Microservice Patterns and Best Practices

PeakIT 004 23.10.2021

DIAMOND

SIEMENS · nagarro · AtoS · Trimble. · Raiffeisen BANK Banking așa cum trebuie

SILVER

accenture · endava · cegeka

PARTNERS

coresi · <ScriuCod/> · Codette · CODECAMP_

MEDIA PARTNERS

CAPITAL · BizBrasov Stirile care conteaza in Brasov · start#up · ROCK FM It Rocks!

AGENDA

AGENDA

05

DevOps

06

Demo

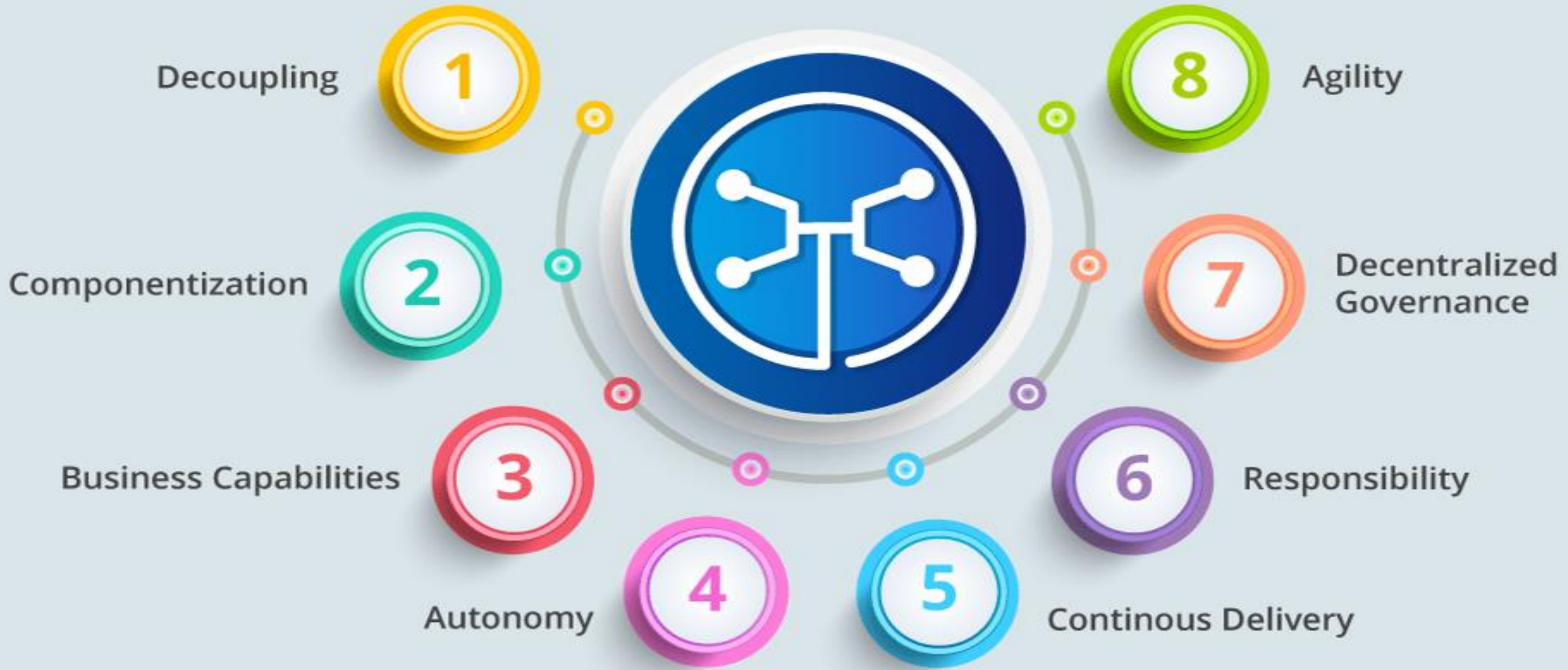
01

Definition and short intro

What is Microservice?

- An approach to application development.
- A suite of modular components or services.
- Each microservice is responsible for a single feature.
- Communicate over well-defined APIs.
- Running in its own process.
- Built around business capabilities.

What is Microservice?



