



To the Cloud with Crosswind

Breaking down a monolith to microservices with the strangler pattern

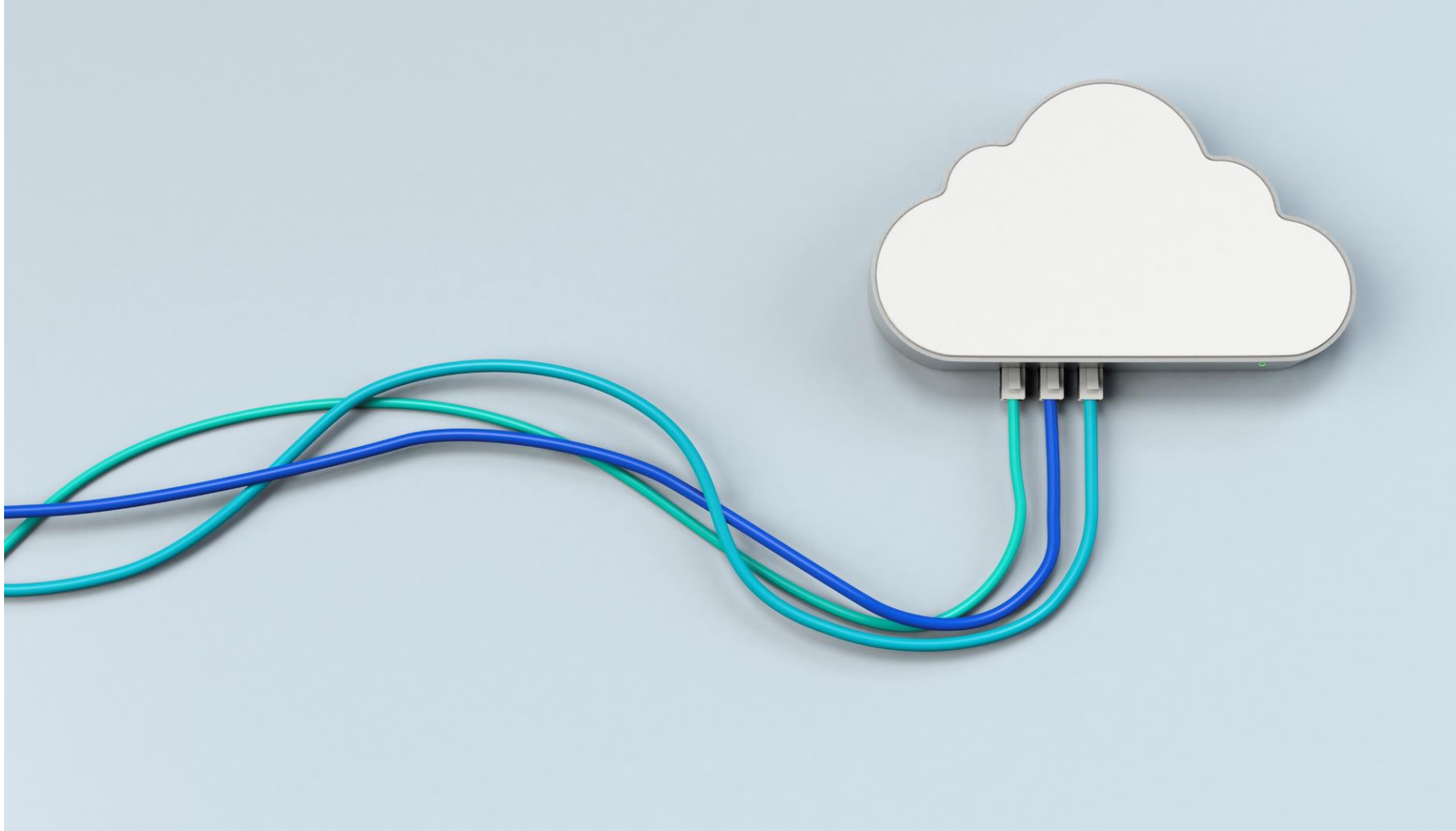
Who am I?

Java Developer and Application Architect at NTT DATA

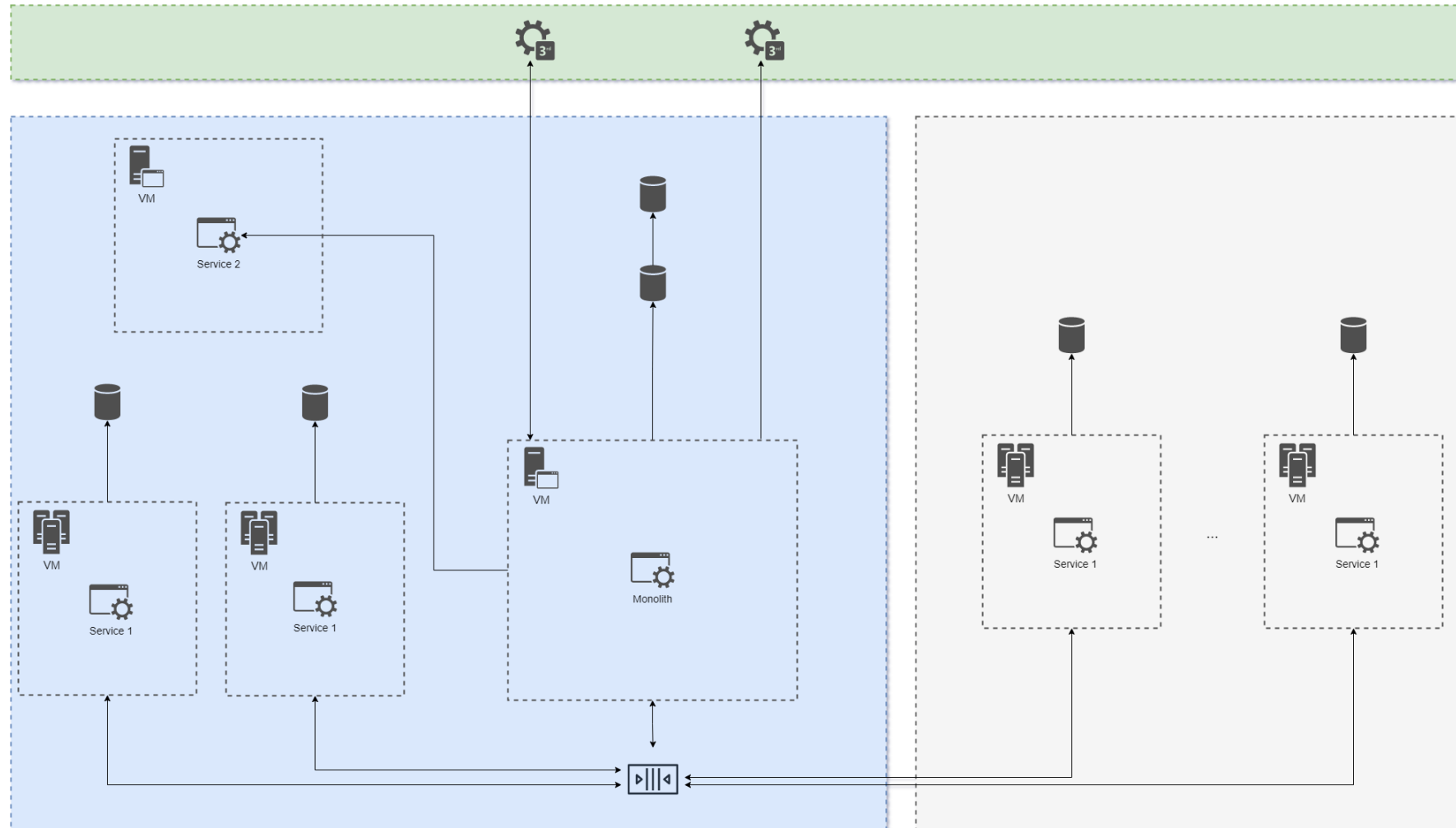
I am a Java developer for 12 years with focus on Java Enterprise back-end development. With time I've noticed I have a knack for understanding complex systems and have thus shifted my attention to architecture and the evolution of complex systems over time. In my spare time I enjoy hiking and landscape photography.



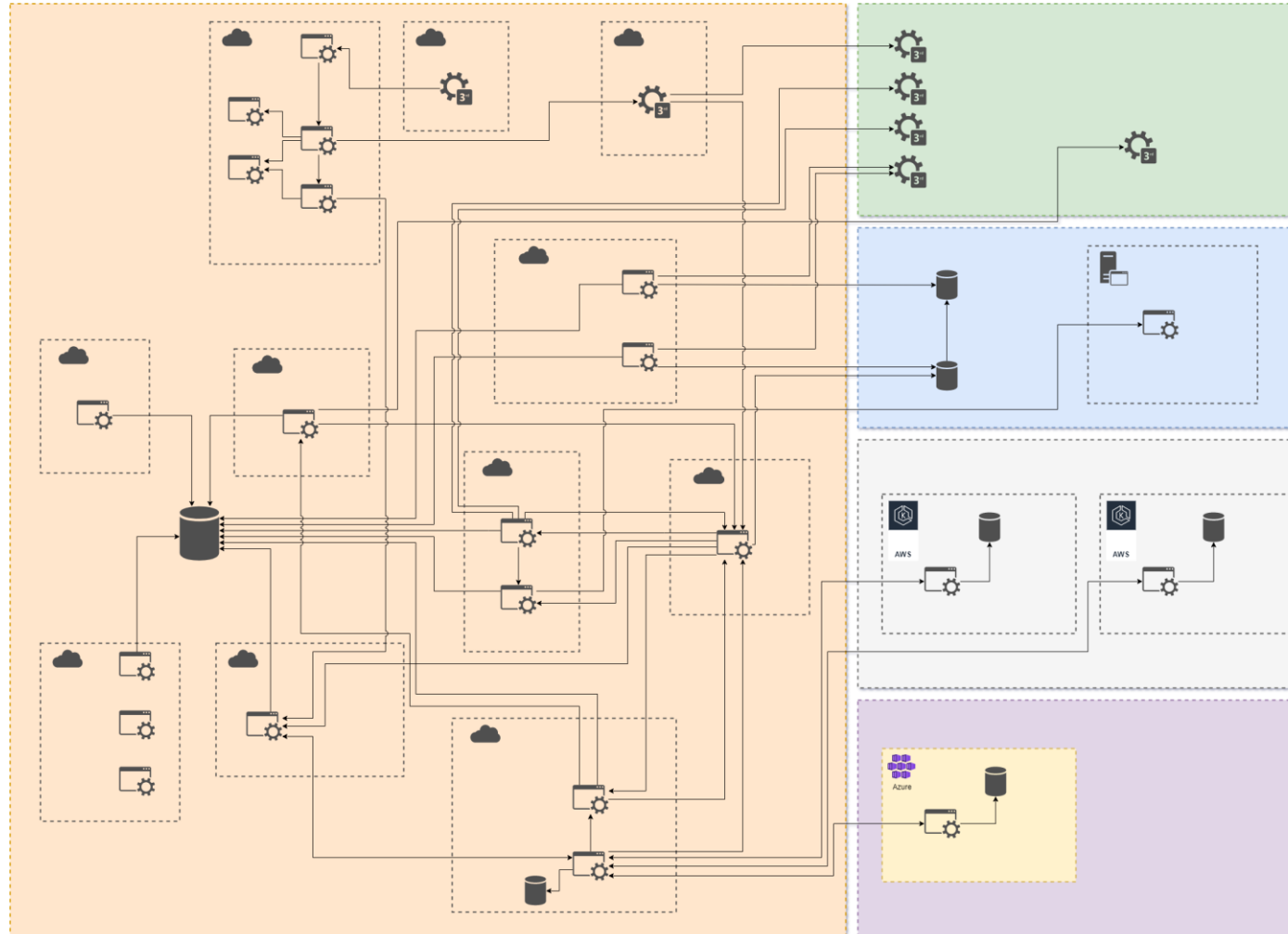
Let's take a journey to the cloud!



