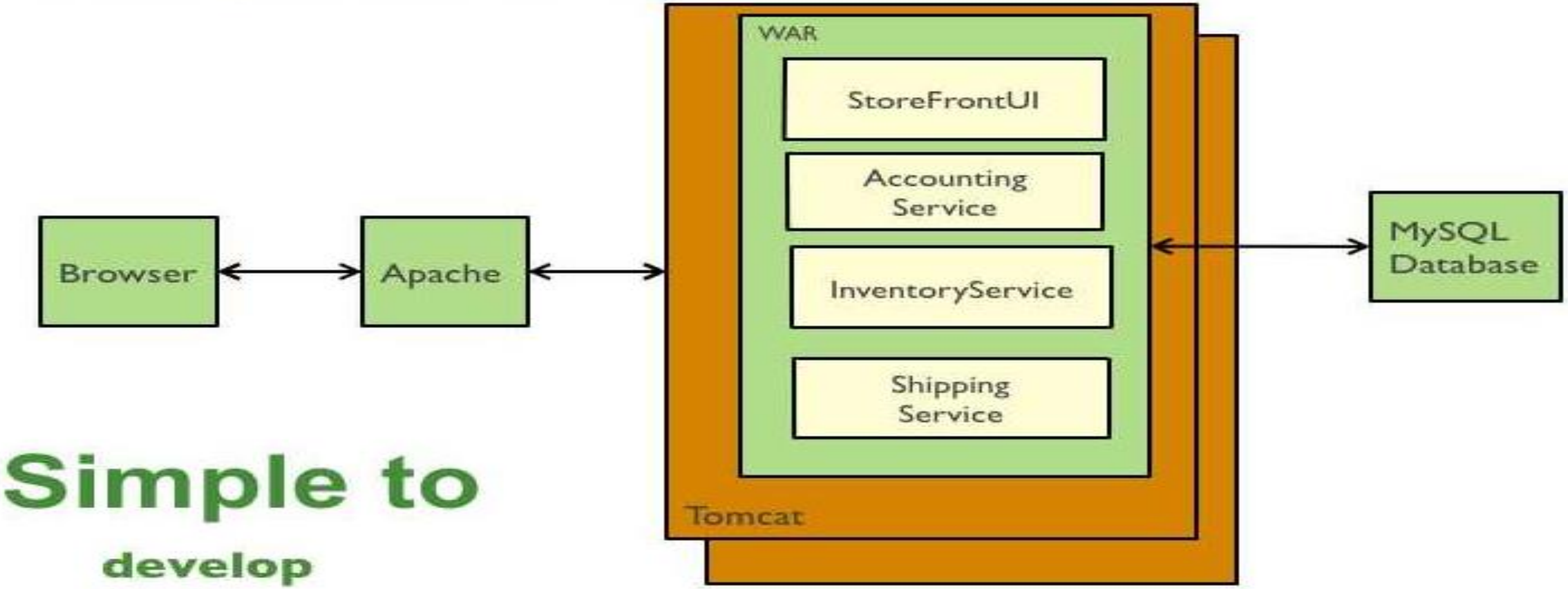
MICROSERVICES

MONOLITHIC ARCHITECTURE

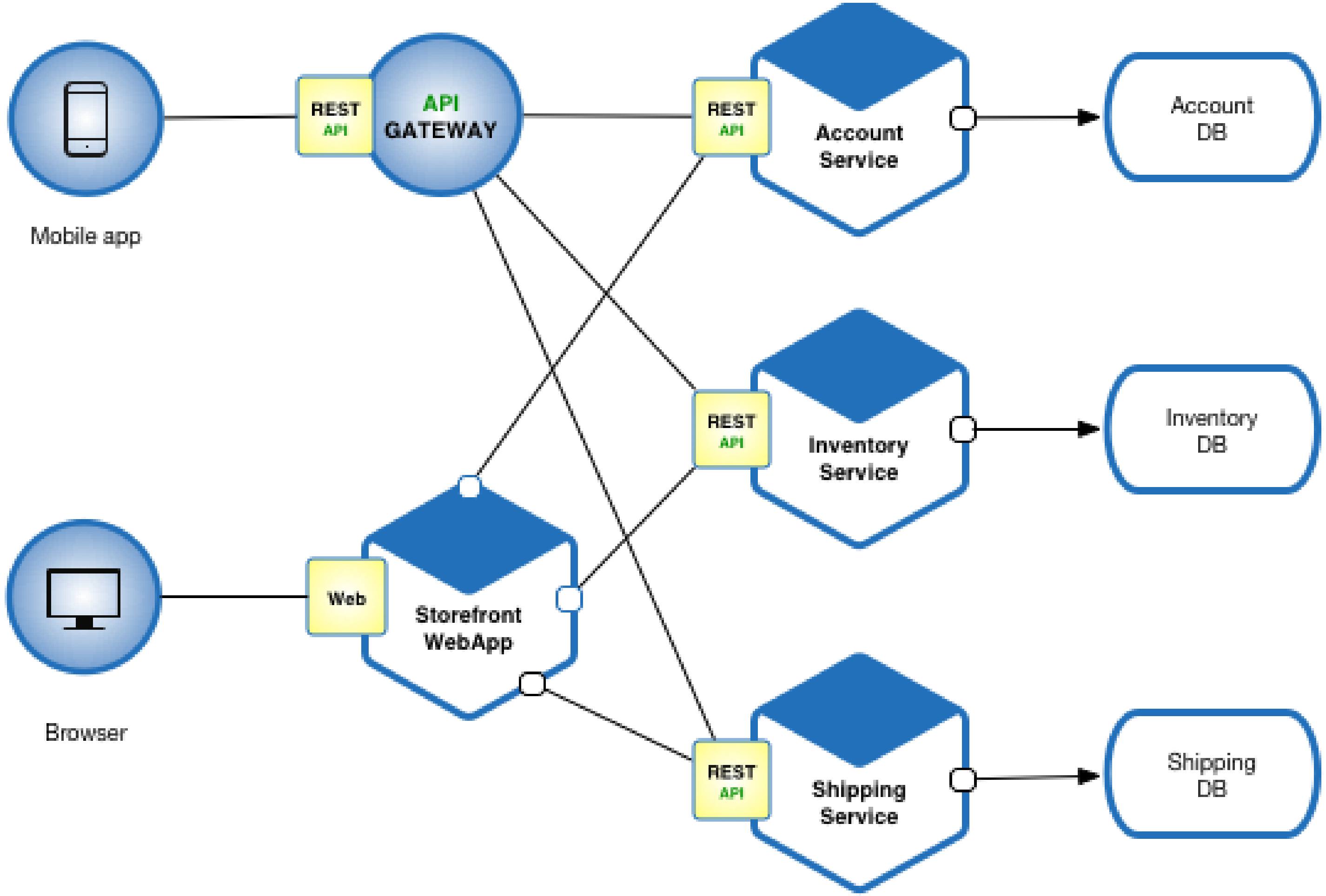
Traditional web application architecture