MSA features

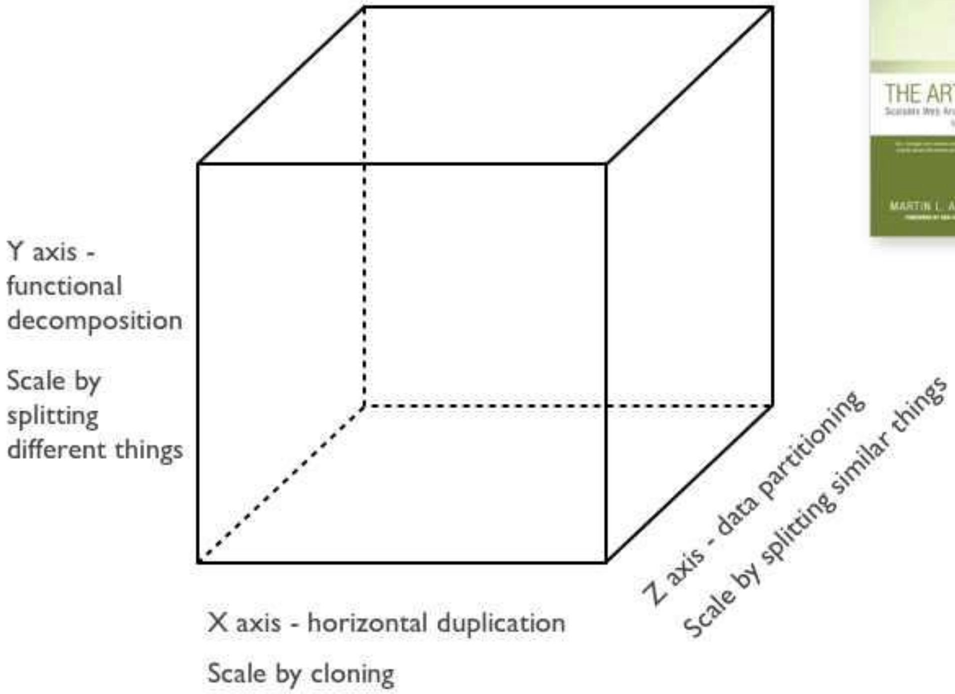
- **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
- **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
- **Business Capabilities** – Microservices are very simple and focus on a single capability
- **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
- **Continuous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
- **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
- **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
- **Agility** – Microservices support agile development. Any new feature can be quickly developed and discarded again

