregates logs from each service instance. Users can search and analyze the logs. They can configure alerts that are triggered when certain messages appear in the logs.

- Performance Metrics [11]

  A metrics service is required to gather statistics about individual operations. It should aggregate the metrics of an application service, which provides reporting and alerting. There are two models for aggregating metrics:

  Push — the service pushes metrics to the metrics service e.g. NewRelic, AppDynamics

  Pull — the metrics services pulls metrics from the service e.g. Prometheus

CONCLUSION

Weather or not to implement microservices architecture to a project is a very sensitive topic and it is concerning many factors. As seen from this research, by implementing it we can separate functionality and make the application vastly more scalable than with using a monolithic approach. But, we have to really think if it is necessary to use microservices. If the software system would be built by a large team, and there are many constrains concerning separate functional blocks, microservice architecture is definitely going to improve the overall system and team performance.

Based on the research, my individual project ‘iTrips’ is perfectly suitable for microservices architecture. For example, the variety of design patterns is a great benefit, as in my project there are 4 different types of users, several different functional block for each, and this continues like branches form a tree. Switching from monolithic application to microservices will make the development considerably easier if this was a real project which should be deployed.

There is, however, one big obstacle. This project is individual, therefore implementing microservices will only complicate the process, thus I won’t be using this approach.

References

[1] - Uckelmann, Dieter, Harrison, and Michahelles. (2011). "An architectural approach towards the future internet of things." 1-18.

[2] - Namiot, D., & Sneps-Sneppe, M. ;(2014). “On M2M Software Platforms. International Journal of Open Information Technologies”, 2, 27-31

[3] - Davide Taibi, Valentina Lenarduzzi, and Claus Pahl (2016) “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation”, 20-26

[4] - Anne Currie (2017) “Microservices - Thinking Outside the Lines”

[5] – Libchan, https://github.com/docker/libchan

[6] – Bibek Shan (2020) “Microservice Architecture — Communication & Design Patterns”

[7] – Middleware (2021), https://middleware.io/blog/microservices-architecture/

[8] - https://microservices.io/patterns/decomposition/decompose-by-business-capability.html

[9] - https://www.baeldung.com/transactions-across-microservices

[10] - https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-integrating-microservices/api-gateway-pattern.html

[11] – Madhuna Udantha (2019), “Microservice Architecture and Design Patterns for Microservices”