
aws
SUMMIT
MILANO
22 GIUGNO 2023

DAT201

Come Tagetik ha disegnato una soluzione multitenant per i propri clienti sfruttando i servizi gestiti AWS per i dati

Jacopo Roma
Cloud & DevOps Manager
Wolters Kluwer

Fabio Testa
Principal Application & Product Architect
Wolters Kluwer

Mattia Berlusconi
Solutions Architect, Data Specialist
Amazon Web Services

aws SUMMIT MILANO

Scarica la App «AWS Events» e accedi a tutte le informazioni dell'evento!

GET IT ON Google Play

Download on the App Store

Agenda

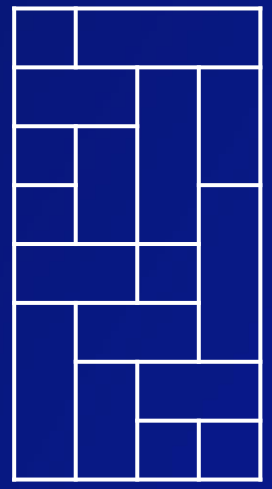
- Database modernization with AWS
- CCH® Tagetik | Wolters Kluwer
- Data requirements and multitenancy
- Data journey
- Results

Database modernization with AWS

Modernization at Amazon

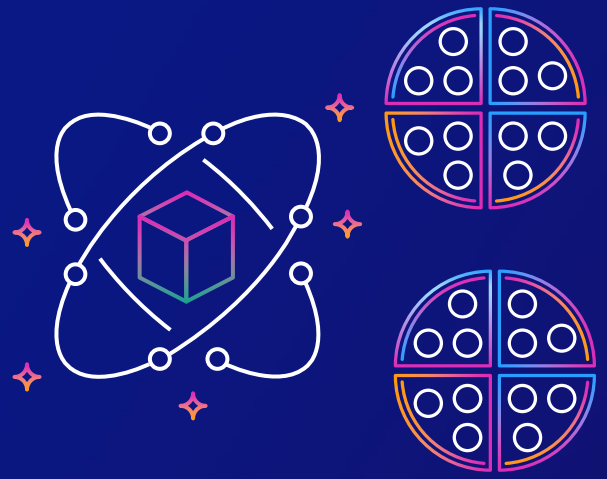
LESSON LEARNED: DECOMPOSE FOR INNOVATION AND AGILITY

Monolithic application



Central database
+ dependent teams

Microservices



Independent databases
+ 2-pizza teams

Scenario

Web servers

Presentation layer

Application servers

Business logic

Database servers

Data layer

Application

3-Tier Legacy Architecture

- End customers access front-end via web
- Based on Commercial Relational Database

Modern data architecture

From traditional . . .

. . . to microservices, decoupled architectures

Web servers

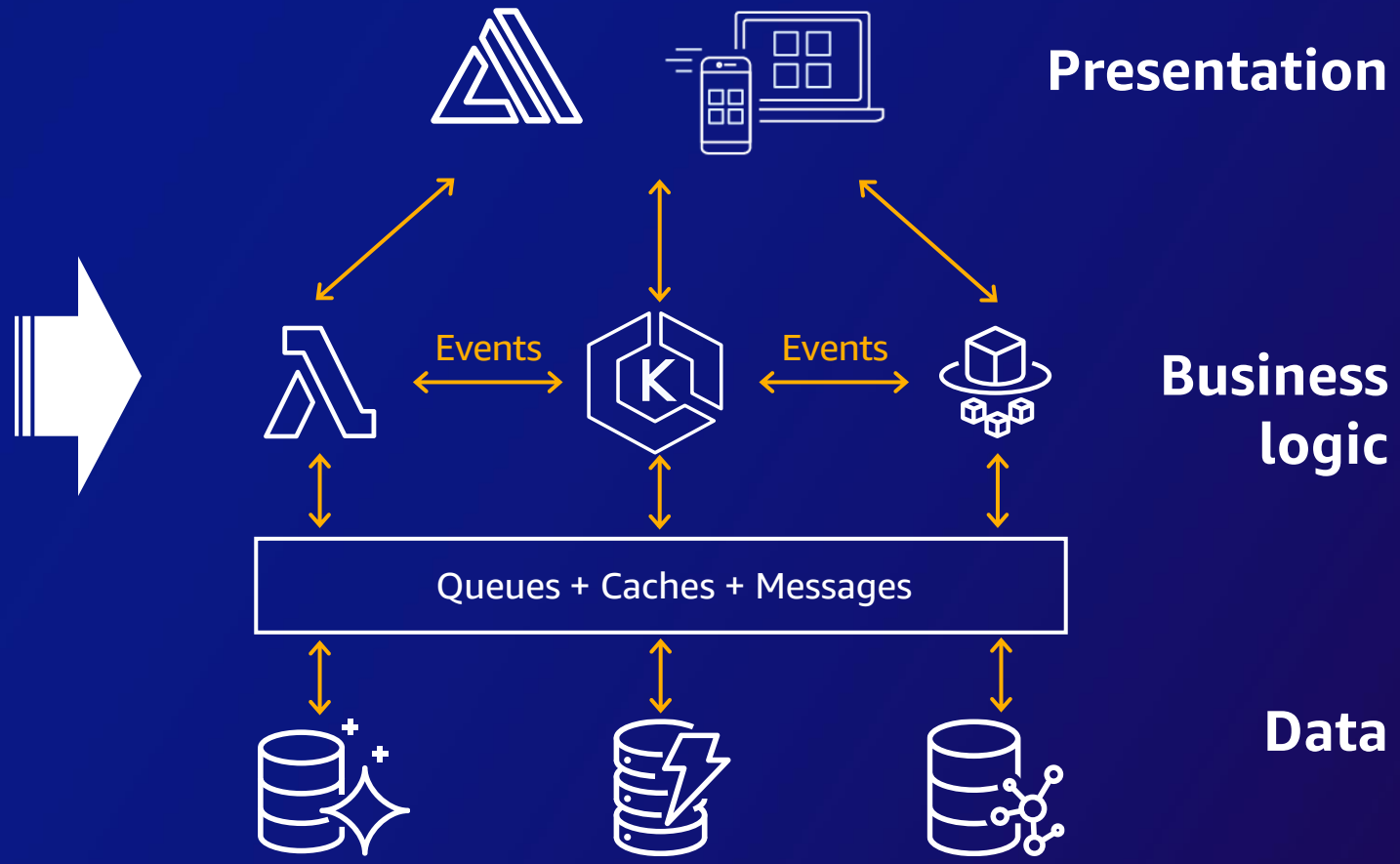
Presentation layers

Application servers

Business logic

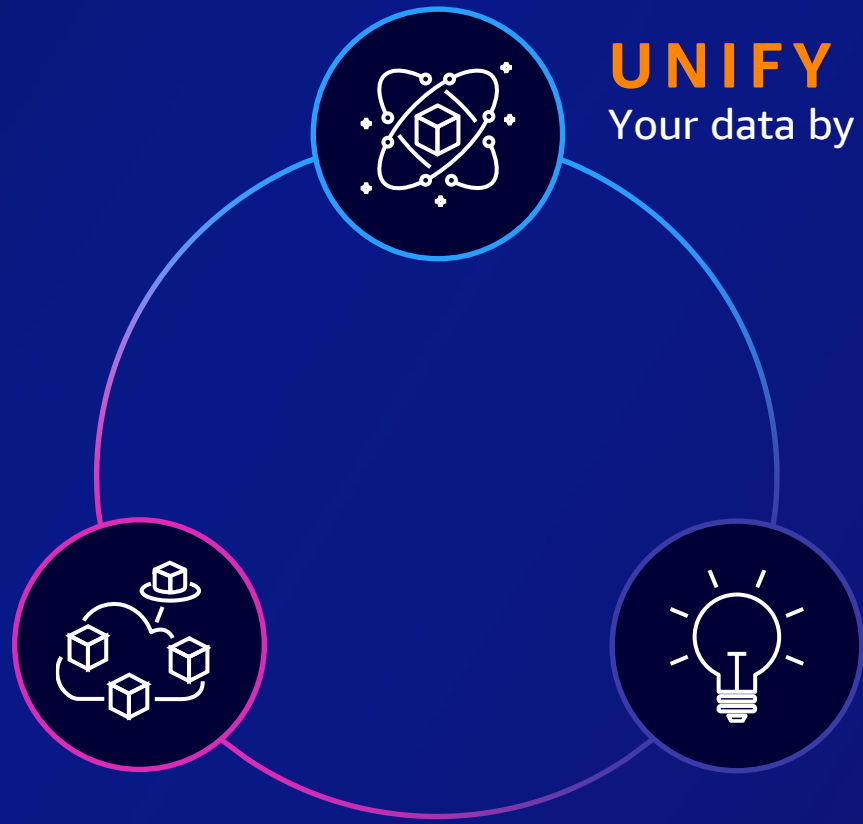
Database servers

Data layer



Modern data strategy for better business outcomes

Start anywhere



UNIFY
Your data by breaking down silos

MODERNIZE
Your data infrastructure to a scalable, trusted, and secure cloud provider