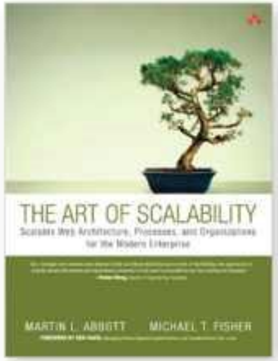
02

Key elements/tenents related to MSA

Scale cube

3 dimensions to scaling

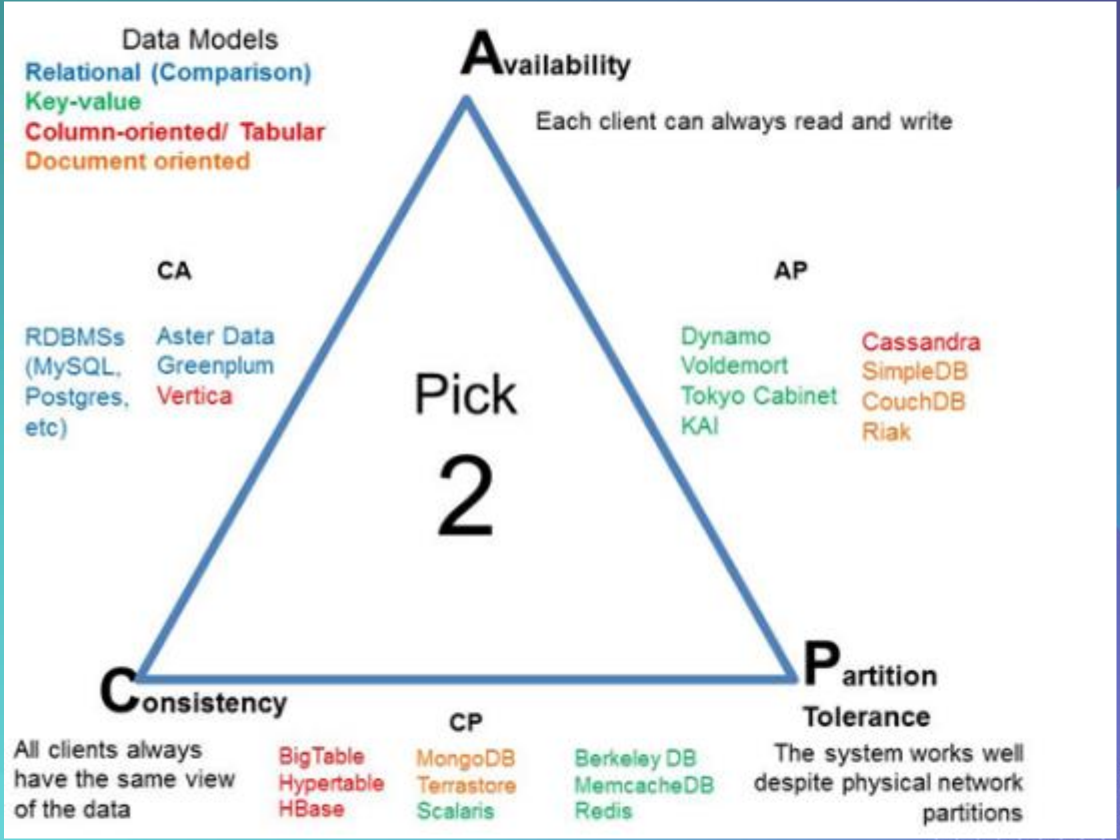

Y axis - functional decomposition
Scale by splitting different things
X axis - horizontal duplication
Scale by cloning
Z axis - data partitioning
Scale by splitting similar things


THE ART OF SCALABILITY
Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise
MARTIN L. ABBOTT
MICHAEL T. FISHER

Scale cube

- **- X-** axis scaling consists of running multiple copies of an application behind a load balancer.
- **- Y-** axis scaling splits the application into multiple, different services. Each service is responsible for one or more closely related functions.
- **- Z-** axis idea can be better understood by using the sharding concept.

CAP Theorem


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick 2
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions

CAP Theoreme

- **Consistency** means that all clients see the same data at the same time, no matter the path of their request. This is critical for applications that do frequent updates.
- **Availability** means that that any client making a request for data gets a response, even if one or more nodes are down.
- **Partition** tolerance means that the application will operate even during a network failure that results in lost or delayed messages between services. This comes into play for applications that integrate with a large number of distributed, independent components.

CAP Theoreme - databases

- CP database: A CP database delivers consistency and partition tolerance at the expense of availability. When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved.
- AP database: An AP database delivers availability and partition tolerance at the expense of consistency. When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, the AP databases typically resync the nodes to repair all inconsistencies in the system.)
- CA database: A CA database delivers consistency and availability across all nodes. It can’t do this if there is a partition between any two nodes in the system, however, and therefore can’t deliver fault tolerance.

12-Factor App compliance

- Codebase (One codebase tracked in revision control, many deploys)
- Dependencies (Explicitly declare and isolate the dependencies)
- Config (Store configurations in an environment)
- Backing Services (treat backing resources as attached resources)
- Build, release, and Run (Strictly separate build and run stages)
- Processes (execute the app as one or more stateless processes)
- Port Binding (Export services via port binding)
- Concurrency (Scale out via the process model)
- Disposability (maximize the robustness with fast startup and graceful shutdown)
- Dev/prod parity (Keep development, staging, and production as similar as possible)
- Logs (Treat logs as event streams)
- Admin processes (Run admin/management tasks as one-off processes)

03

NFR related to MSA

We focus on the most important non functional requirements and why there are needed

Reliability

There are two important aspects to application reliability.

1. Whatever data that you present to the system user, how accurate is that data? Are there any bugs in the source of that data?
2. How reliable is the data present in your database? This is crucial, because such data could be provided to other systems, or be used to generate important analytics.

How to ...?

Write Great Code

Write Great Tests

Use Continuous Integration

Automate The Testing Process

Ensure Proper Database Design

Availability

- Availability is a measure of how frequently your system provides the desired functionality, to your users.

How to …?

Modularity Improves Availability

Redundancy Improves Availability

Monitoring Improves Availability

Performance

- When we type in a web page URL in a browser, performance from a users perspective is the time it takes for the page to load.
- Performance is essentially a measure of how much time a system takes to respond to a user request.

How to ...?

- Improve The Code
- Fine-Tune Your Databases
- Cache Your Data
- Build Redundancy
- Improve System Capacity
- Ensure Modular Design

Scalability

Let’s say we have an application A, that supports N number of users with its current infrastructure. If we increase the infrastructure by a factor of 10, can we hope to support 10 times the current users?

How we can support a multifold increase in number of users with a linear increase in infrastructure?

How to ...?

