Clean Architecture

Wednesday, 18 September 2024 7:25 AM

Scenario: Telecom Service Provider - Modernizing their Systen
Introduction: Your telecom service provider company, *TelStar*, across various regions. The company is looking to upgrade its sy scalability, and flexibility in its operations, from managing custo network data efficiently. This will also help in launching new se Let's see how various architectural concepts help achieve this m

1. Domain-Centric Architecture: TelStar realizes that their curre monolithic fashion, making it hard to manage changes. They de into **domains**:

- **Customer Management** (Handling subscribers and their de
- **Billing System** (Tracking customer payments and bills)
- **Network Operations** (Managing signal quality, usage, etc.)
- **Support Services** (Handling customer queries, complaints)

Each domain is an independent unit but works together to mak smoothly. This separation brings focus and simplifies both main **Analogy:** Think of it as dividing the telecom company into speci Customer Care, Billing, and Network Services. Each department works in harmony to provide the service.

2. Application Layers: Each domain needs its own structure to m TelStar decides to organize their codebase using **layers**:

- **Presentation Layer:** How users (customers or employees) i (Mobile apps, websites)

ns

has millions of subscribers
ystems for better performance,
omer subscriptions to handling
rvices like 5G more smoothly.
nodernization.

ent system is built in a
cide to break the system

tails)

e the telecom system run
ntenance and feature updates.
alized departments like
t (domain) has its own tasks but

manage operations effectively.

nteract with the system.

- **Service Layer:** Logic that processes requests, such as activa
- **Data Layer:** The database where all the information like cus records are stored.

Example: When a customer wants to upgrade their mobile plan presentation layer (app), the service layer checks eligibility, and records.

3. CQRS (Command-Query Responsibility Separation): TelStar' issues when managing simultaneous actions like viewing netwo customer profiles. To solve this, they implemented **CQRS**:

- **Command:** Handles actions that change data (e.g., adding a plan).
- **Query:** Handles actions that only read data (e.g., checking r history).

Example: When a user checks their current data usage (query), with the process of adding new users (command), ensuring fast

4. Event Sourcing: In TelStar, every action—like a customer upg data usage—is tracked as an **event**. Rather than just updating re event that occurred over time, allowing them to track all chang **Example:** If a customer upgraded their plan multiple times, TelS plan changes and track data usage during each period, helping i

5. Functional Cohesion: Each domain within TelStar has its own ensuring **functional cohesion**, the company makes sure that rel same domain, without unnecessary dependencies.
Example: The **Billing System** handles everything related to cust mix with **Network Operations**. This separation ensures that bill

ting a new plan for a customer.
stomer details and billing

, the request goes through the
l the data layer updates the

s system had performance
ork usage and updating

a new customer or changing a

network usage or billing

the system doesn’t interfere
ter operations.

grading their plan or network
ecords, the system stores every
es.
Star can retrieve a full history of
in making better offers.

tasks and operations. By
lated actions stay within the

omer payments and does not
ing updates don’t interfere

with monitoring the network.

6. Bounded Contexts: Each domain in TelStar (Customer Manag
own **bounded context**, meaning the data and rules in one doma
another. This keeps operations clear and prevents confusion.
Example: The **Billing** domain may have a different definition of
the **Network Operations** domain, where "active" refers to data
status.

Conclusion: By adopting these architectural patterns, TelStar ca
services like 5G faster, and handle millions of customers with in
security. This also allows them to manage specific business proc
gives them a competitive edge in the telecom market.

1. Domain-Centric Architecture:

- **Framework:**
  - **DDD (Domain-Driven Design)** by Eric Evans is the core
    centric architecture.
- **Supporting Technologies:**
  - **Spring Framework (Java)** – The Spring ecosystem provi
    implementing domain-driven design with modularity ar
  - **NestJS (Node.js)** – A progressive Node.js framework th
    architecture, especially when building scalable server-s

2. Application Layers:

- **Frameworks:**
  - **Layered (N-Tier) Architecture** – This is one of the most
    applications, with separation into layers like presentati

gement, Billing, etc.) has its
ain do not overlap with

“active customer” than
usage instead of payment

an scale better, launch new
mproved performance and
cesses more effectively and

principle behind domain-

des excellent support for
nd separation of concerns.
at promotes domain-centric
ide applications.

common patterns in enterprise
on, business, and data

applications, with separation into layers like presentati
- **Supporting Technologies:**
  - **ASP.NET Core (C#/.NET)** – ASP.NET Core MVC offers a g applications with distinct layers for presentation, busin
  - **Angular / React (Frontend)** – These JavaScript framew Presentation layer in enterprise applications.