INNOVATE
By inventing new experiences and reimagining existing processes

Step 1: Move to a managed database

Adopt cloud-native relational databases
Amazon RDS, Amazon Aurora

Relational database application

Managed database

Agility, easily managed, cost-effective

- Reduce operational burden
- Take advantage of platform features

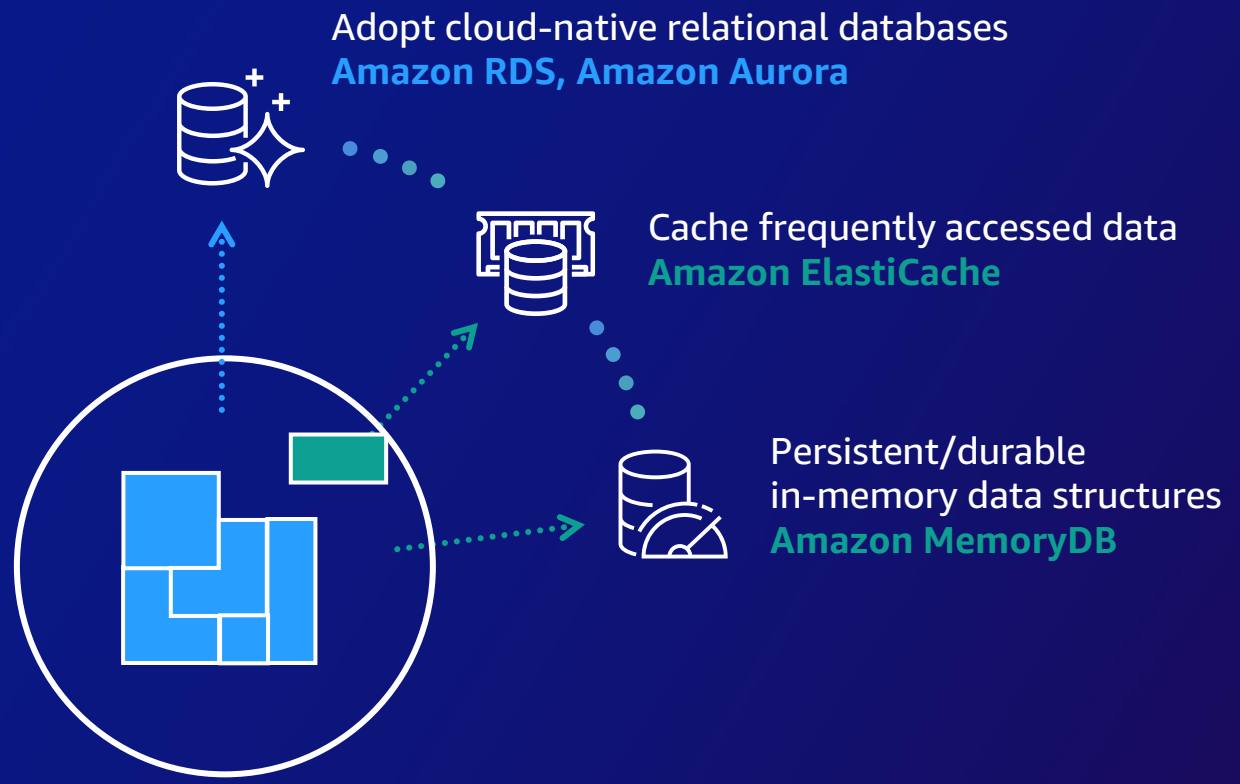
Step 2: Cache and persistent in-memory data

Address scaling pain

- Low-hanging fruit, easy to adopt
- Offload repetitive read operations
- Offload session state
- Caching

Increase performance

- Frequent counters
- Fast-changing rankings
- Submillisecond data access



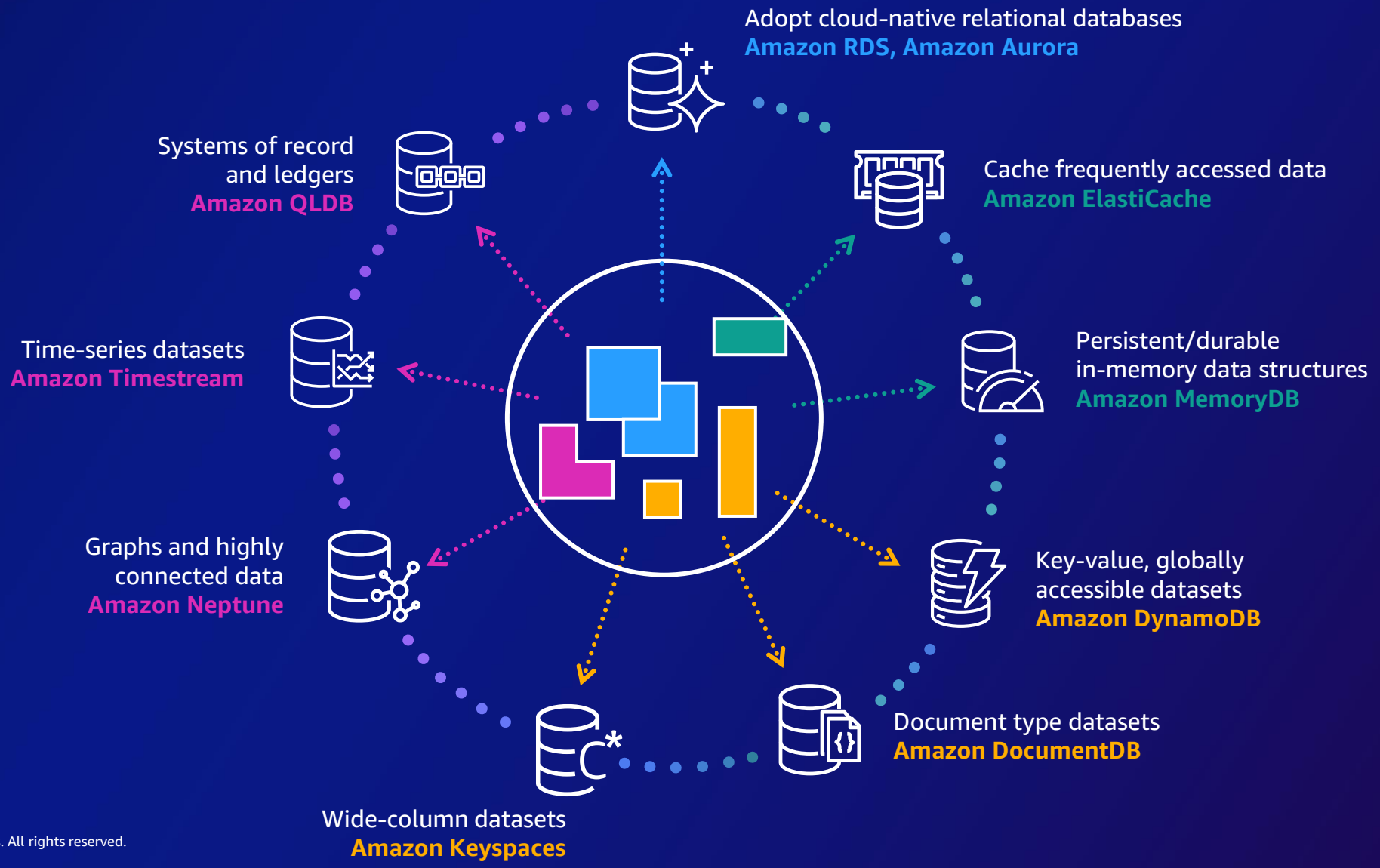
Step 3: Specialized data interactions

Specialized datasets

- Social graphs
- Recommendation engines
- Measurements
- Systems of record
- Digital records