How it all started



How it's going



TOPICS

1

Microservices

2

Monolith

3

Are microservice the right choice?

4

Splitting the monolith

5

Splitting the database

6

Data or code? What do I split first?

7

Dealing with new features

8

People, Tools and Processes

9

Summary and Conclusion

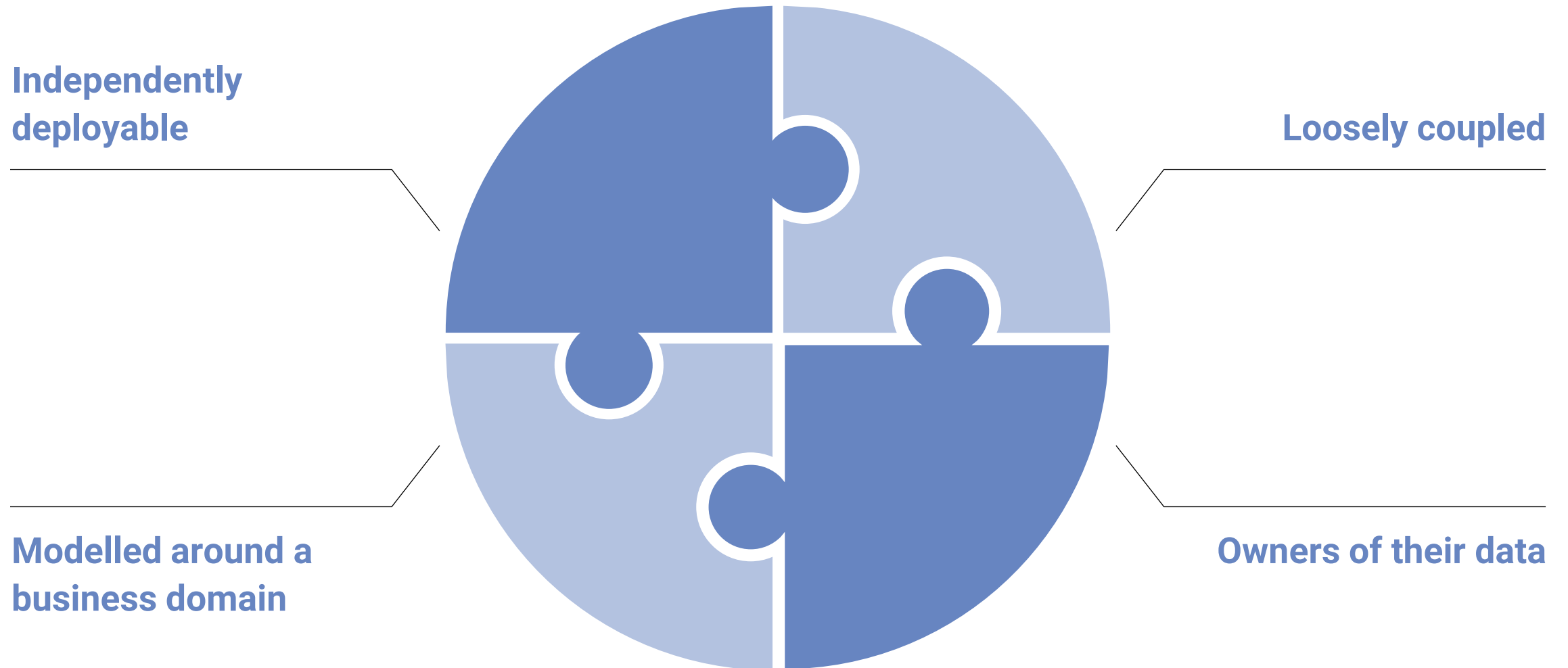
10

Questions

Microservices

What are they and how can they help?

What defines a microservice?



Microservices - The good and the bad



- **Independently deployable**
- Scalable
- Robust
- Easy to understand
- Easy to change
- **Allow for different technological choices**

Keyword: Flexibility



- **Communication over network**
 - Latency
 - Failure handling
- Complex interactions
- **Complex transaction handling**
- Question of ownership
- Prone to cargo cult programming

Keyword: Overhead

How small is a microservice?

Doesn't matter!

Focus on:

Independent deployability

Loose coupling

Modeling around business domain

Data ownership

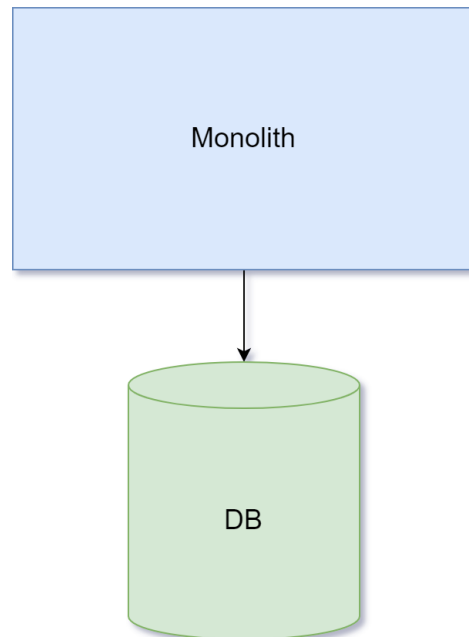


Monolith

Should we really bury it in programming history?

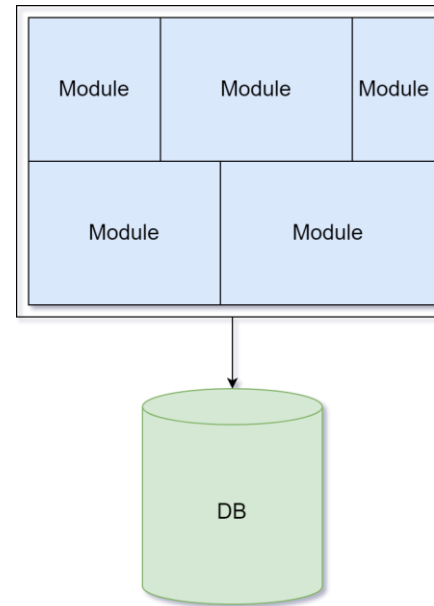
What is a monolith?

- Single unified software application
- Self-contained and independent
- Single unit of deployment

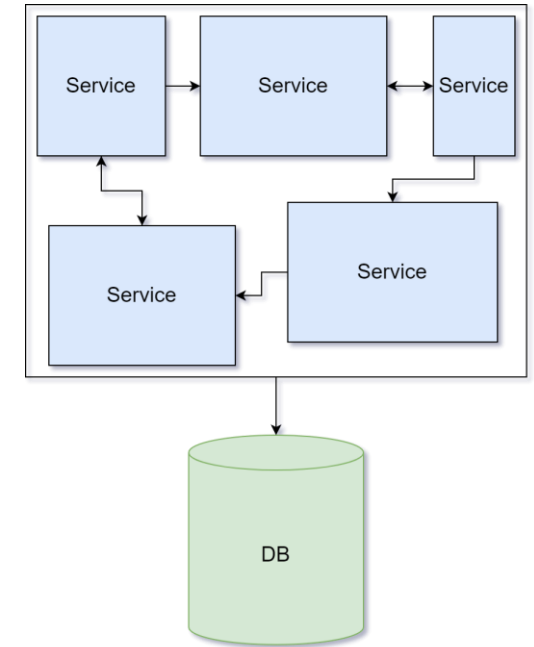


Types of Monoliths

- Single process monolith
- **Modular monolith**
- **Distributed monolith**
- 3rd party blackbox



Modular monolith



Distributed monolith

[Sam Newman "Monolith to Microservices" \(O'Reilly Media, Inc.\)](#)

Monolith - The good and the bad



- **Easy to deploy**
- **Easy E2E testing**
- Simplifies code reuse
- Less architectural overhead
- Less infrastructure overhead

Keyword: Less overhead



- **Coupling and cohesion**
- Stepping on others' feet
- Ownership
- **Single technology stack**
- Difficult to understand
- **Difficult to change**

Keyword: Complexity

Decisions

Should I switch to a microservice architecture?

prime video | TECH

Video Streaming

Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%

The move from a distributed microservices architecture to a monolith application helped achieve higher scale, resilience, and reduce costs.

Marcin Kolny
Mar 22, 2023

<https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90>



zensoftware

Cloud & Off-Prem Development

#microservices, #monolith, #amazon, #architecture

Amazon Prime Video swaps Microservices for Monolith: 90% Cost Reduction

Published on 17 May 2023 by Arjan Franzen

Microservices  Monolith 

prime video

<https://www.zensoftware.nl/en/news/amazon-prime-video-swaps-microservices-for-monolith-90-cost-reduction>

THE NEW STACK

CLOUD SERVICES / MICROSERVICES / SERVERLESS

Return of the Monolith: Amazon Dumps Microservices for Video Monitoring

A blog post from the engineering team at Amazon Prime Video has been roiling the cloud native computing community with its explanation that, at least in the case of the video monitoring, a monolithic architecture has produced superior performance over a microservices and serverless-led approach.

May 4th, 2023 7:23am by Joab Jackson

<https://thenewstack.io/return-of-the-monolith-amazon-dumps-microservices-for-video-monitoring/>

CLOUD — FEATURED — READ THIS — AWS — MICROSERVICES — PRIME VIDEO — SERVERLESS

Amazon Prime Video team throws AWS Serverless under a bus

Monoliths are sexy again.

ED TARGETT

May 4, 2023 . 7:52 PM — 4 min read

<https://www.thestack.technology/amazon-prime-video-microservices-monolith/>

amplification

v1.7.6

Home / Blog / Amazon Ditches Microservices for Monolith: Decoding Prime Video's Architectural Shift

Amazon Ditches Microservices for Monolith: Decoding Prime Video's Architectural Shift

Michael Solati

May 18, 2023

<https://amplification.com/blog/amazon-ditches-microservices-for-monolith-decoding-prime-videos-architectural-shift>

So, why still choose microservices?

Switching to a microservice architecture must be a conscious and deliberate decision.



So, why still choose microservices?



Good reasons

- Team autonomy
- Improve scaling and robustness
- Faster and more granular rollouts
- Resource distribution
- Improve efficiency with use of new technology



Bad reasons

- Expected increase in performance
- Don't understand domain
- Working in/for a small company
- It's industry standard
- I've been told to do it

My decision process



Distribute system load



Reduce resource contention



Improve robustness



Improve developer experience



Update technology stack



Splitting the Monolith

This is where the fun starts



Where to start?