Simple to

- develop
- test
- deploy
- scale

MICROSERVICE ARCHITECTURE



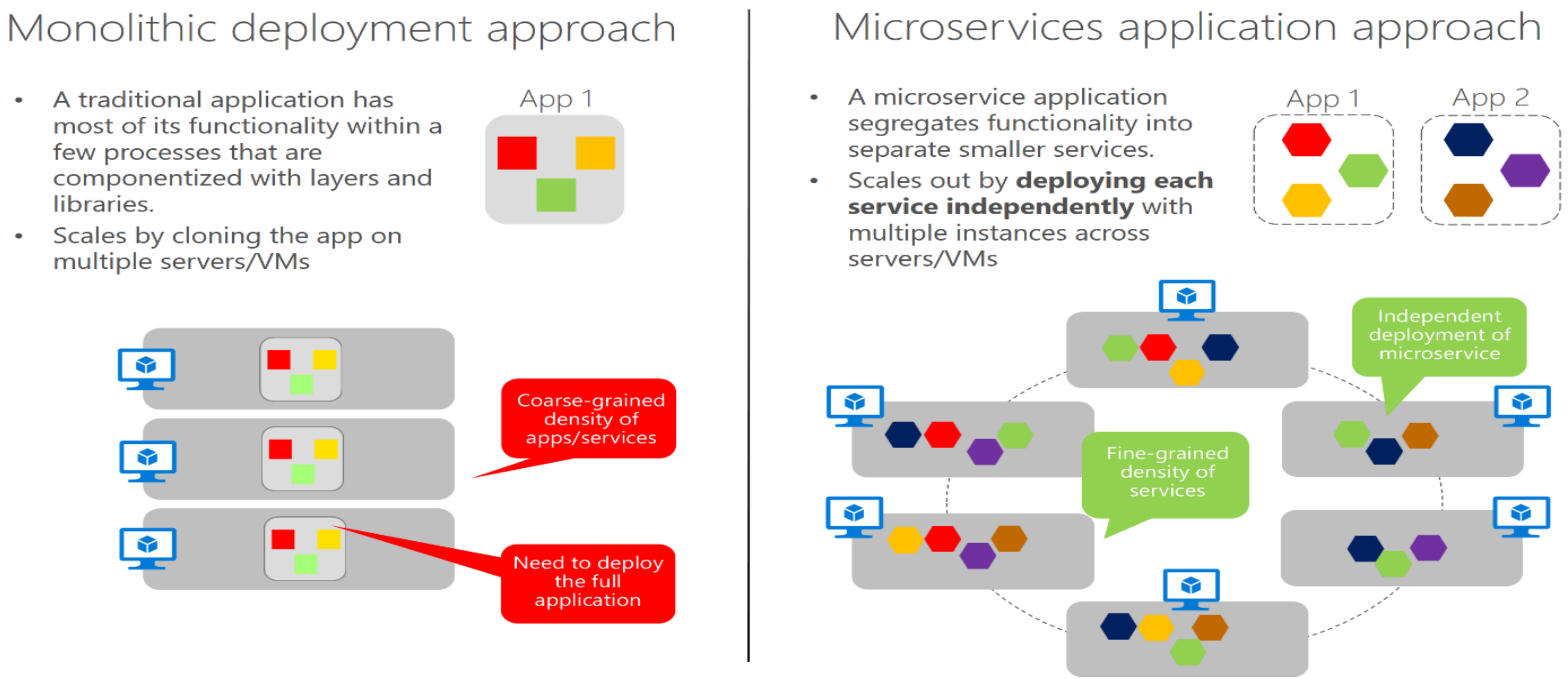
MICROSERVICE

Microservices - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