Benefits

- Agility
- Scalability
- Performance



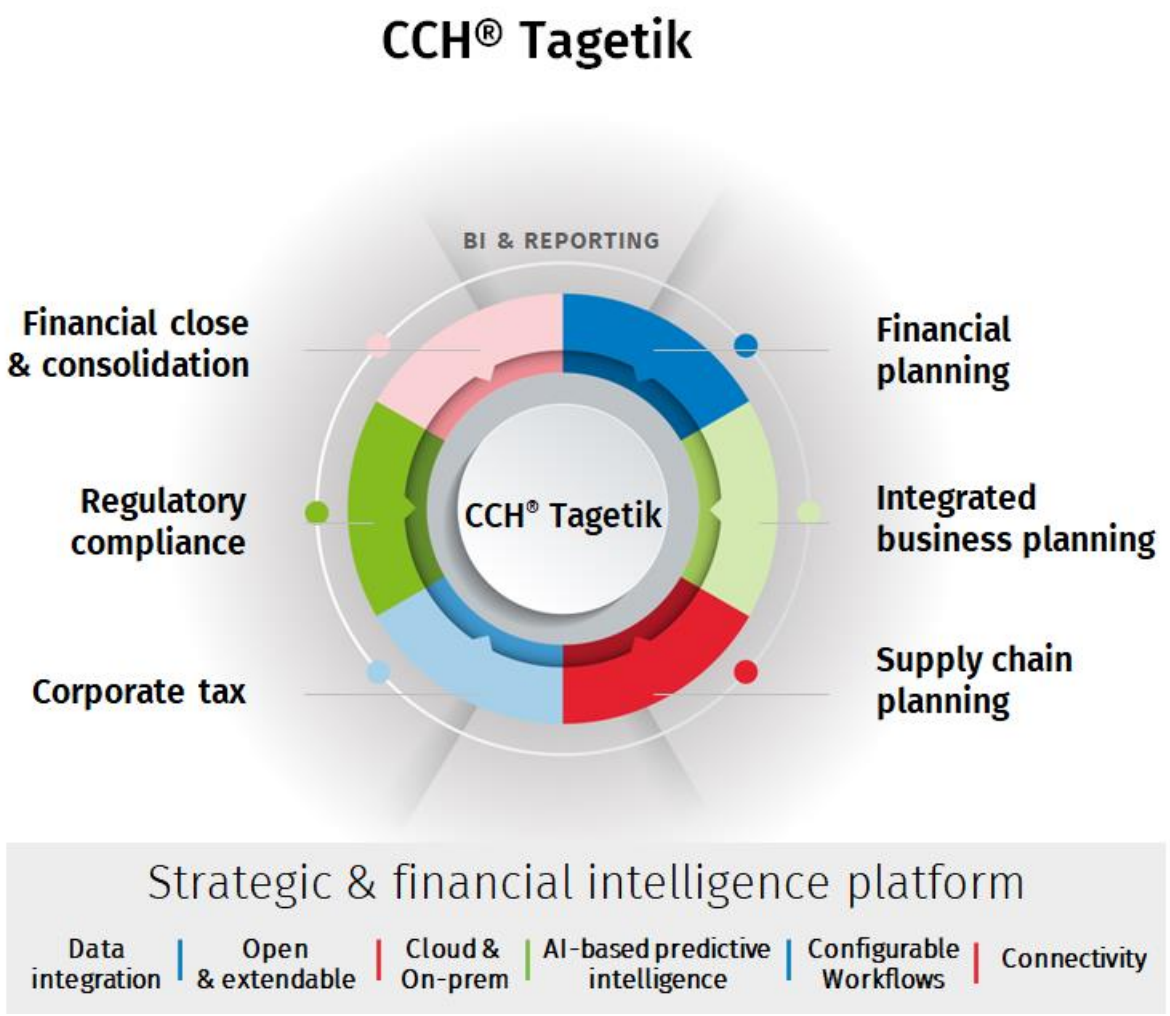
CCH® Tagetik | Wolters Kluwer

Jacopo Roma
Cloud & DevOps Manager
Wolters Kluwer

Fabio Testa
Principal Application & Product Architect
Wolters Kluwer

What's Tagetik

- A data-centric platform
- Office of Finance processes digitalization
- Business critical
- Demanding compliance and regulatory requirements
- Expert Solution
- DB Independent – 4 RDBMS Supported



What's Tagetik


Insurance


Banking and Financial Services


Fashion


Manufacturing


Construction


CPG & Retail


Services


Automotive


Power & Utilities

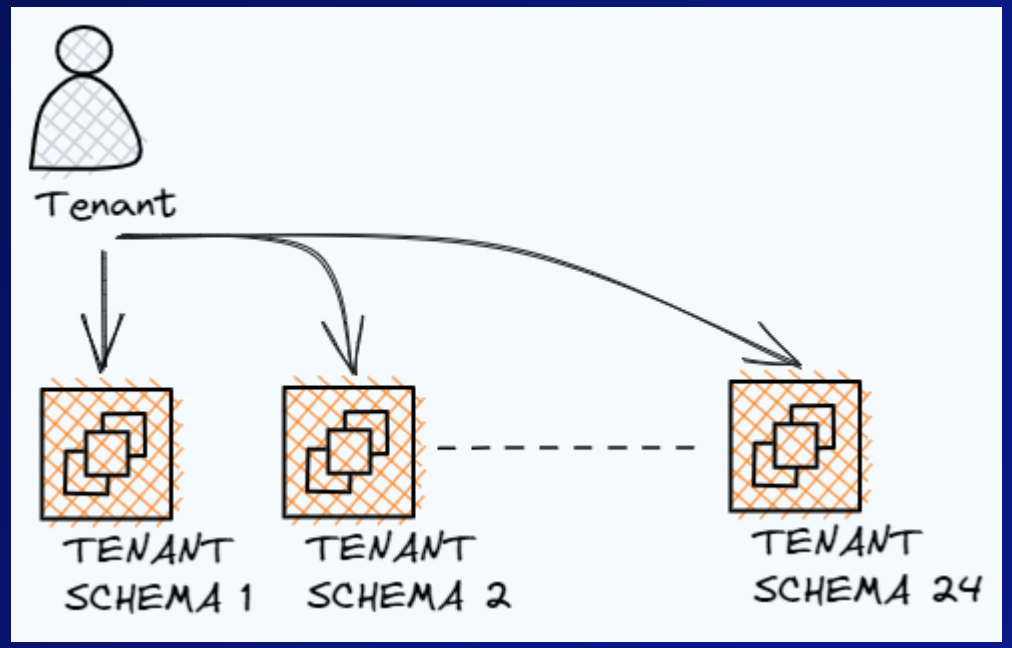
GENERALI
BNP PARIBAS
TIMEX
Krauss Maffei
VINCI
Vestey Foods
esa
DAIMLER
EDISON
europ assistance
you live we care
CORESTATE Capital Group
SCOTCH&SODA AMSTERDAM COUTURE
AGFA
MTHøjgaard
METRO
ambea
Sumitomo
Anglian Water Group
AEGON
南京银行 BANK OF NANJING
PRADA
MAHLE
Keepmoat
unicoopfirenze
arena GROUP
SOGEFI GROUP
uniper
ACR Asia Capital Reinsurance Group
Rabobank
BREITLING
LEONARDO
ASTALDI
Dr.Oetker
aggreko
上汽大众 SAIC VOLKSWAGEN
VEOLIA
Bupa
BPER: Banca
MARCOLIN
ArcelorMittal
Green Courte Partners
Keurig DrPepper
ROPES & GRAY
TOYOTA
TenneT Taking power further

Tagetik Cloud SaaS

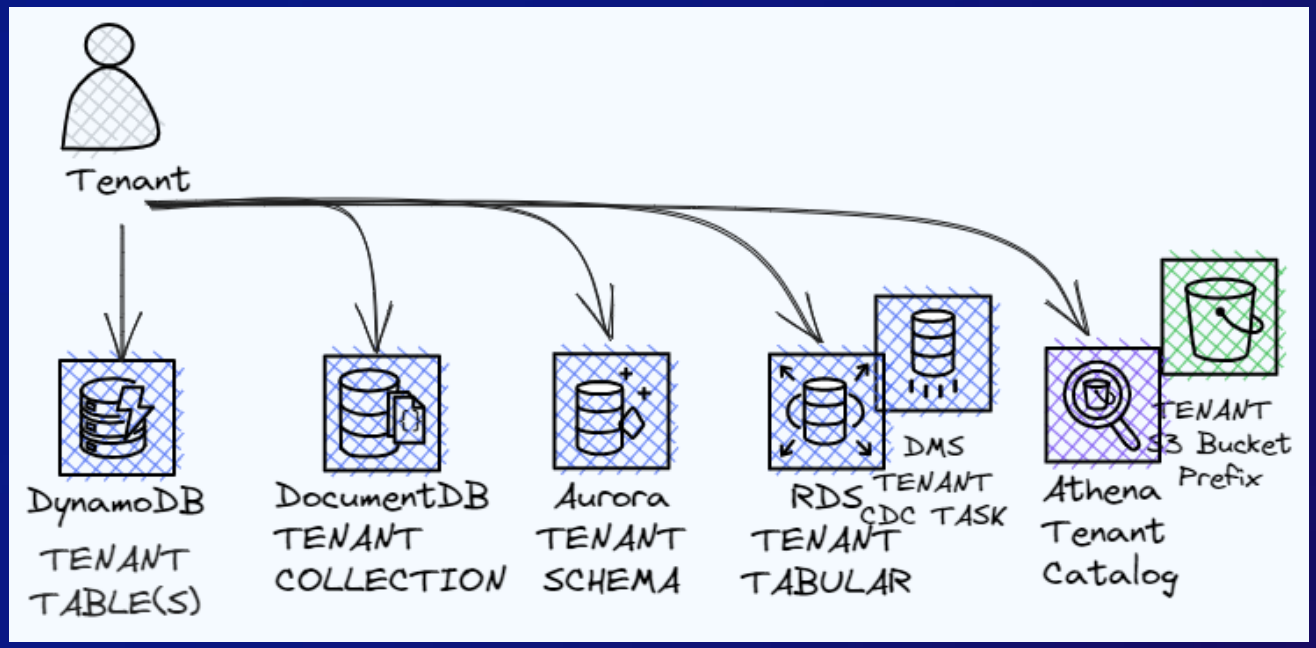
- More than 50% of our Customers are Cloud Customers
- ~1800 tenants (production environments)
- 9 AWS Regions
- 1500+ Servers and 6000+ Databases
- 9+ PB Customers data
- ISO 27001/22301/9001
- SOC1/2 TYPE II

Tagetik Cloud SaaS - Challenges

3 Tier Architecture



Microservices and Cloud Native



Data requirements and multitenancy

What is a tenant?

In the realm of web applications, a "tenant" represents a "**distinct entity**"

Typically a **customer** or an **organization** that operates within a secure, isolated environment of a **shared infrastructure**.