- **Start small and work in iterations**
- Identify domains and boundaries
- Prioritize simple and well isolated modules
- Consider starting with a modular monolith
- **Favor copying over cutting or reimplementing**
- **Favor moving new functionality to microservices**
- Evaluate your progress constantly
- Improve based on previous experience

Examples

Reusable configuration calculator

- Different department required access to monolith functionality
- Extracted functionality and provided it to the new consumer
- Existing consumer was still using monolith
- Eventually switched all consumers to new service

Log Archiver

- Monolith was storing special audit logs
- Customer required long term storage of audit logs
- Archiver implemented as service separate from monolith

Patterns: Strangler Fig Application

- Term initially used by Martin Fowler to describe this pattern
- Type of fig that wraps around host tree replacing it in the end
- In Software:
 - The new system is initially supported by the existing monolith
 - Old and new coexist until eventually the old gets replaced
- Great for extracting externally exposed modules

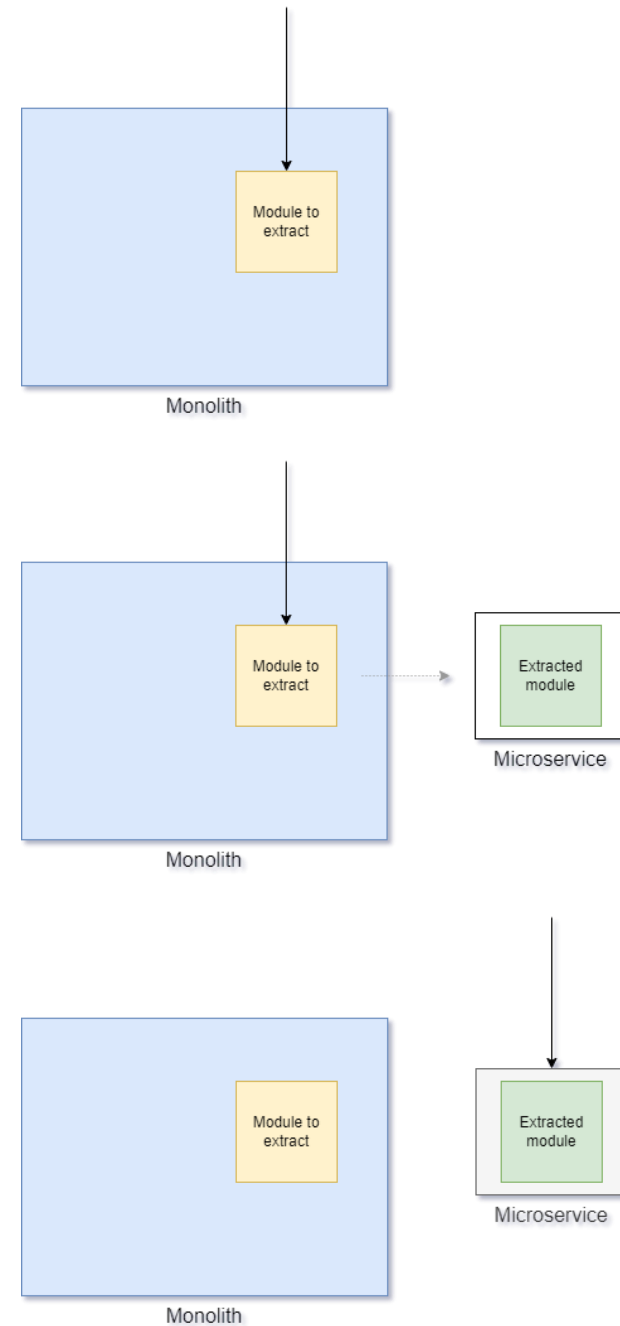
<https://martinfowler.com/bliki/StranglerFigApplication.html>



Patterns: Strangler Fig Application

Step by step

1. Decide what to module to extract
2. Extract the module as a microservice
3. Redirect calls to the new module



Patterns: Strangler Fig Application

Benefits

- Incremental approach to monolith split
- Allows deployment and release separation
- **Fallback to original implementation possible anytime**
- Microservice validation is easy

Patterns: Strangler Fig Application

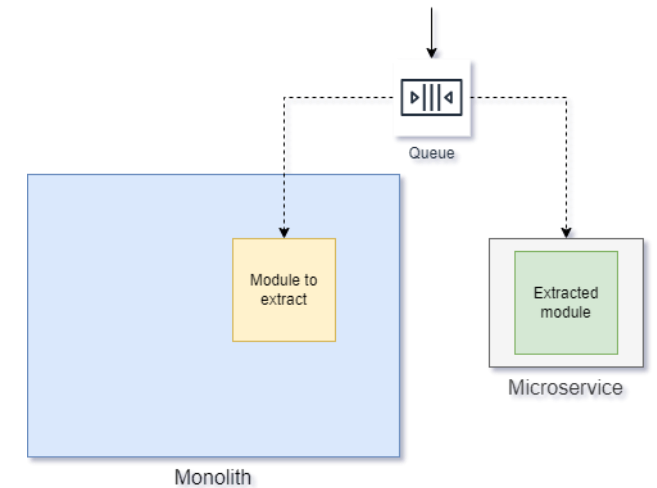
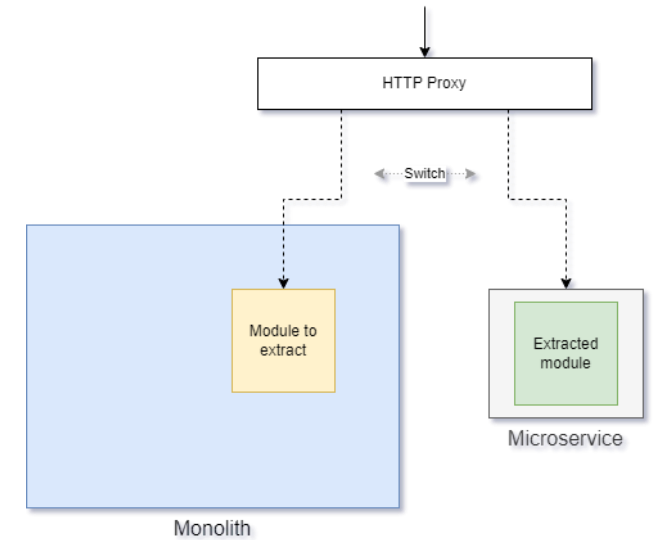
Implementation Examples

■ HTTP Reverse Proxy

- Insert a reverse proxy between clients and monolith
- Redirect the calls via the proxy when going live with the microservice
- Clients only need to change at most once to the proxy

■ Message Interceptor

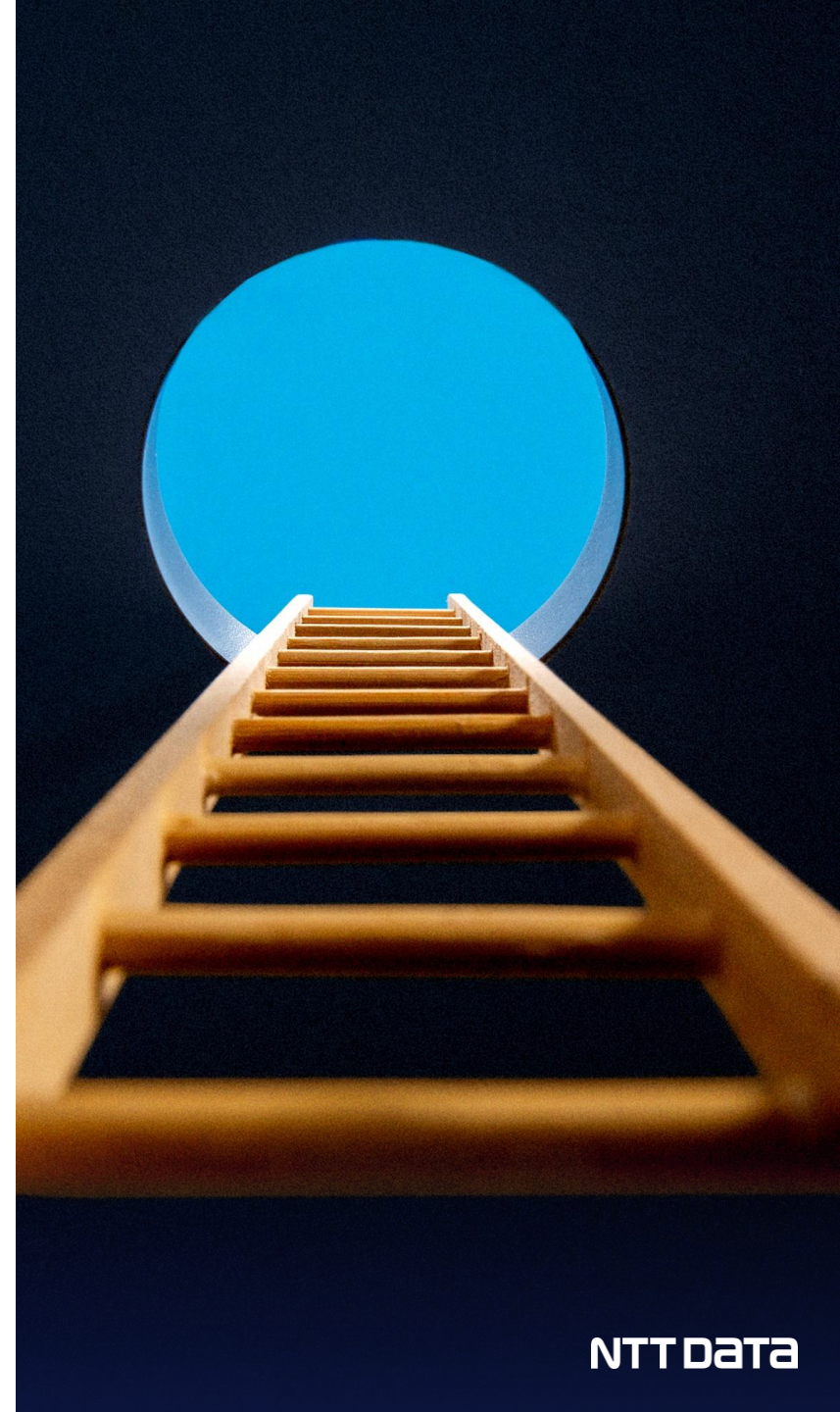
- Based on Queues and Topics
- Simple if infrastructure already in place
- Decoupled approach simplifies pattern application