COMPARISON



Monolithic deployment approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs

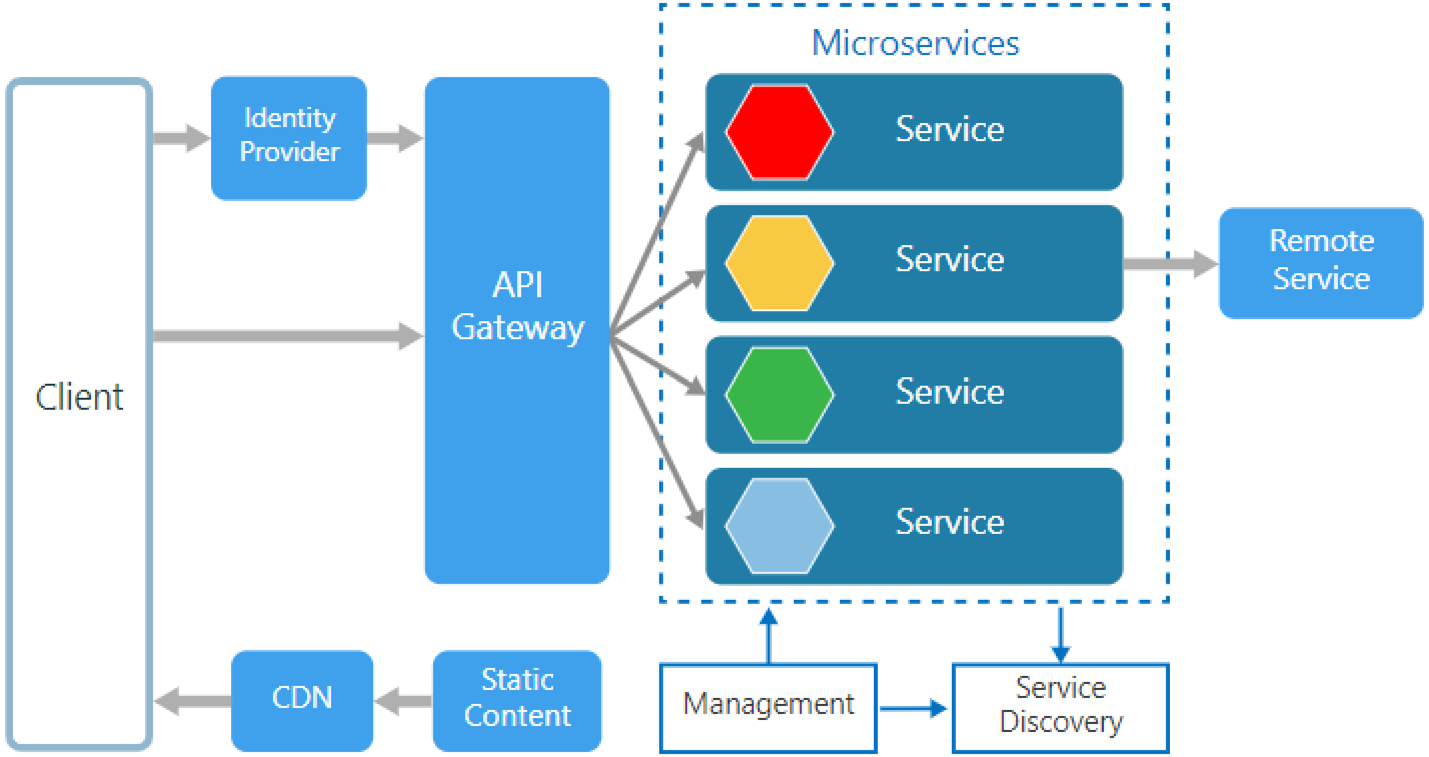
Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs

App 1
Coarse-grained density of apps/services
Need to deploy the full application
App 1
App 2
Independent deployment of microservice
Fine-grained density of services

MICROSERVICES ARCHITECTURE STYLE

A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and should implement a single business capability.



OTHER COMPONENTS OF MICROSERVICES ARCHITECTURE

Management. The management component is responsible for placing services on nodes, identifying failures, rebalancing services across nodes, and so forth.

Service Discovery. Maintains a list of services and which nodes they are located on. Enables service lookup to find the endpoint for a service.

API Gateway. The API gateway is the entry point for clients. Clients don't call services directly. Instead, they call the API gateway, which forwards the call to the appropriate services on the back end. The API gateway might aggregate the responses from several services and return the aggregated response.

CHARACTERISTICS OF A MICROSERVICE

In a microservices architecture, services are small, independent, and loosely coupled.

Each service is a separate codebase, which can be managed by a small development team.

Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.

Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.

Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.

Services don't need to share the same technology stack, libraries, or frameworks.

BENEFITS

Independent deployments. You can update a service without redeploying the entire application, and roll back or roll forward an update if something goes wrong. Bug fixes and feature releases are more manageable and less risky.

Independent development. A single development team can build, test, and deploy a service. The result is continuous innovation and a faster release cadence.

Small, focused teams. Teams can focus on one service. The smaller scope of each service makes the code base easier to understand, and it's easier for new team members to ramp up.

Fault isolation. If a service goes down, it won't take out the entire application. However, that doesn't mean you get resiliency for free.

Mixed technology stacks. Teams can pick the technology that best fits their service.

Granular scaling. Services can be scaled independently. At the same time, the higher density of services per VM means that VM resources are fully utilized. Using placement constraints, a services can be matched to a VM profile (high CPU, high memory, and so on).

CHALLENGES

Complexity. A microservices application has more moving parts than the equivalent monolithic application. Each service is simpler, but the entire system as a whole is more complex.

Development and test. Developing against service dependencies requires a different approach. Existing tools are not necessarily designed to work with service dependencies. Refactoring across service boundaries can be difficult. It is also challenging to test service dependencies, especially when the application is evolving quickly.

Lack of governance. The decentralized approach to building microservices has advantages, but it can also lead to problems. You may end up with so many different languages and frameworks that the application becomes hard to maintain.

Network congestion and latency. The use of many small, granular services can result in more interservice communication.

Data integrity. With each microservice responsible for its own data persistence. As a result, data consistency can be a challenge. Embrace eventual consistency where possible.

ISSUES

When to use the microservice architecture?

How to decompose the application into services?

- Decompose by business capability and define services corresponding to business capabilities.
- Decompose by domain-driven design subdomain.
- Decompose by verb or use case and define services that are responsible for particular actions. e.g. a Shipping Service that's responsible for shipping complete orders.
- Decompose by by nouns or resources by defining a service that is responsible for all operations on entities/resources of a given type. e.g. an Account Service that is responsible for managing user accounts.