- Have Modular Applications
- Aim for statelessness in your app.
- Make sure to cache everything

04

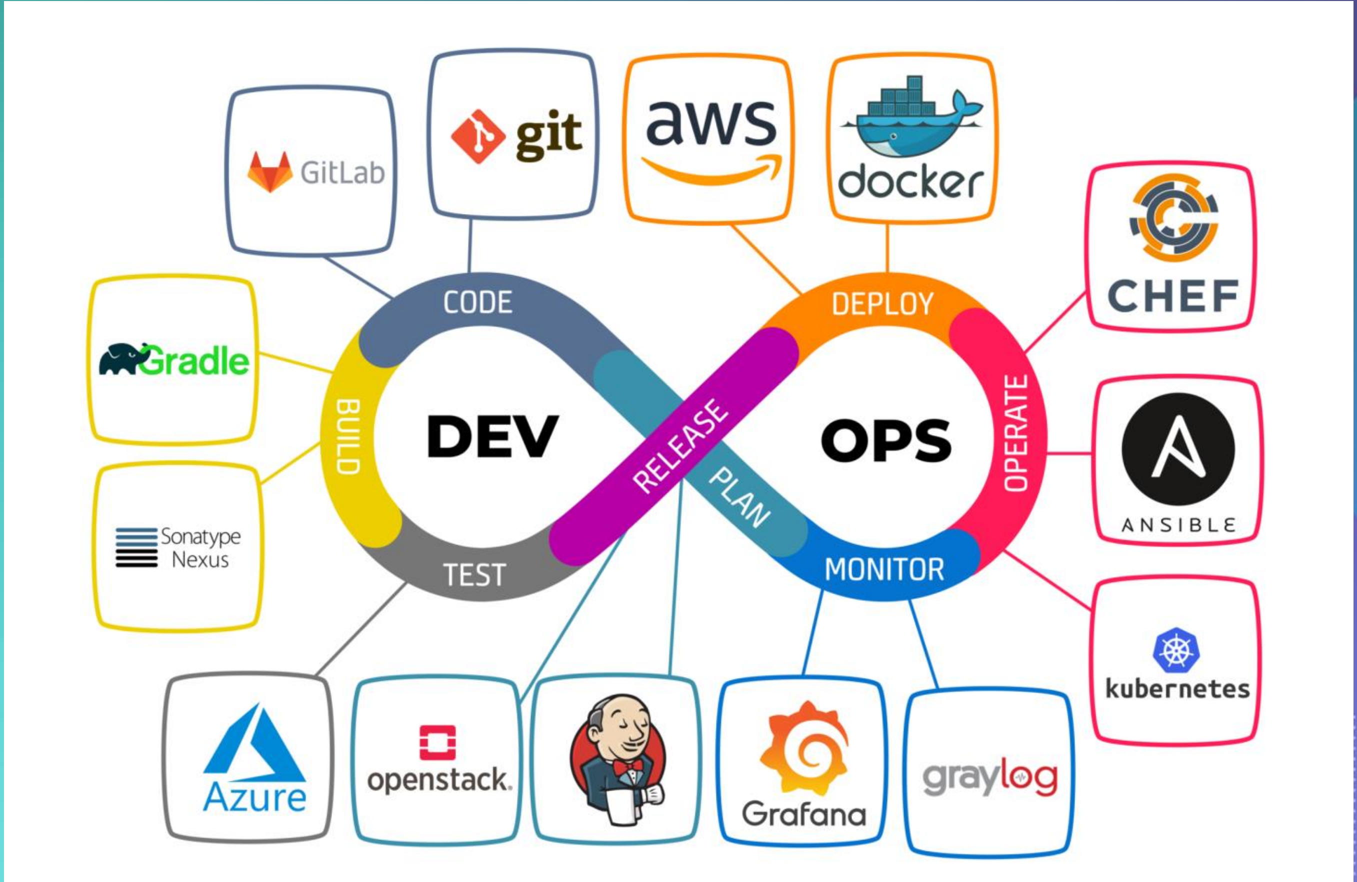
Few examples

- https://developers.soundcloud.com/blog/service-architecture-2
- https://blog.karmawifi.com/how-we-build-microservices-at-karma-71497a89bfb4