Patterns: Strangler Fig Application

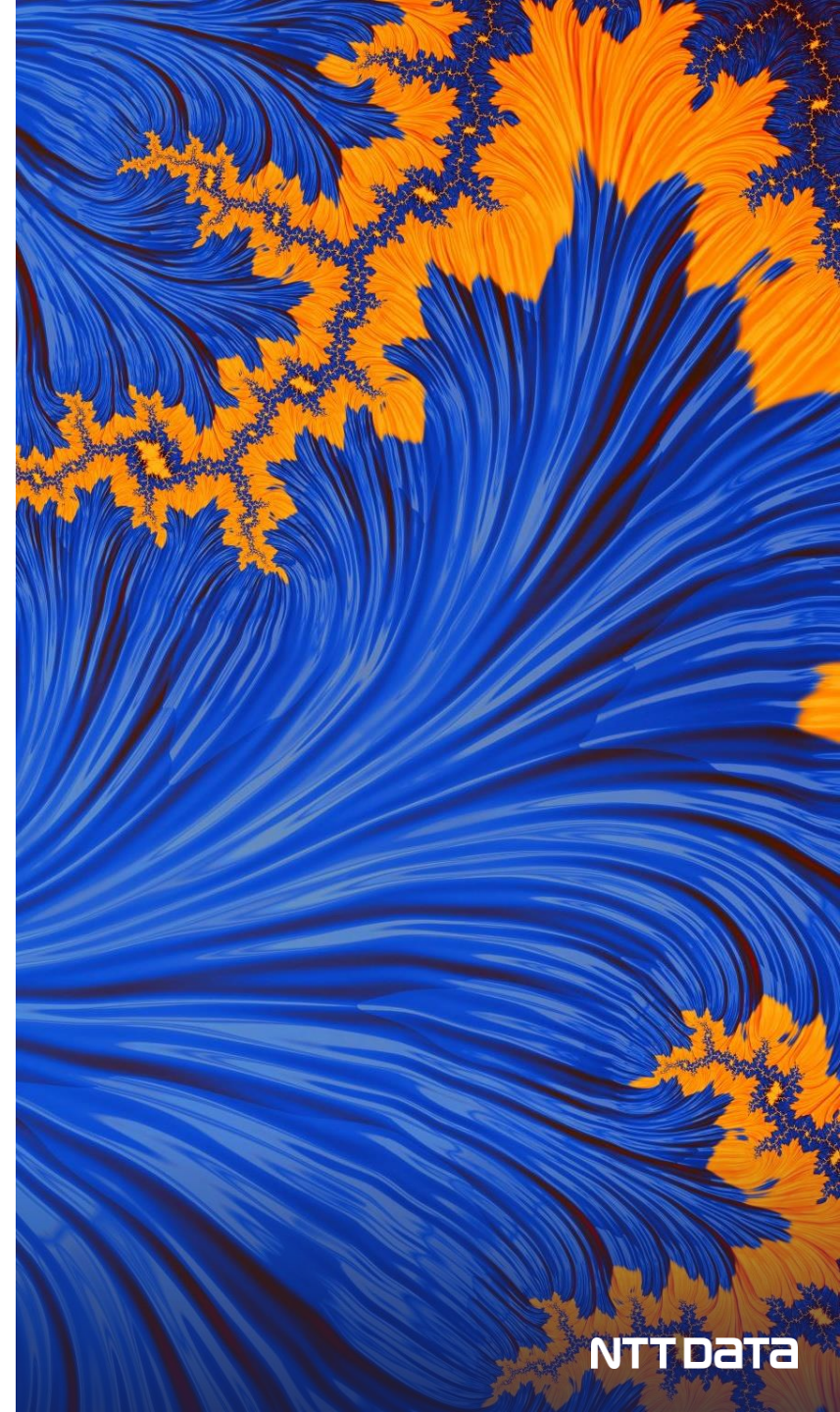
Common Pitfalls

- Forgetting about added latency
- Creating complex logic in the middleware
- Creating your own proxy



Patterns: Branch by abstraction

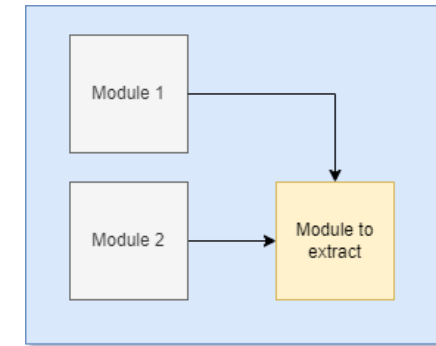
- Aimed at situations where functionality is deep inside the monolith
- Reduces code contention with other developers
- Removes the need for long living branches



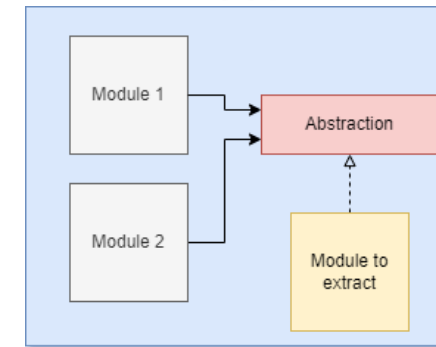
Patterns: Branch by abstraction

Step by step

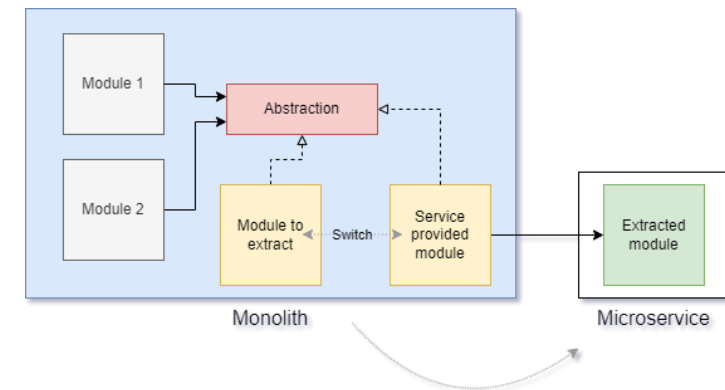
1. Decide what to extract
2. Create and use abstraction
3. Create new implementation and switch concept
 - Switch example: feature flag
4. Clean up



Monolith



Monolith



Monolith

Microservice

Patterns: Branch by abstraction

Benefits

- Fallback to original implementation is very simple
- **Causes minimal disruption**
- Offers the possibility to shift only parts of the extracted module
- Great when dealing with background processes(ex. jobs) nested inside the monolith



Patterns: Parallel run

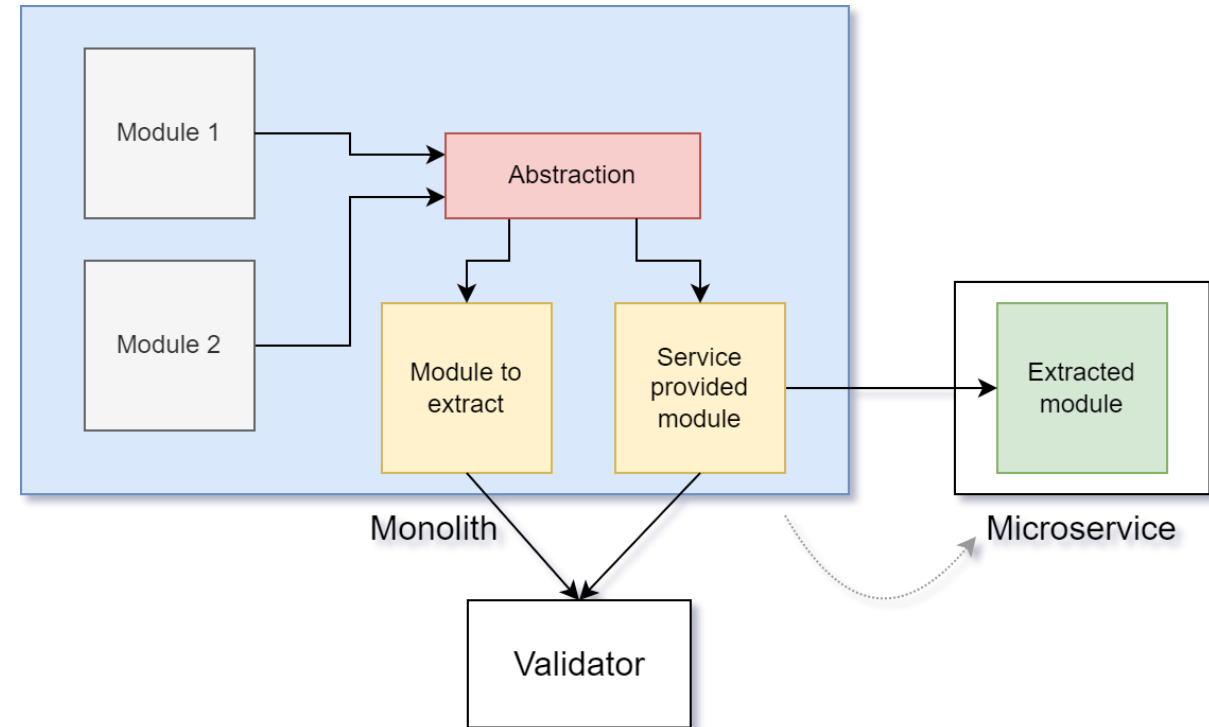
- Runs old and new implementation simultaneously
- Useful when dealing with critical functionality
- Allows for comparison between existing and new implementation



Patterns: Parallel run

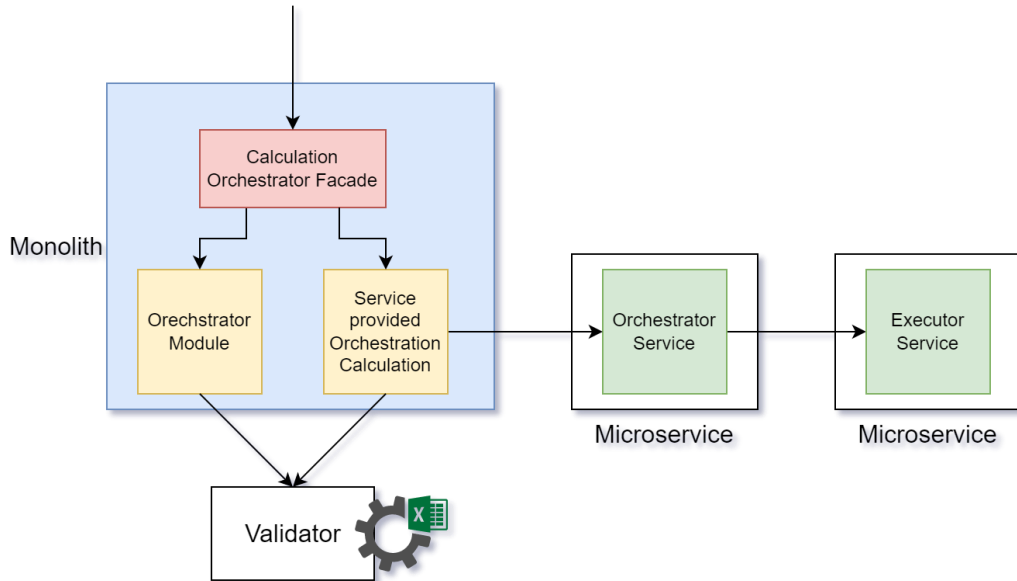
Step by step

1. Both new and old implementations are called during regular operation
2. Validation module compares the results of both implementations
3. Once satisfied with results switch to the new implementation



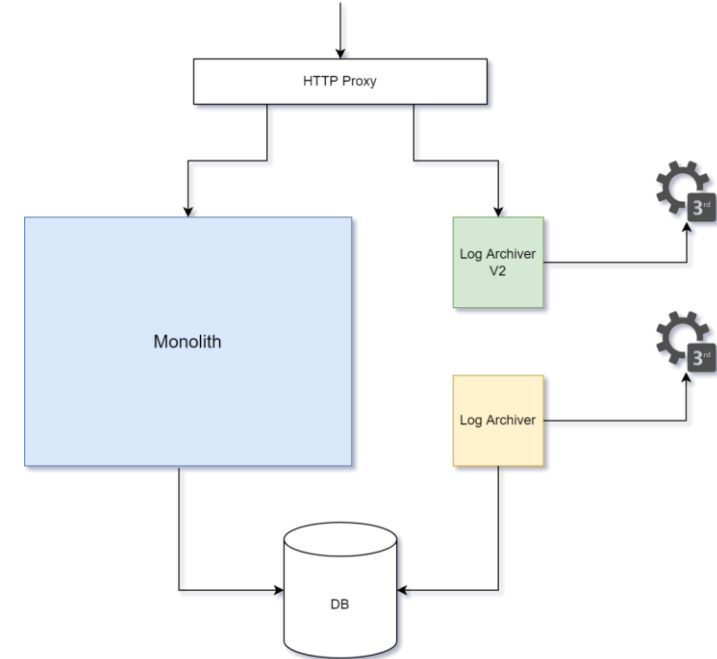
Examples: Combining the patterns

Strangler Fig Application + Branch by abstraction + Parallel run



- Different jobs types running on new and old implementation
- Compared results manually using excel
- As soon as results were validated all jobs were switched to new functionality

Strangler Fig Application + Parallel run



- Audit Log archiving is critical
- Logs are forwarded both to Monolith as well as Log Archiver V2
- Log Archiver is still running against the productive storage
- Log Archiver V2 is running against a mock so that behavior is evaluated

Splitting the Database

So, what about data?