A **unique user group** that has its own set of resources, data, and configurations.

Multi-tenancy options

Application-Level Multi-Tenancy

In this scenario, each tenant can have their own instance of the application, running on the same or separate server, or with the same application instance manage one or more tenant. In any case they share the underlying software codebase.

Database-Level Multi-Tenancy

Here, each tenant shares the same database server but has their own unique schema, tables. This provides efficient use of server resources, but requires rigorous data access controls to ensure data privacy and security.

User-Level Multi-Tenancy

In this case, each user within the system is considered a tenant. This is common in B2C applications where each user's data, preferences, and functionality may be segregated and secured.

Hybrid Multi-Tenancy

A combination of application, database, and user-level multi-tenancy can be used, depending on the application's requirements and the data privacy needs of the tenants.

Tagetik Architecture flavors

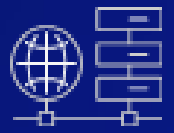
Single Tenant Monolith

Tenant is here referred to **application,** we cannot manage more than one customer with the same application

Single tenant application can manage 24 primary and 64 secondary data sources (Oracle, SAP Hana, SQL Server, Postgres, Amazon Athena)

1 DB to manage user authentication and authorization

23 DB to manage financial and analytical workspace



Multitenant Monolith

Tenant is here referred to **application** level and **database** level. We can manage more than one customer with the same application

Multi-tenant application with a proprietary database solution (next future will use natively **AWS documentDB**)

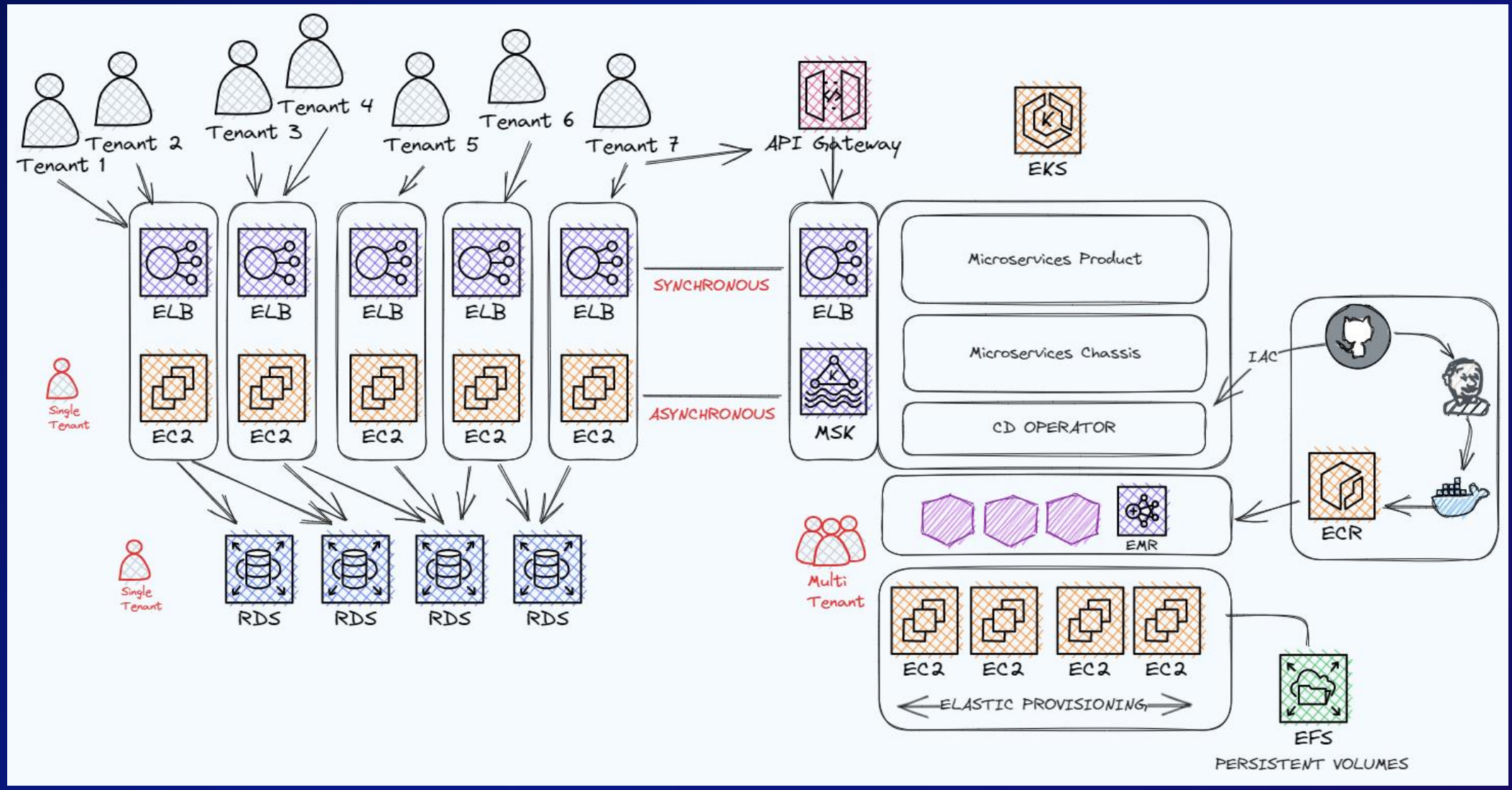
With the same application deployment we can manage more than one customer

Microservices

Tenant is here referred to application level and database level.

Microservices are designed to be cloud-native and multitenant-native at application and database-level

Tagetik Architecture flavors



Tagetik Datastores

ORACLE

Microsoft SQL Server

SAP HANA

PostgreSQL

Amazon Relational Database Service

Amazon DocumentDB

Amazon DynamoDB

Amazon Simple Storage Service (Amazon S3)

Amazon Elastic File System (Amazon EFS)