05

DevOps

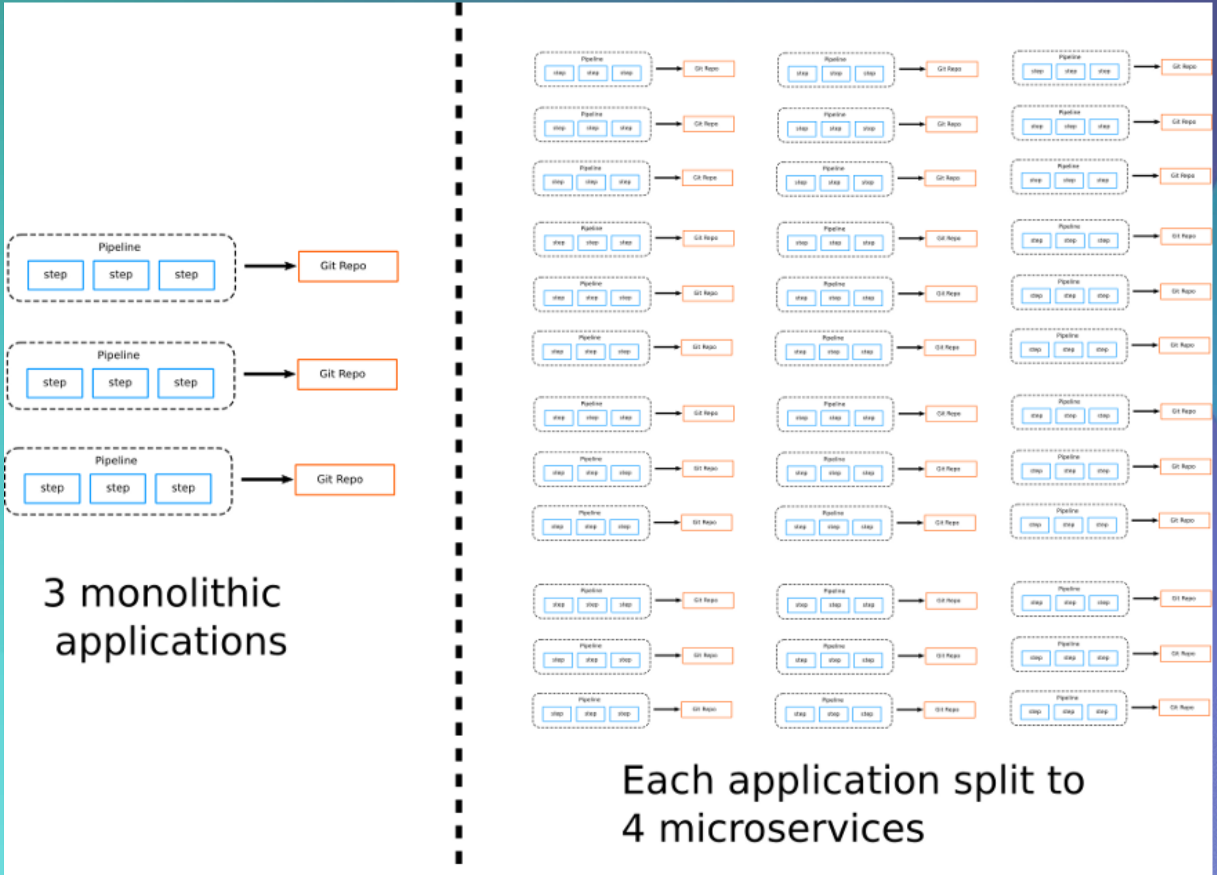
DevOps


GitLab
git
aws
docker
CHEF
CODE
DEPLOY
Gradle
BUILD
DEV
RELEASE
PLAN
OPS
OPERATE
ANSIBLE
Sonatype Nexus
TEST
MONITOR
kubernetes
Azure
openstack
Grafana
graylog

What is ...?

- A DevOps culture means that rather than handing off work from the development team to operations, each team takes responsibility for the whole lifecycle of its service. Breaking down the silos between development and operations means that developers get a better understanding of the infrastructure and process involved in releasing their service, while operations better understand the functionality of the whole system. By characterizing the release process as an engineering problem, infrastructure is treated as code and the process is optimized and automated. In adopting a DevOps culture, it's important to avoid falling into the trap of creating a dedicated DevOps team to take care of managing deployment infrastructure. Doing that just creates another silo, and your organization will miss out on the full benefit.

Why...?



3 monolithic applications

Each application split to 4 microservices

How to ...?

Continuous integration

Code changes are frequently merged into the main branch. Automated build and test processes ensure that code in the main branch is always production-quality.

Continuous delivery

Any code changes that pass the CI process are automatically published to a production-like environment. Deployment into the live production environment may require manual approval, but is otherwise automated. The goal is that your code should always be ready to deploy into production.

Continuous deployment

Code changes that pass the previous two steps are automatically deployed into production.

06

Demo

Feedback

http://bit.ly/peakit004-feedback

Completați aici, în sală

Durează 2-3 minute

Feedback anonim - pentru formator si AgileHub

THANK YOU

for climbing #peakit004 with us.

You are awesome!

AgileHub

THANKS

Does anyone have any questions?

addyouremail@freepik.com
+91 620 421 838
yourcompany.com