Main things to consider

- Reminder: Microservices own their data
- **Each Microservice should have its own logical database**
- The database engine can be shared, but not the data
- **Avoid shared databases**



Patterns: The Shared Database

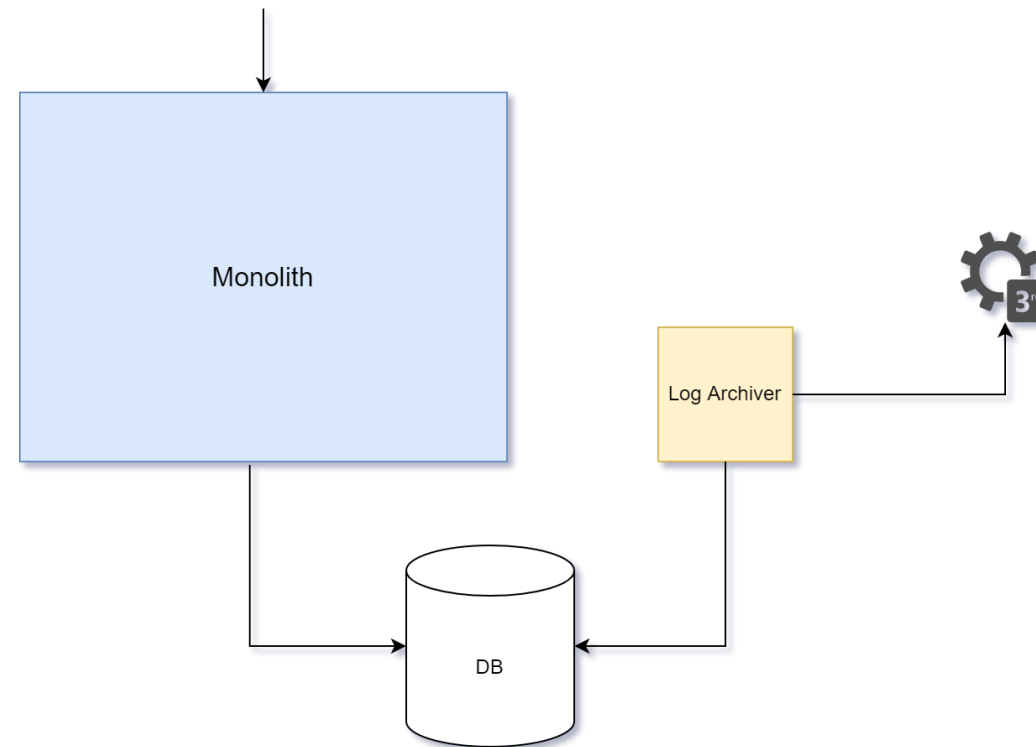
- Wait...What!?
- Can be a good starting point
- **Is not a long term solution**
- Has some very specific cases where it makes sense
- **However, avoid it if you can**



Patterns: The Shared Database

Usage scenarios

- Static read-only data(ex. Country codes)
- The database itself acts as a service
- **Example:** Log Archiver



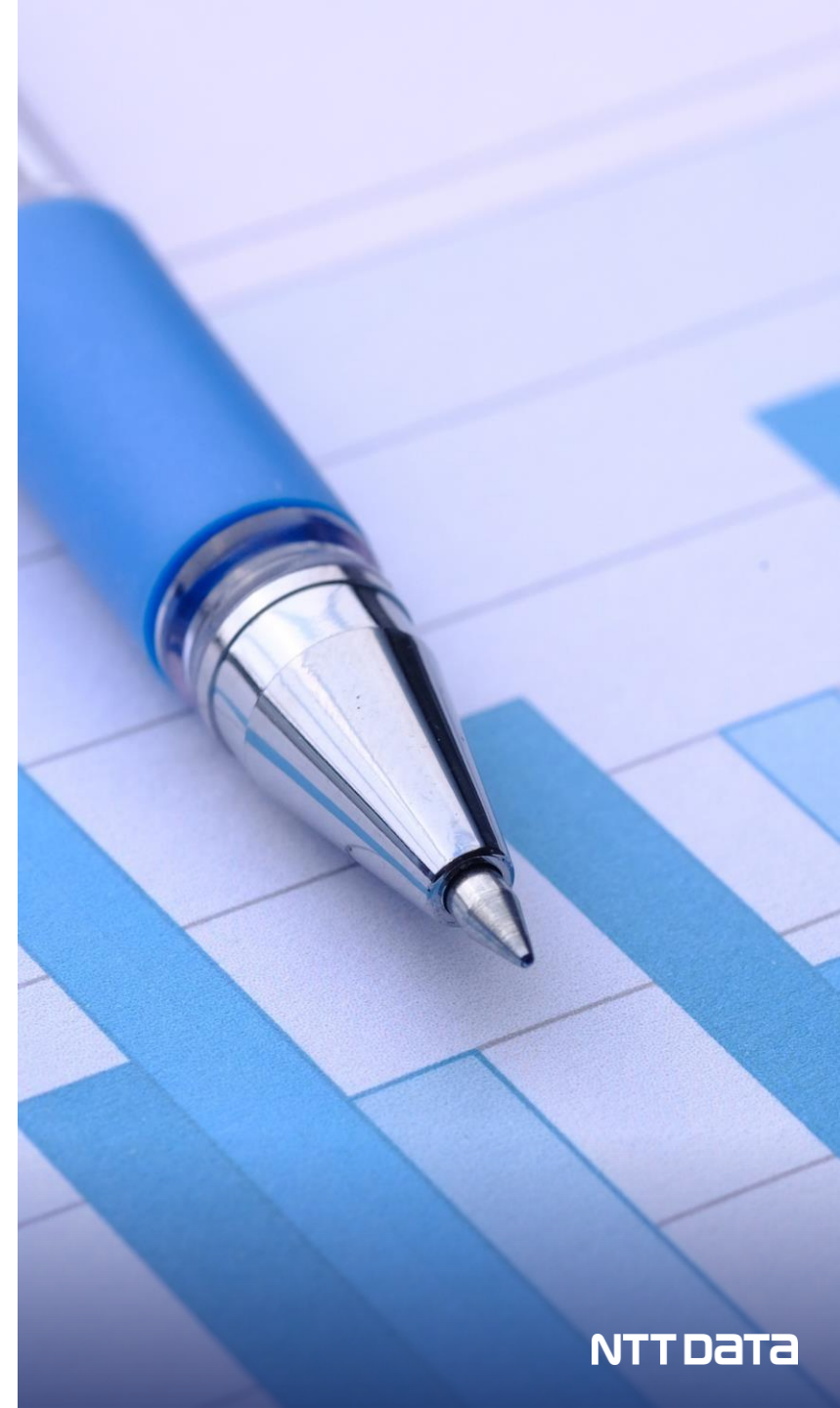
Patterns: DBaaS/CQRS/Reporting DB

- Specialization of the shared database pattern
- Database itself is a part of the distributed service architecture
- **Database is read-only**
- **Offers high flexibility in data querying**

Patterns: DBaaS/CQRS/Reporting DB

Usage scenarios

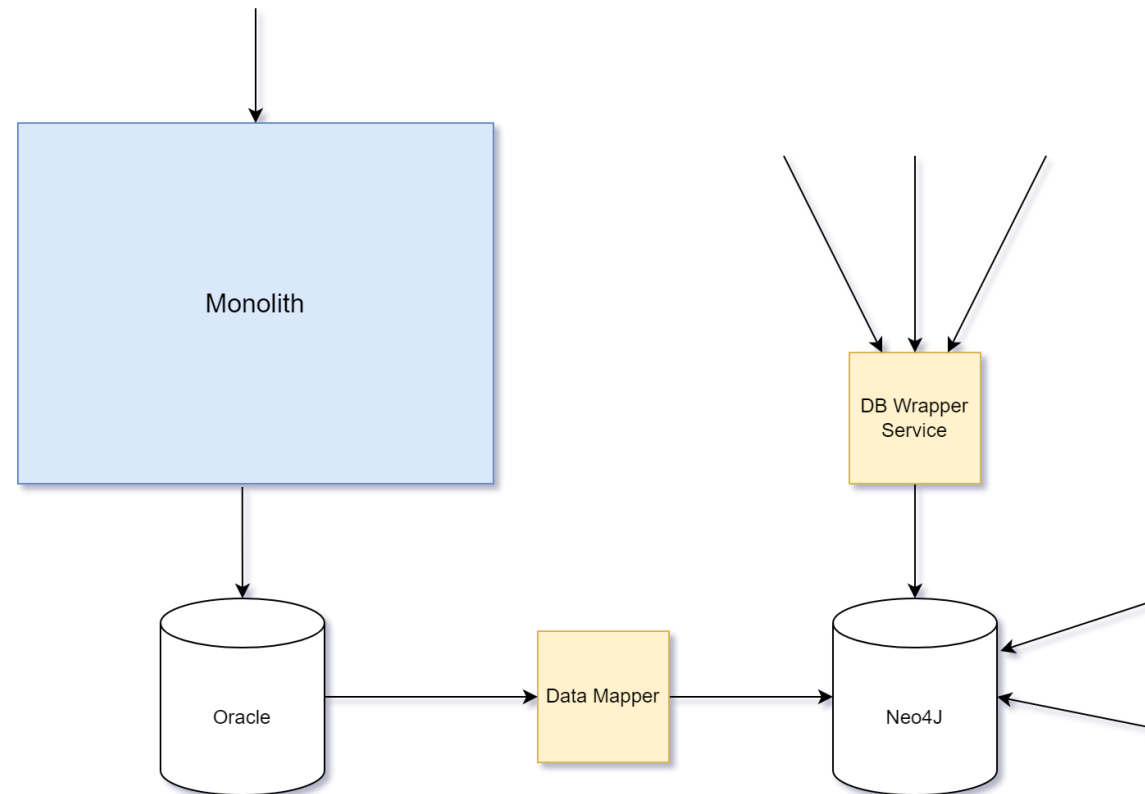
- Flexible report generation
- Data analysis and aggregation
- Data curation systems
- Can be used together with a database wrapping service