Amazon Athena

Amazon Aurora

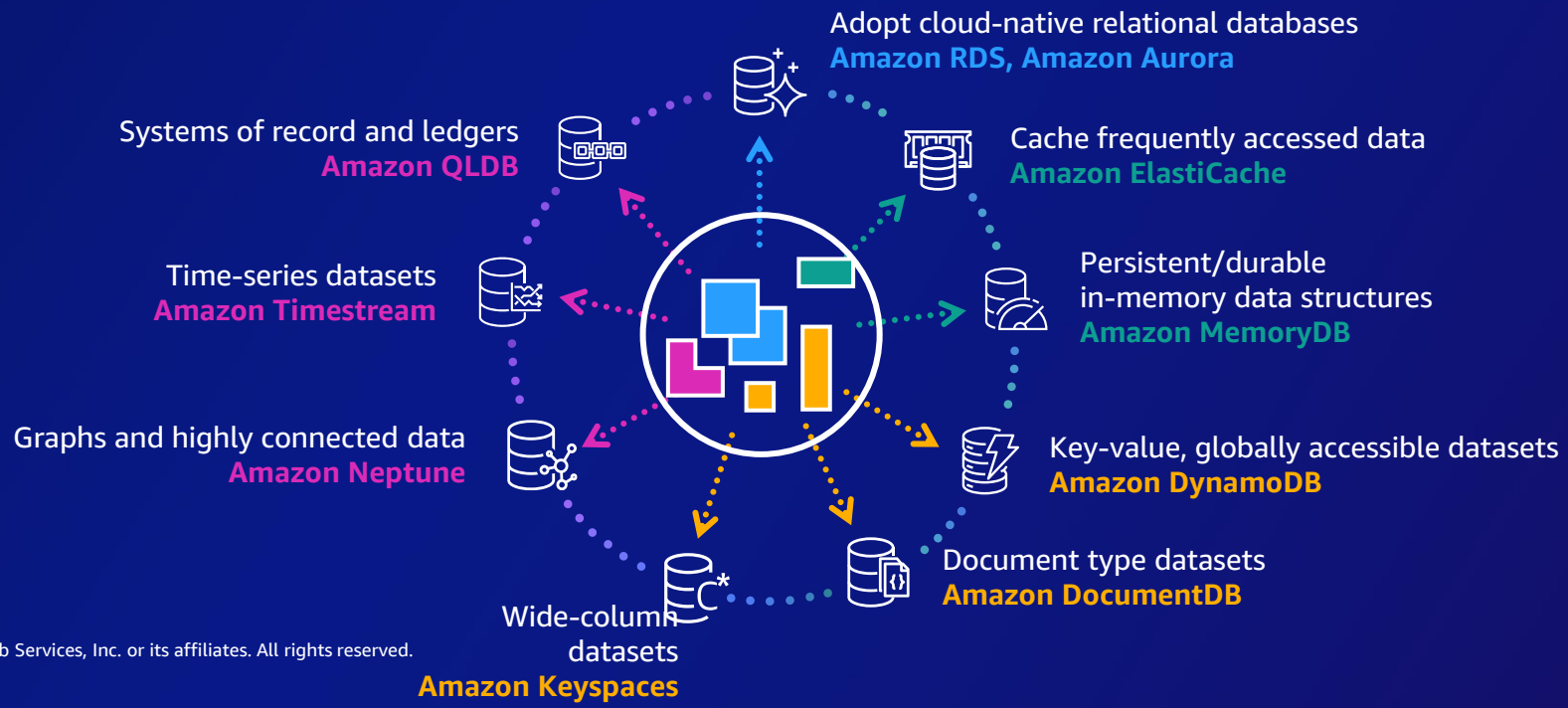
Amazon EMR

Amazon ElastiCache

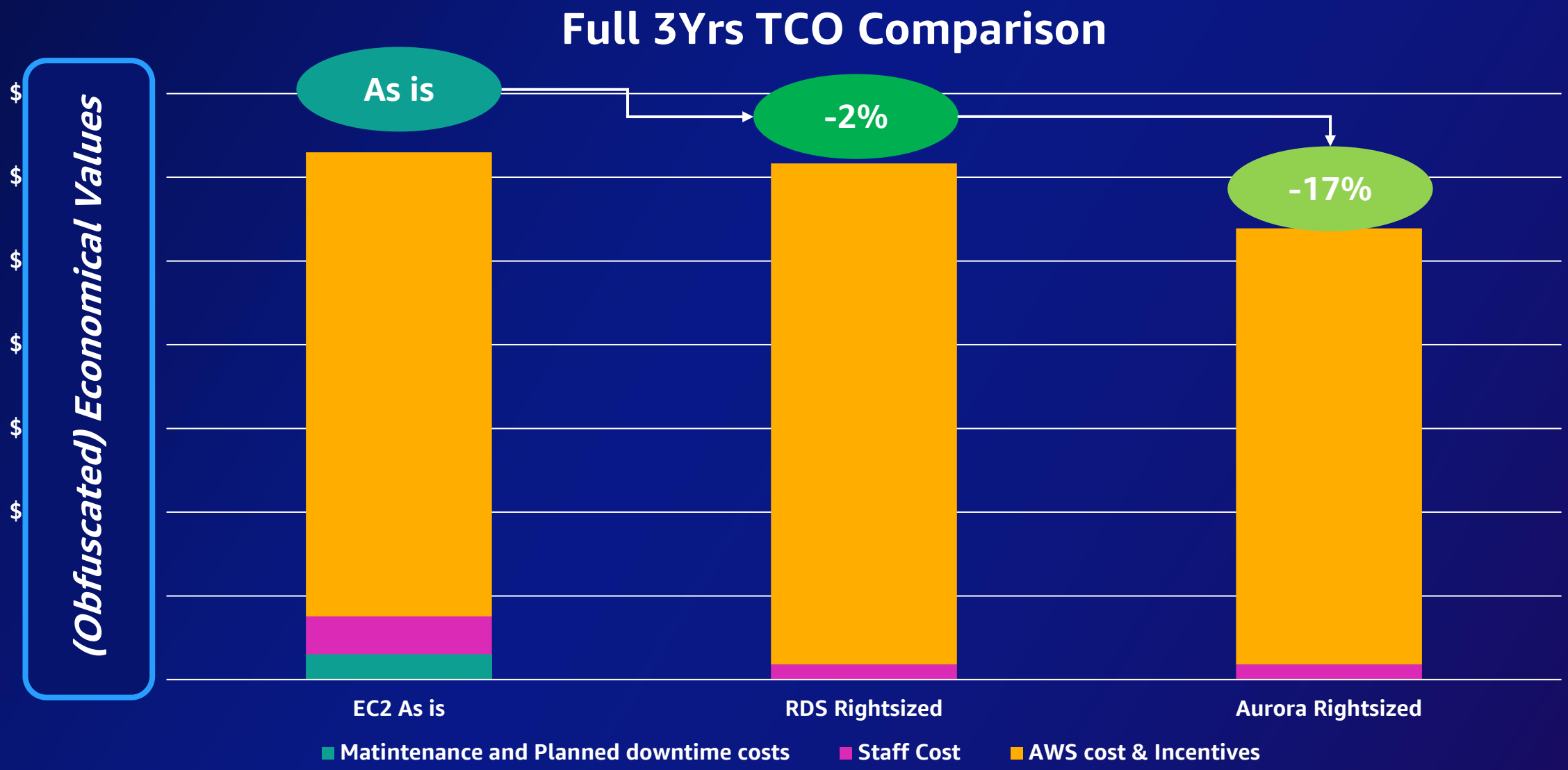
Managed Databases – Journey to adoption

Why Tagetik looked at managed databases

Agility	Accountability	Efficiency	Security
Decentralized Cloud DevOps	**From Product to Service**	More performance	Simplify and enhance DBA activity auditability
Time to market for new features	FinOps by Functional Area	**Less maintenance, more development**	Simplify BYOK implementation
Cloud Native development	Decentralized monitoring and response	Ops scalability to support business growth	**Simplify Compliance and Certifications assessment process**



Why Tagetik looked at managed databases



Decision Making

Management Overhead

RDS handles tasks like backups, patch management, and failover. Improve our RPO/RTO down to few minutes

Performance Requirements

We deliver RDS offers automated scaling, beneficial for apps with variable loads.

Availability & Resiliency

RDS provides enhanced availability, like Multi-AZ deployments and simplify our Business Continuity Plan

Reduced Time-to-Market

Transitioning to Managed Services can free up your platform team's time for development, speeding up your time for automation.

Migration - From EC2 to RDS

Assessment with AWS support

Understand your current database setup, workload, and performance needs. Identify compatibility issues

Planning

Choose the right RDS database engine, instance size, and storage type. Map your recipes from base installation on EC2 to RDS

Testing & Optimization

Use AWS support to optimize loads and performances, use automated tests to set limits and find the proper behaviour

Execution

Use the automations to move customer data with minimum downtime

Managed Databases – Organization and Governance

* Everything as code

1. Infrastructure Code

Infrastructure code effectively describe and provide the cloud architectural components (service catalogue)

It’s packaged as a module (can be stored on git itself) and has a state to be maintained

It is strictly linked with cloud providers

It's written by the Platform or the Product Dev Teams

2. Applicative code

Applicative code contains the business logic

It’s packaged as a docker image

It is actually the microservice itself and all languages can be used (java, python, c#)

It’s written by the product dev team

3. Templating code

Templating code contains the deploying logic for the applicative code

Templating code is packaged as an helm chart

Templating code is also used to deploy the core services (metrics, visualization, tracing)

It is written by platform or product dev team (templates are provided from platform team)

Database as code – stateful deployment

Provisioning & Deprovisioning

We can provision fully-configured RDS instances in a snap, and when their time comes, elegantly decommission them, all in just a dozen of minutes

Backup & Restore

With the harmony of AWS RDS and Terraform, we can rest easy knowing our data is safeguarded, enabling us to seamlessly roll back the hands of time after any disaster - it's like having a time machine for your data, always ready for action.

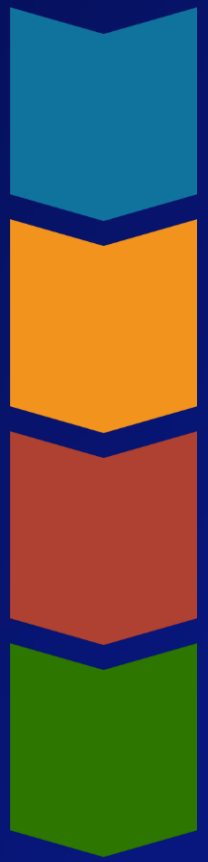
Database upgrades

Through the synergistic blend of Terraform and AWS RDS, we can manage version upgrades with minimal downtime, maintaining a continuous service

Modular and Reusable

Leveraging Terraform's modularity with AWS RDS is akin to professional architecture with digital LEGO. Each reusable module adds to a powerful, adaptable, and streamlined infrastructural design, promoting both efficiency and innovation

Microservices Organization changes



No Ops philosophy **must** be **followed** due to **high complexity (no manual tasks)**