3. CQRS (Command-Query Responsibility Separation):

- **Frameworks:**
  - **Axon Framework (Java)** – A popular CQRS and Event So applications. It helps separate command handling from
  - **MediateR (C#)** – A simple in-process messaging library (commands/queries) from the handlers in .NET applica
- **Supporting Technologies:**
  - **EventStore (Database)** – A specialized database that su CQRS, helping to store and retrieve events efficiently.
  - **Apache Kafka** – Can be used to implement asynchrono command and query events through distributed messa

4. Event Sourcing:

- **Frameworks:**
  - **Eventuate (Java)** – Provides libraries for building applic and CQRS.
  - **Lagom Framework (Java/Scala)** – A microservice frame sourcing and CQRS, with a focus on reactive architectu
- **Supporting Technologies:**
  - **EventStore** – This open-source database is specialized i streams of immutable events.
  - **Couchbase / Cassandra** – NoSQL databases are often u architectures to store events efficiently.

5. Functional Cohesion:

on, business, and data.

good structure for creating
ess, and data.
orks are commonly used as the

ourcing framework for Java
query handling.
to decouple requests
tions.

upports event sourcing and

us CQRS patterns, handling
ging.

cations using Event Sourcing

ework that supports event
re.

in event sourcing, storing

sed in event sourcing

- **Frameworks:**
  - **Functional Programming Languages** – Languages like **S** and **F#** naturally support functional cohesion and help c cohesive systems.
- **Supporting Technologies:**
  - **Akka (Scala)** – A toolkit for building highly concurrent, d applications, allowing strong cohesion between related
  - **Kotlin (Coroutines)** – Kotlin's coroutine feature allows v concurrent applications in an elegant way.

6. Bounded Contexts (DDD):

- **Frameworks:**
  - **Axon Framework (Java)** – Besides supporting CQRS and encourages the implementation of bounded contexts, p domains.
  - **Microsoft Orleans (C#)** – This virtual actor framework h defining isolated bounded contexts.
- **Supporting Technologies:**
  - **Microservice Architecture** – Often, bounded contexts a microservices, where each service is responsible for a s
  - **Kubernetes** – Used for deploying bounded context mic and independence between services.

7. Event-Driven Architecture:

- **Frameworks:**
  - **Apache Kafka** – A distributed messaging platform that event-driven architectures, processing millions of event
  - **NATS** – A lightweight messaging system for event-drive designed to work in multi-cloud environments.
- **Supporting Technologies:**

cala (with Akka), **Haskell**,
create highly modular and

distributed, and fault-tolerant
d services.
you to build cohesive,

d Event Sourcing, Axon
providing a way to modularize

helps build domain services by

are implemented in the form of
specific domain or context.
roservices, ensuring scalability

can serve as a backbone for
ts in real-time.
en, distributed architectures,

- **RabbitMQ** – Message broker software that implements pub/sub or task queues.
- **AWS SNS/SQS** – Managed services for message broker applications, allowing you to send, store, and receive m systems.

8. Microservices Architecture (to support Bounded Contexts a

- **Frameworks:**
  - **Spring Boot with Spring Cloud (Java)** – Supports micros integrated features for service discovery, load balancin
  - **Micronaut (Java)** – A JVM-based framework that suppo applications and integrates well with CQRS and event s
- **Supporting Technologies:**
  - **Docker** – Containerization helps isolate microservices a easy deployment and scaling.
  - **Kubernetes** – Orchestrates microservices deployed in c tolerance and scalability.

s event-driven patterns like

ing in event-driven
nessages across distributed

nd CQRS):

services architecture with
g, and resilience.
orts microservice and serverless
ourcing.

nd their bounded contexts for

containers, ensuring fault