Patterns: DBaaS/CQRS/Reporting DB

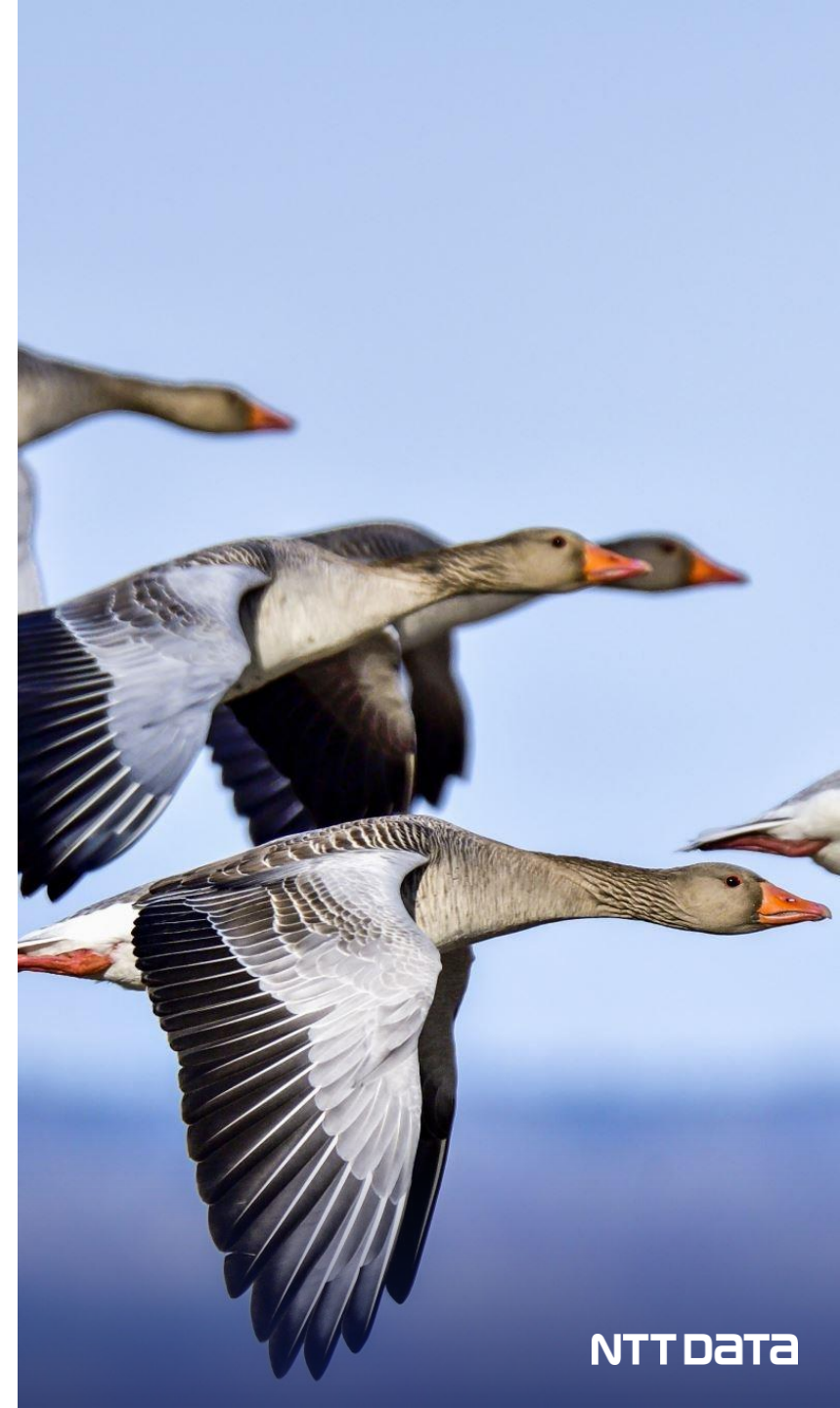
Example

- Graph database as a service



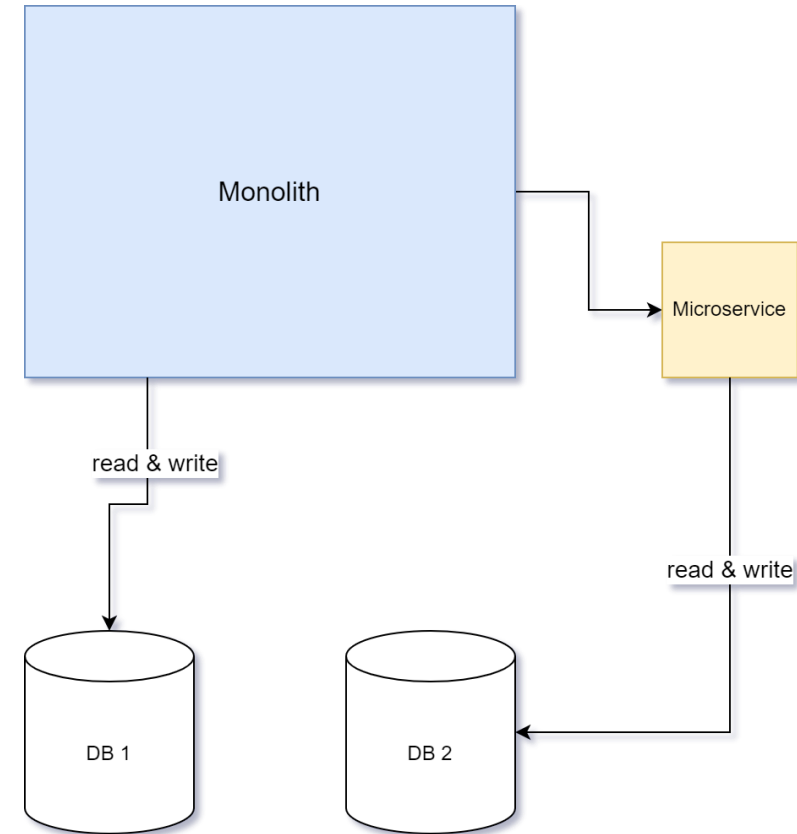
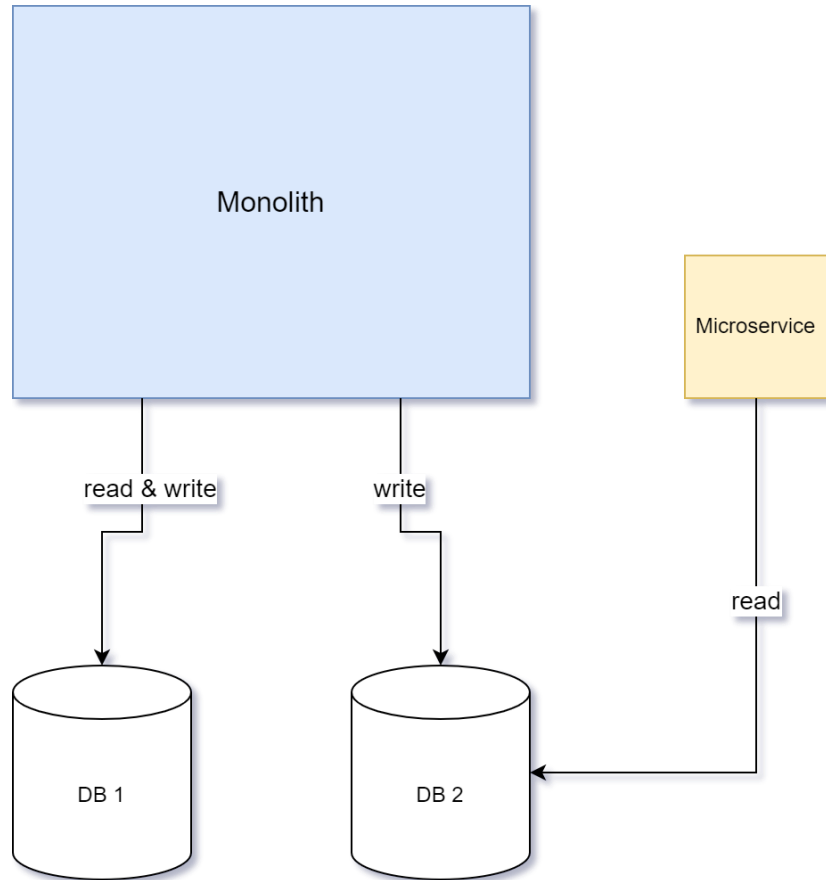
Patterns: Synchronize data in application

- **Application uses two databases to store data**
- Reads and writes to one DB, only writes to second DB
- Very helpful to enable fallback scenarios
- **Enables change of data ownership**



Patterns: Synchronize data in application

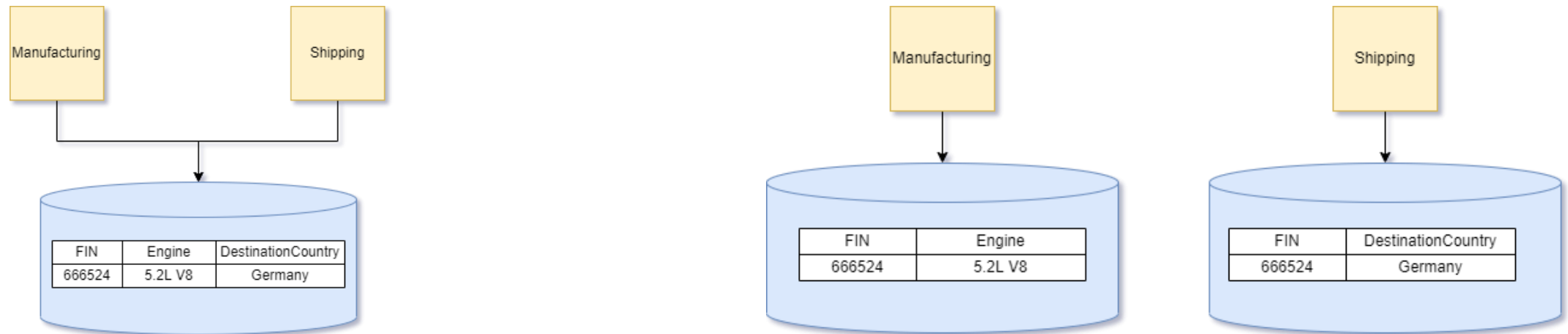
Example



Splitting the Database

Tables

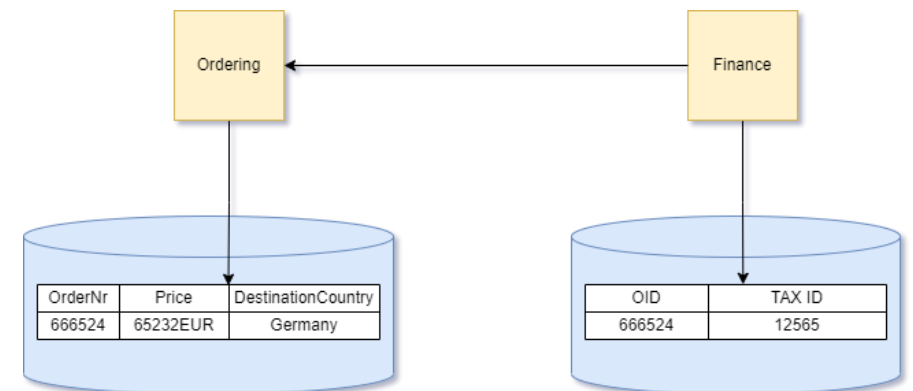
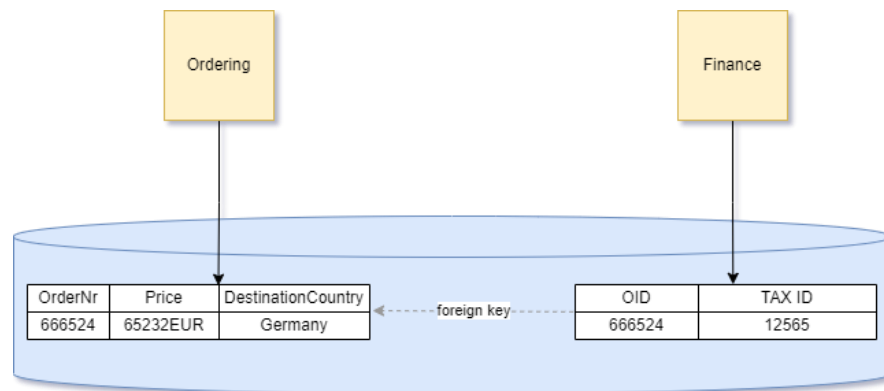
- Look for data belonging to different business areas found in the same table
- Data should be clearly identifiable via keys
- Avoid business keys, use internal keys if required