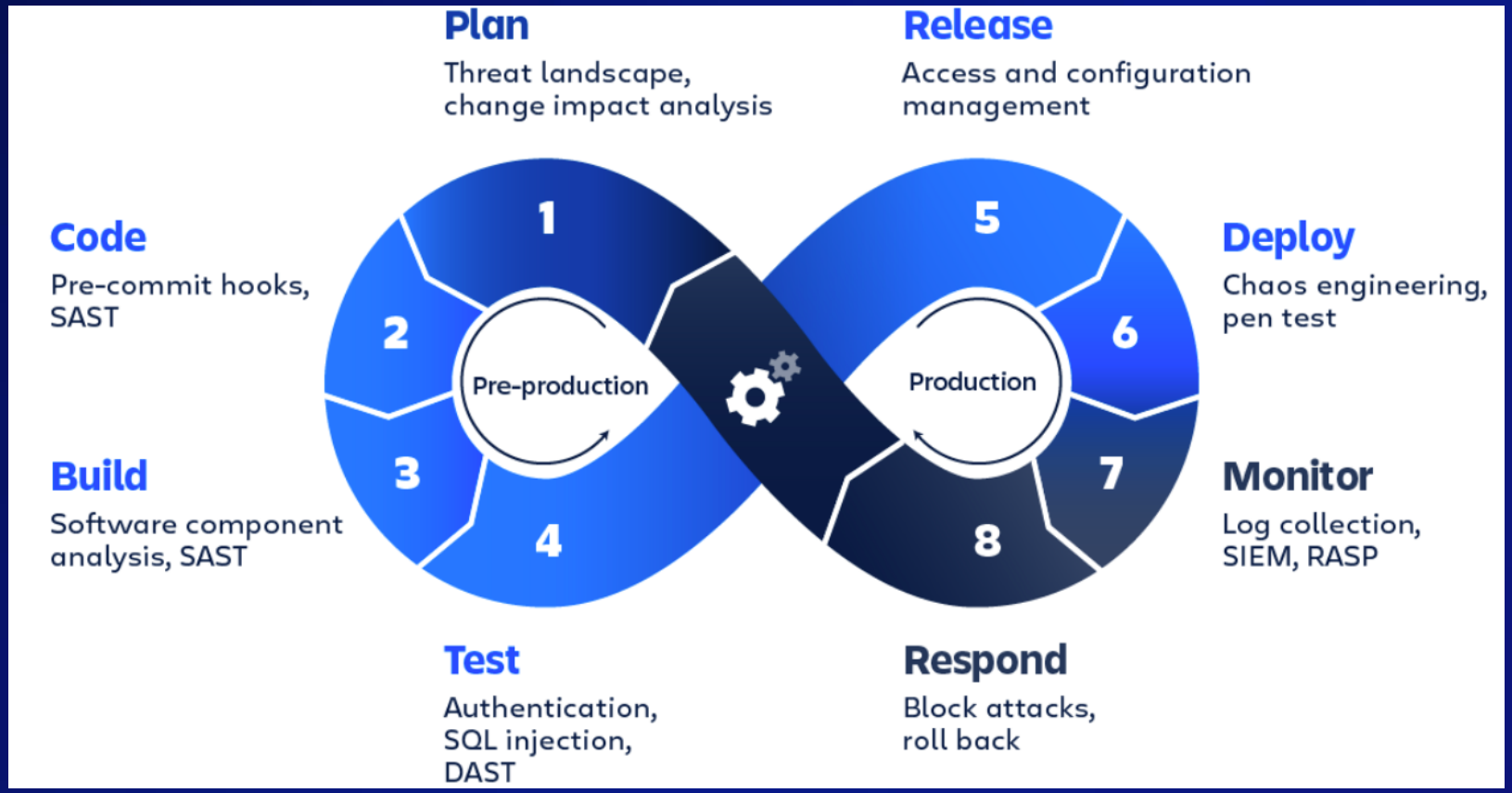
Infrastructure code must be **developed** as a very **first step** and also **developers training**

Release Manager is now the **Dev Scrum Team** and no more a **central team**

Product Owner can **deliver value** at **convenience. No more 3 months release process**

Technological changes require organizational changes

DevSecOps Lifecycle



- Change of **paradigm** from **Monolith** where you've got **dedicated teams** (Developers, Builders, Testers, Cloud Ops to manage services)
- Inside this **infinite cycle** a **team** should be **autonomous**
- Have the **pre-production** and **the production** steps **strictly separated** to properly manage this 2 blocks

You build it,
You secure it,
You run it,
You own it!

Governance & Compliance

Governance

Governance describe the overall **management approach** trough which managers **direct and control the organization**

Compliance

Compliance means **conforming** with **stated requirements**... for example **laws, regulation, contracts, strategies and policies**

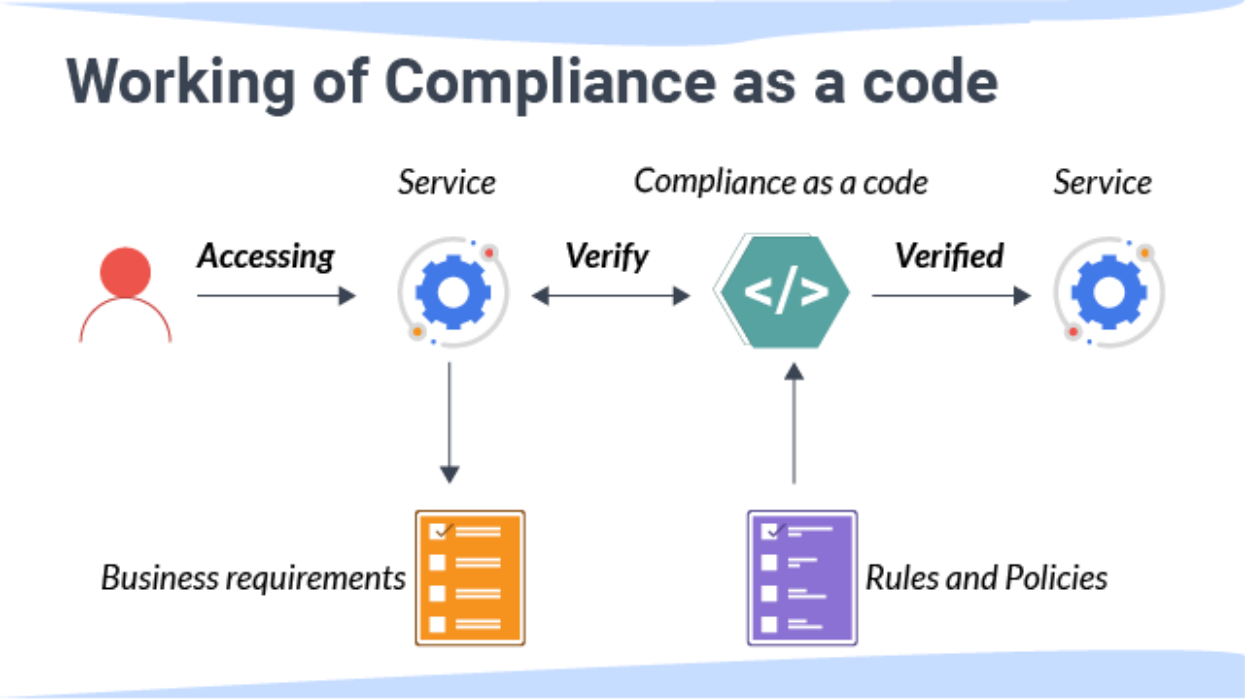
Validation

The **assurance** that a product service, or system **meets the needs of the customer** and other identified stakeholders

Verification

The **evaluation** whether a product, service, or system **complies with a regulation**, requirement, specification, or imposed condition

Compliance as code



Representing your **compliance requirements in code** that can be used to test configuration of your environment to determine **whether it complies with the specification**

Native **CSP** tools (**Content Security Policy**)

- **AWS System manager compliance**
- **Kubernetes** compliance manager (Pod Security Policy)

Opensource Tools

- **Cloud Custodian**

ISO 27001: 2019

Control: A.13.2.1
Information transfer policies and procedures

AWS

IAM Policy definition

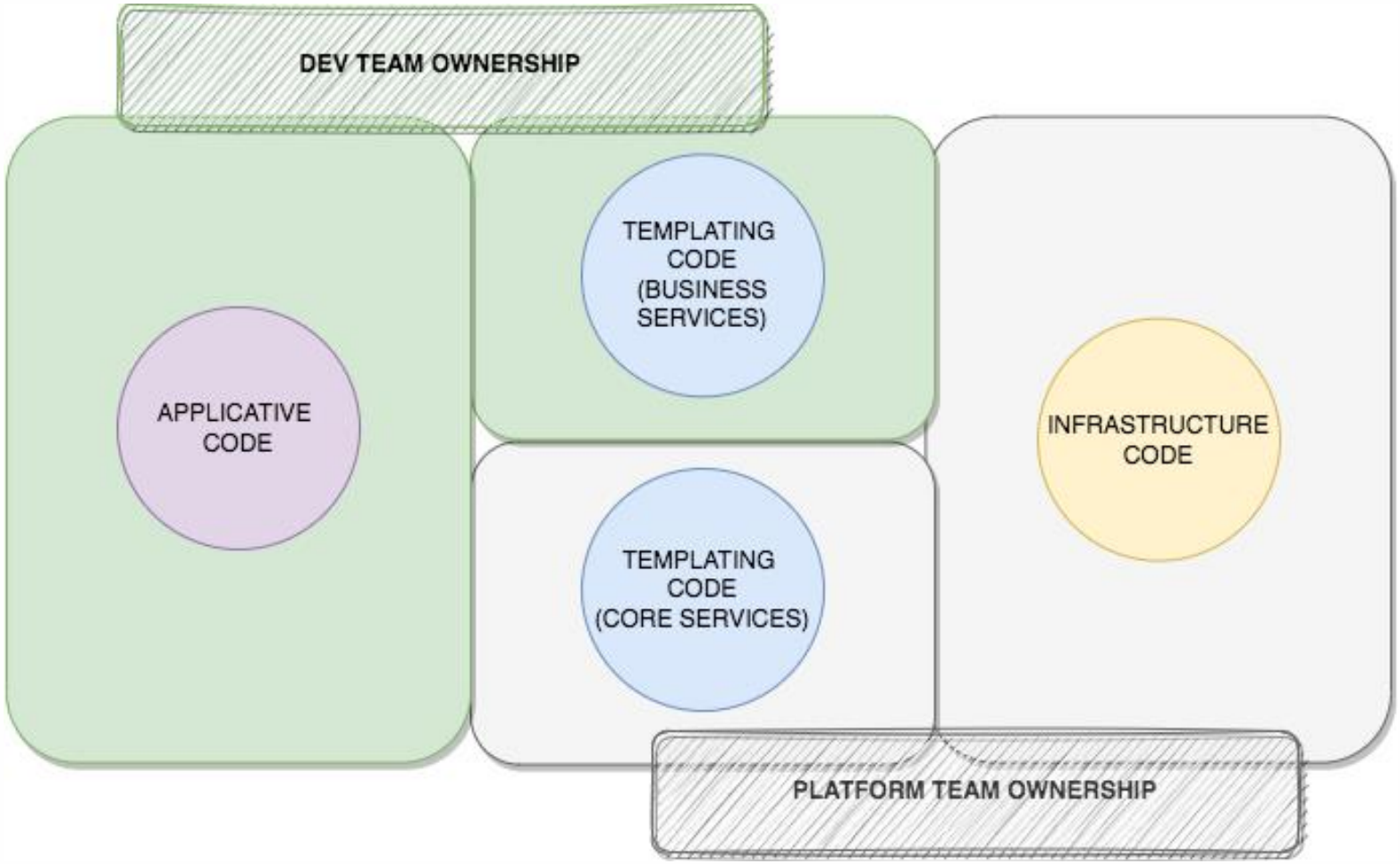
Team Ownership – Decentralize to DevSecOps