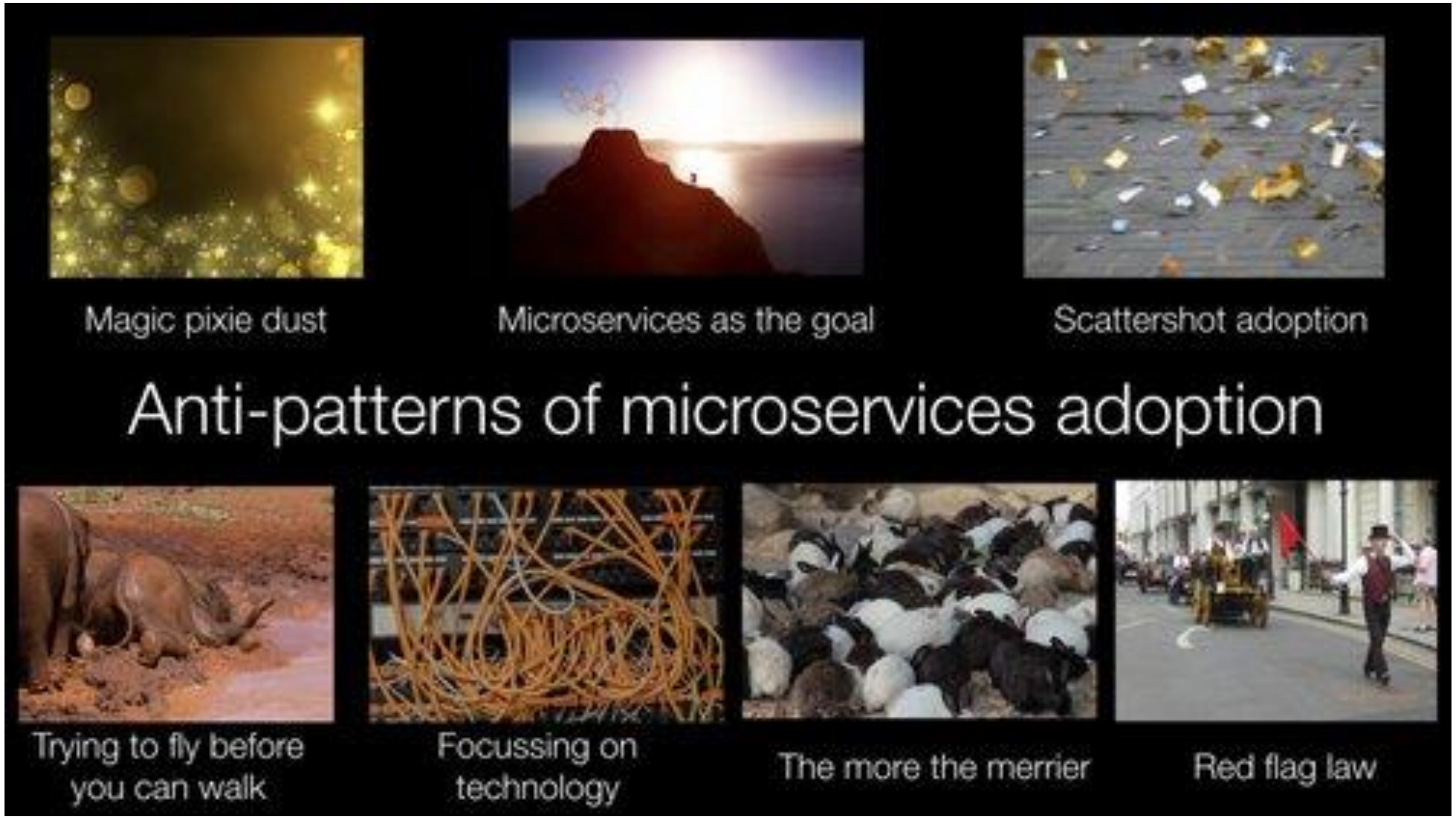
How to maintain data consistency?

Maintaining data consistency between services is a challenge because 2 phase-commit/distributed transactions is not an option for many applications.

How to implement queries?

implementing queries that need to retrieve data owned by multiple services.

MICROSERVICES ADOPTION ANTIPATTERNS



Magic pixie dust

Microservices as the goal

Scattershot adoption

Anti-patterns of microservices adoption

Trying to fly before you can walk

Focussing on technology

The more the merrier

Red flag law

REFERENCES

1. https://microservices.io/patterns/monolithic.html
2. https://microservices.io/patterns/microservices.html
3. https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices
4. https://microservices.io/patterns/decomposition/decompose-by-business-capability.html
5. https://microservices.io/patterns/decomposition/decompose-by-subdomain.html
6. https://microservices.io/microservices/antipatterns/-/the/series/2019/06/18/microservices-adoption-antipatterns.html