Splitting the Database

Handling Foreign Keys

- Database joins need to be done in code
- Performance decreases due to network overhead and lower performance compared to database engines
- Data consistency handling required
 - Handle deleted data
 - Handle inconsistent data



Splitting the Database

Transactions

- Transactions maintain the data integrity in complex systems
- We always want ACID transactions(atomicity, consistency, isolation, durability)
- **In distributed system atomicity is not possible**
- The **Saga** pattern can be used to handle transactions in distributed system
- **Handling rollbacks in distributed systems is much more closely related to business requirements**

Domain-Driven Design

- Collection of principles and patterns
- Helps create models of complex business domains
- Brings code together with business reality
- DDD Bible:
Eric Evans, "Domain-Driven Design: Tackling Complexity in the Heart of Software"



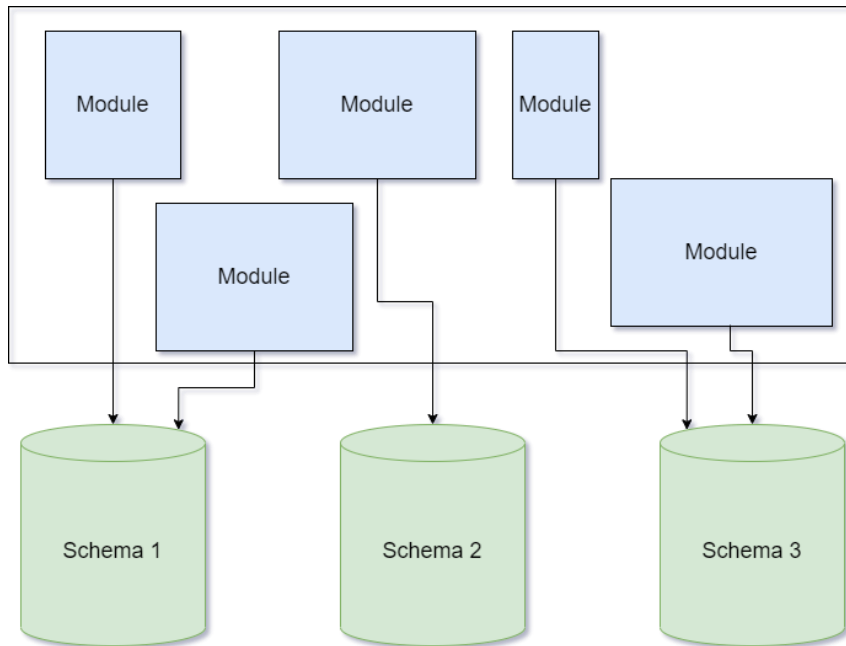
Data or Code?

What do I split first

Data or Code?

Data

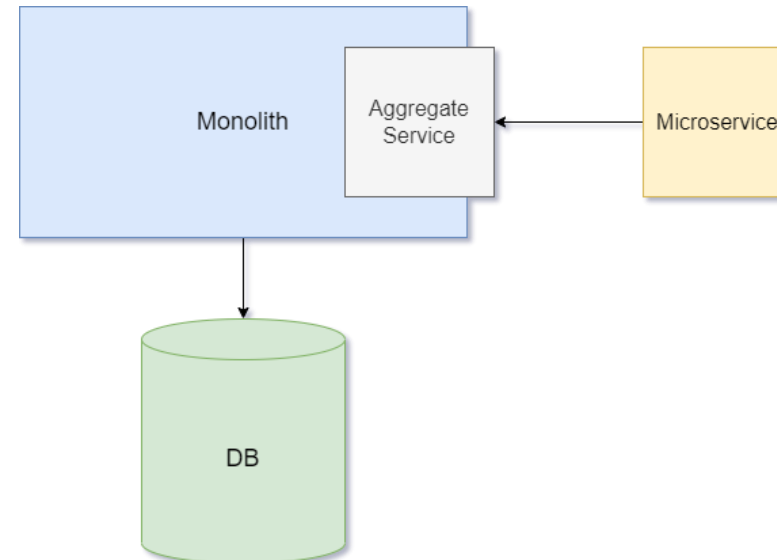
- Have a very well-known domain
- Easiest way to start is using the **Multi-Schema** pattern
- Can be combined well with the modular monolith



Data or Code?

Code

- When domain is unclear, or no confidence in domain knowledge
- Easier starting point to extract just functionality
- Works well with **Aggregate exposing monolith** pattern

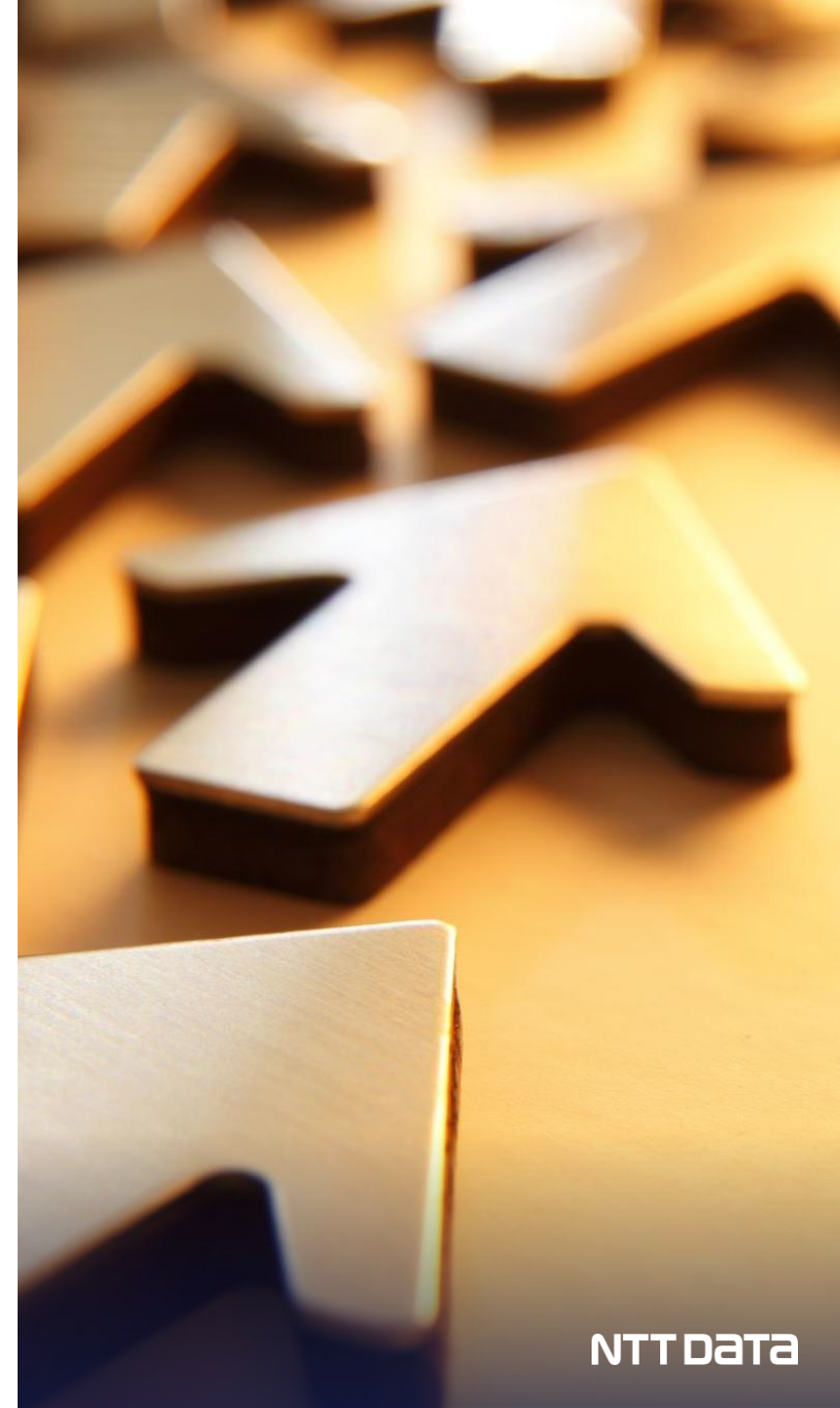


Dealing with new features

Yes, stakeholders will still want them

Dealing with new features

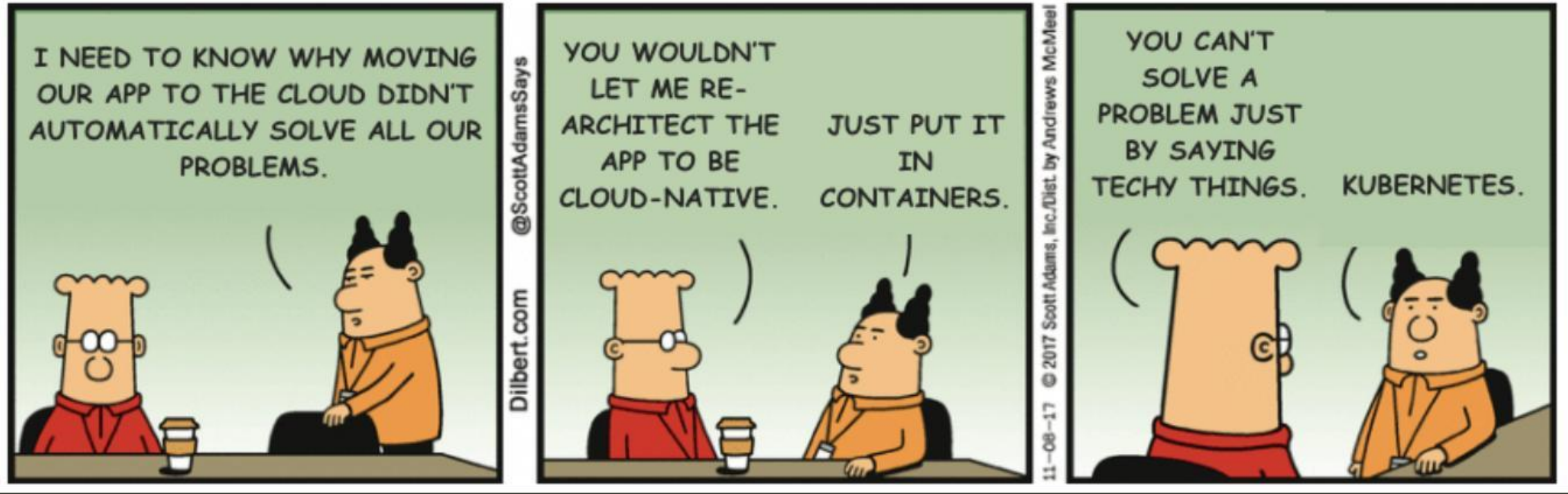
- Evaluate how strongly a new feature is coupled to existing features
- **Find synergies to logic that could be extracted as a microservice**
- **Always look for simple and ideally harmless changes**
- Avoid long lived branches, they lead to big bangs
- Apply the patterns:
 - **Strangler Fig**
 - **Branch by Abstraction**
 - **Parallel Run**
 - **Data Synchronization**
 - **Aggregate Exposing Monolith**