Phase 1

1. To develop and **create the platform** and especially having only one microservices feature **was a good start**

2. But then adding more and more features **to speed up the process** we understand the need to marry the DevOps principles and **let product dev team to contribute to infrastructure code**

3. Moreover by adding **serverless app** we understand applicative code is not needed anymore but **the final artifact is the infrastructure code itself** (terraform code)

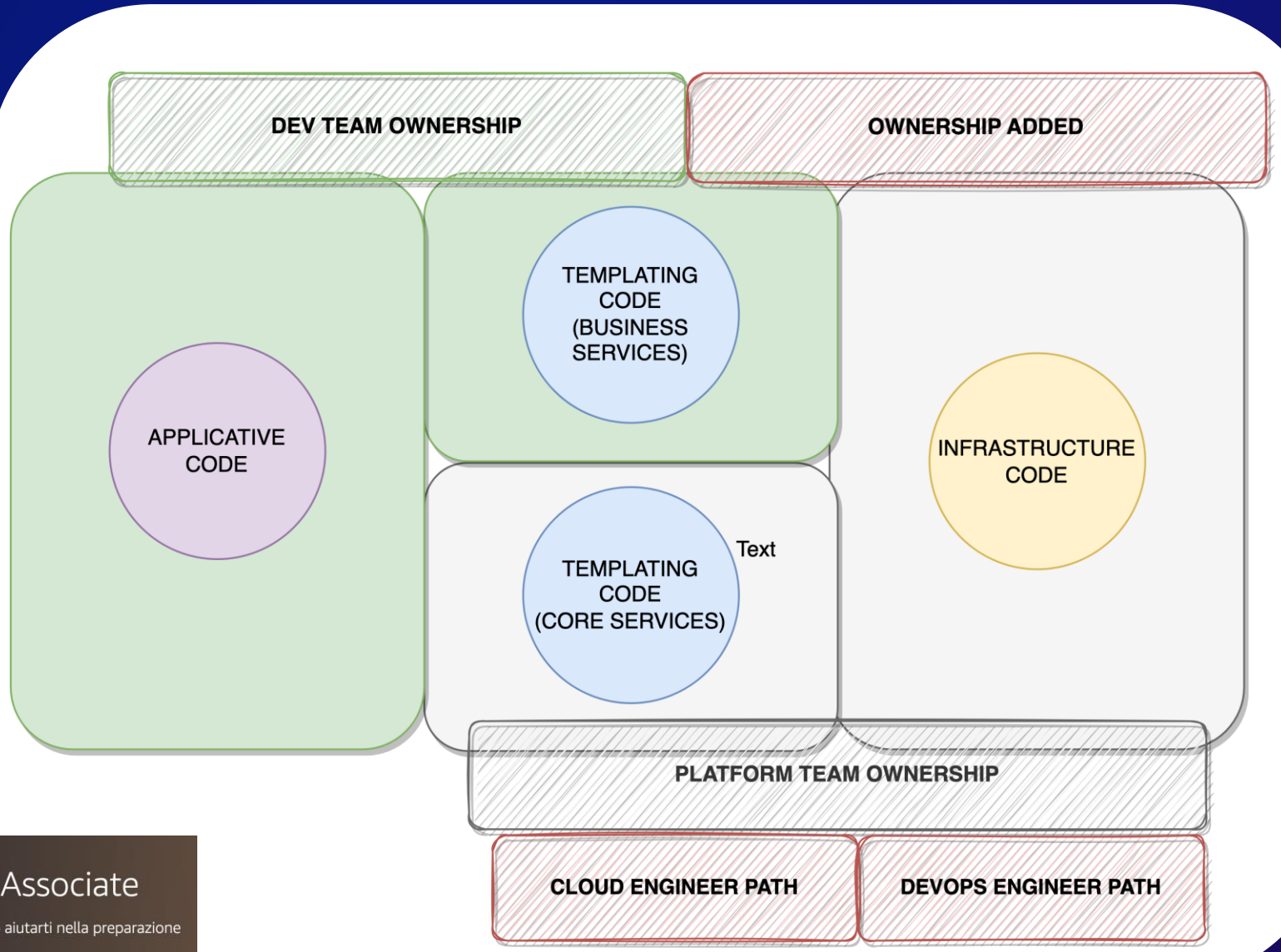


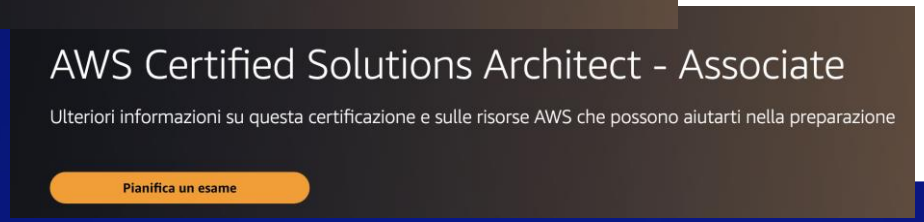
Phase 2

Adding that ownership must be followed by the **training of product dev teams with devops competencies.**

We started with **8 teams (4 functional areas)**

By adding this new processes we understand also the need of **changing the platform and dev teams organization itself**





Phase 3

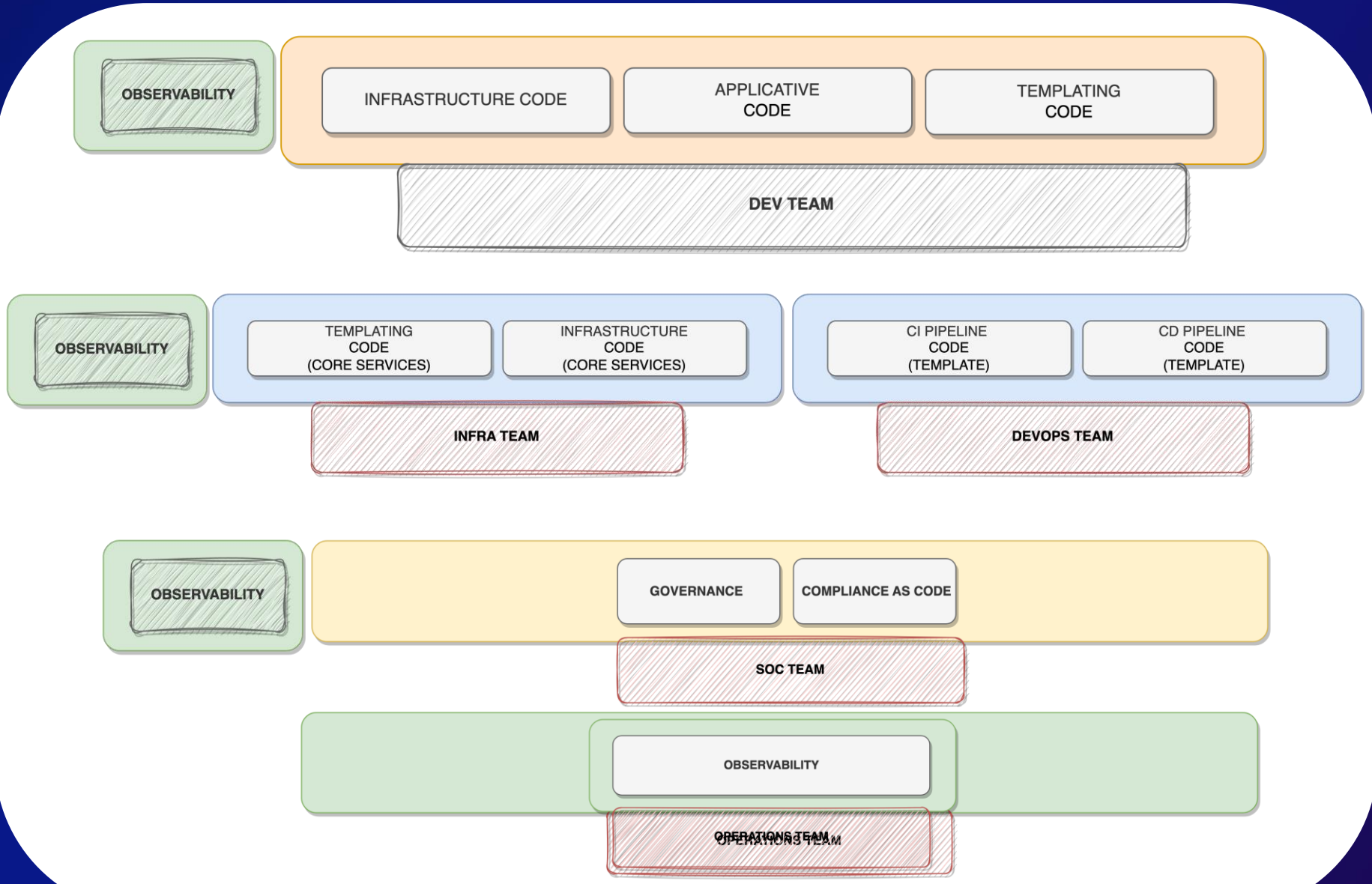
“Divide et Impera”