People, Tools and Processes

It's not only the architecture that has to evolve

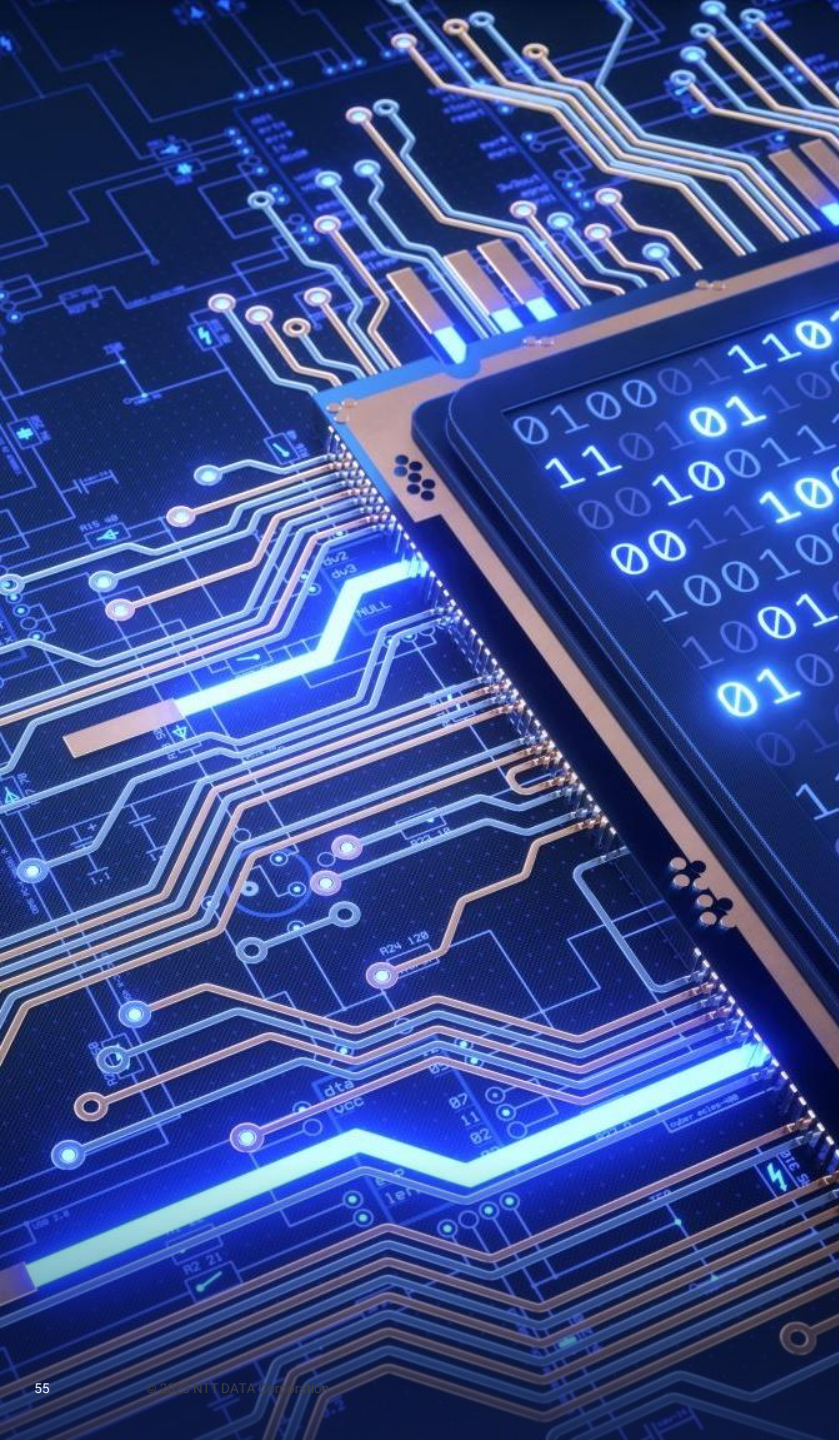
People, Tools and Processes



People, Tools and Processes

Requirements

- Keep stories small and incremental
- Describe the need, the development team has the job to decide how to fulfill the need
- **Stakeholders shouldn't decide where a feature will be implemented, you will end up with a Big Ball of Mud**
- People(including developers) have a hard time understanding the complexity of microservice architectures



People, Tools and Processes

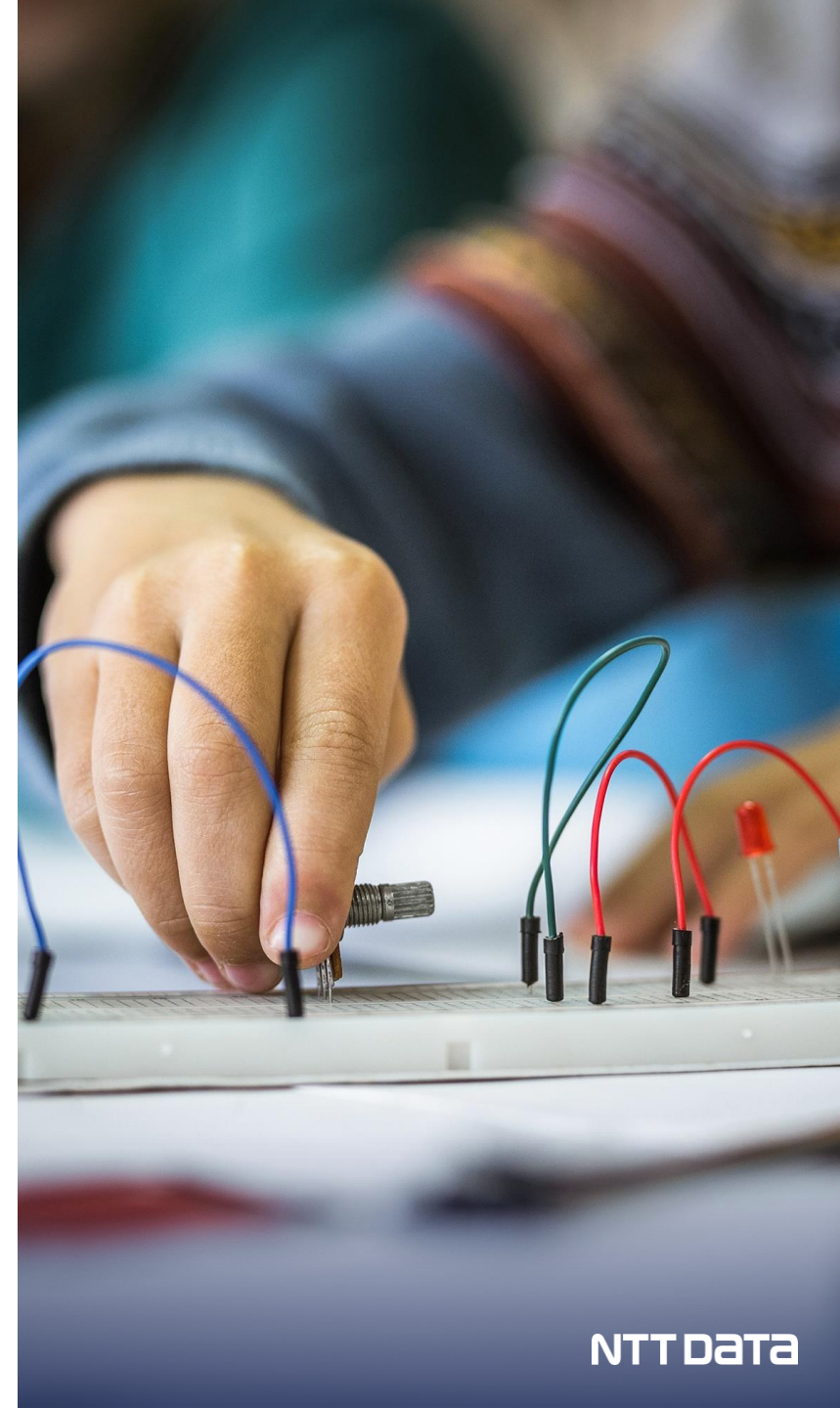
Development

- Ownership of microservices becomes more complex the more microservices you have
- **Local development complexity, more microservices means more overhead to test**
- **Communicating goals clearly and taking transparent decisions is important**
- Interface handling in microservices is complex
- Scheduled task handling in replicated systems is more complex(SchedLock)

People, Tools and Processes

Testing

- **Test as early as possible**
- Test automation will give you higher confidence when rolling out services
- **Tester time should be focused on acquiring business knowledge and designing tests instead of executing them**
- Developers need to be closely involved in the testing process
- Adopt testing frameworks:
 - **Newman**
 - **REST Assured**
 - **Locust**





People, Tools and Processes

Operations

- DevOps is not just about tools but also a cultural philosophy
- Developers need to get used to the concept of “you build it, you run it”
- Automate your release process to reduce your “time to market”
- Large number of tools available:
 - GitHub Actions/GitLab CI/CD
 - ArgoCD
 - Grafana(incl. Loki and Tempo)
 - Dynatrace
 - Jaeger
 - Istio

Summary

Summary

TH

how do i migrate a monolith to microservices



Migrating a monolithic application to a microservices architecture can be a complex process that requires careful planning and execution. Here's a high-level approach to help you get started:



1. **Analyze the monolith:** Understand the existing monolithic application thoroughly. Identify its different components, dependencies, and business domains. This analysis will help you determine which parts can be decoupled and transformed into microservices.
2. **Define microservices boundaries:** Identify the logical boundaries for microservices based on business domains, functionality, or modules within the monolith. This involves breaking down the monolith into smaller, more manageable services.
3. **Identify service interactions:** Analyze how different services within the monolith interact with each other. Determine the communication protocols, data formats, and dependencies involved. This analysis will help you design the communication patterns for your microservices.
4. **Design microservices architecture:** Define the architecture for your microservices. Choose appropriate technologies, frameworks, and tools. Decide on data storage solutions, communication protocols (e.g., REST, message queues), and deployment strategies (e.g., containerization, orchestration).
5. **Prioritize microservices:** Decide on the order in which you'll migrate different parts of the monolith to microservices. Consider factors like business impact, dependencies, and technical feasibility. You might start with low-risk, isolated components to gain early wins and build momentum.

Summary

6. **Refactor and decouple:** Begin refactoring the monolith into microservices incrementally. Break down the identified components into separate services, ensuring loose coupling and high cohesion. Modify the code to adapt to the microservices architecture, adhering to the principles of single responsibility and separation of concerns.
7. **Establish APIs and contracts:** Define clear and well-documented APIs for each microservice. This allows different services to communicate with each other effectively. Consider using API gateways or service meshes to manage the interaction between services.
8. **Implement inter-service communication:** Set up communication mechanisms between microservices. This can include synchronous HTTP/REST calls, asynchronous messaging using queues or publish-subscribe systems, or event-driven architectures. Use appropriate protocols like JSON or Protobuf for data interchange.
9. **Containerize and orchestrate:** Containerize your microservices using technologies like Docker. Utilize container orchestration platforms such as Kubernetes to manage deployment, scaling, and high availability. This ensures your microservices can be easily deployed and managed in a distributed environment.
10. **Deploy and test:** Deploy your microservices to a test environment and thoroughly test their functionality, integration, and performance. Use automated testing frameworks and tools to streamline the testing process and ensure the reliability of your system.

Summary

11. **Gradual migration:** Once you have tested and validated the microservices, start gradually migrating users and functionality from the monolith to the microservices. Monitor the performance and stability of the system during the migration process, ensuring a seamless transition for end-users.
12. **Iterate and optimize:** Continuous improvement is essential in a microservices architecture. Monitor and gather feedback from users and system metrics. Refactor and optimize your microservices based on real-world usage patterns and performance insights.

Remember that migrating to microservices is not a one-time task but an ongoing journey. It requires careful planning, coordination, and a focus on the benefits that microservices can bring to your application's scalability, maintainability, and agility.

Questions

Contact



Alin Vulparu

alin.vulparu@nttdata.com

Tel: +49 151 1634 1269



NTT DATA
Trusted Global Innovator