1. We used an agile framework (Safe) for efficient team collaboration and swift, high-quality code delivery.

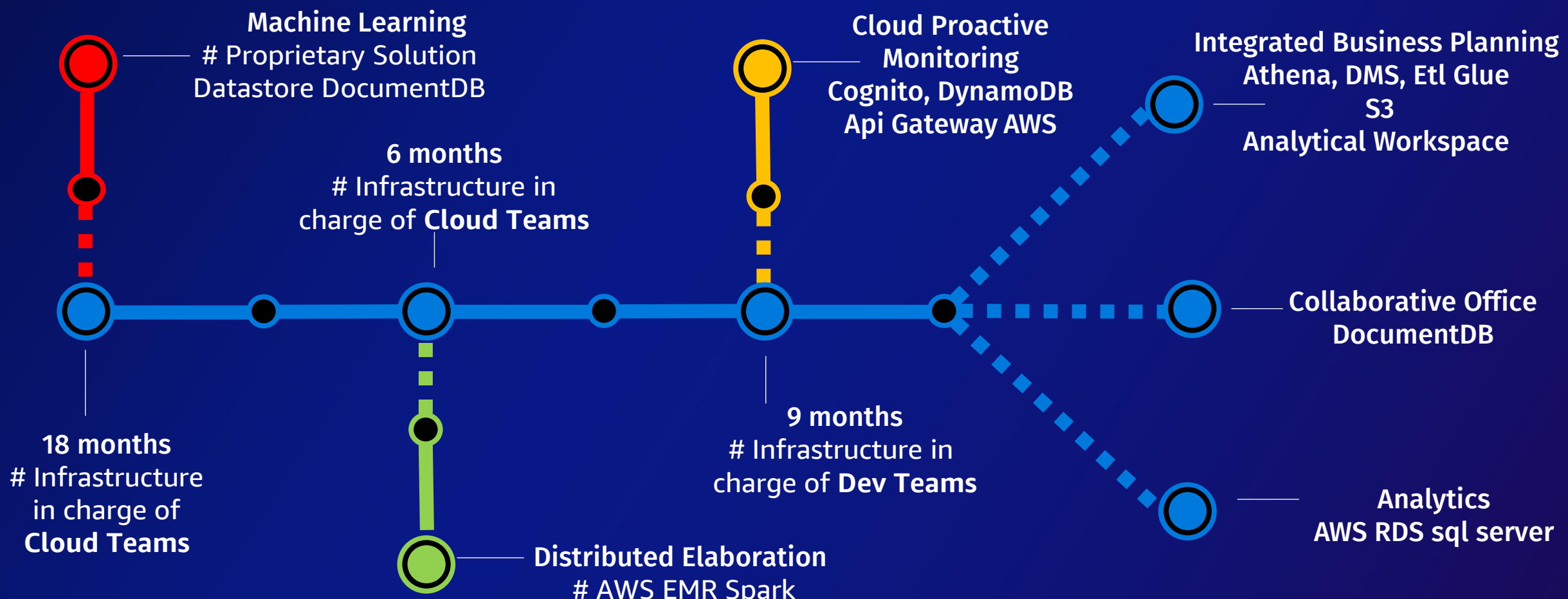
2. Form an Infrastructure Development Team to manage infrastructure code and services catalog.

3. Set up a DevOps Team to create CI/CD templates, improving development process efficiency.

4. Establish a SOC Team to manage compliance and governance via code-based methods, adhering to various security and compliance standards

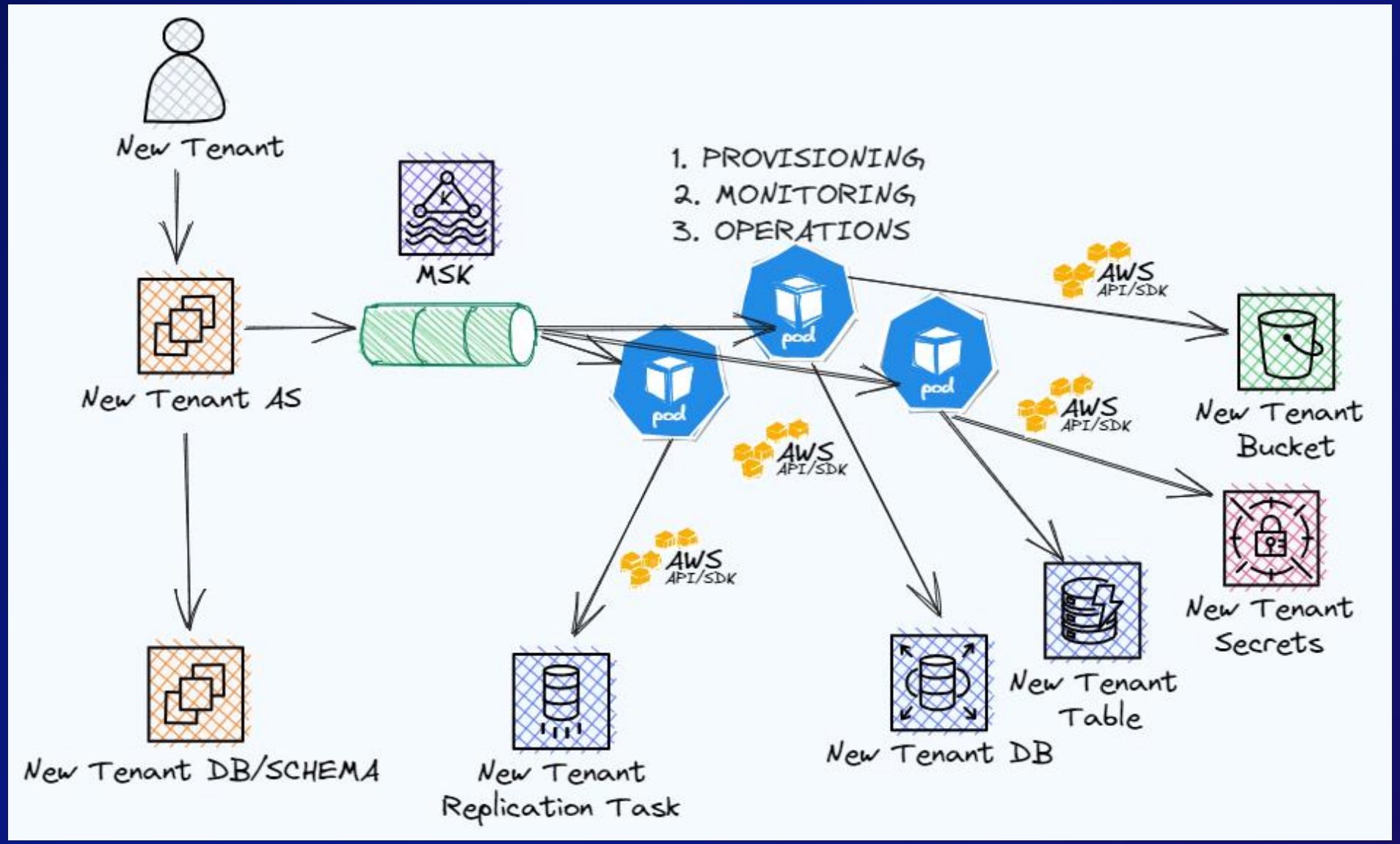


TimeLine



Managed Databases – Design benefits

How the Architecture changes



Lessons learnt

Tagetik module	Tagetik value	Customer value
Machine Learning	Deploy new regions EKS in hours (Kafka, Redis, DocumentDB)	Scalable ML models training and inference
Distributed elaboration (EMR)	Deploy EMR on EKS in minutes	Big Data processing moved from hours to minutes
Self service portal	Deploy DynamoDB, Cognito, AWS Api Gateway in minutes + S3 bucket microfrontend	Customer can monitor the health of application every second from a dedicated console
Integrated Business Planning	Deploy serverless application DMS, Athena, ETL Glue, DocumentDB in minutes	Performant and scalable Reporting features

Thank you!

Please complete the session survey in the mobile app

Mattia Berlusconi
Solutions Architect, Data Specialist
Amazon Web Services

Jacopo Roma
Cloud & DevOps Manager
Wolters Kluwer

Fabio Testa
Principal Application & Product Architect
Wolters Kluwer