ScaleMePlease
Microservice Design Patterns

Michel Barret
Pierrick Rassat
Better, Faster, Stronger

SERVICES
Better, Faster, Stronger

Reliability  Availability  Scalability
Data       API        Storage
SERVICES
Protocols  Behaviour
Resilience  Evoluability
SMP – MMORPG

Create a Marketplace

HTTPS Only
From Monolith to microservices

• More and more users
  • More requests, more data to process and store

• Improve reliability, availability…
  • Single point of failure, cascading errors
Breaking up into Microservices

• Single Responsibility Principle

• Reduce tight coupling between services

• Breaking up into functions or model domains
First Service
First Service
Community market stock exchange rate

Record a new sale
Get item exchange rate

SMP Stock Market

Service Responsability
Service API

...
First Service
Make the service reliable

[Diagram showing a circuit breaker and bulkhead]

[Logo: Netflix and Hystrix]

Defend Your App
First Service
Avoid single point of failure
First Service

Firewall

Cluster
First Service
Load blancer

Load Balancer

Database 1

Database 2
First Service
Gateway offloading

Load Balancer

HTTPS

HTTP

HTTP/HTTPS

Databases
First Service

Single point of failure

Diagram showing two Load Balancers, each connecting to a database.
First Service
Current state of our architecture

[Diagram showing a network architecture with Load Balancers and DNS]
Cloud services make it magic!
Event Based System Design
Mismatch Between Read and Write Representations

SMP Stock Market
SMP Stock Market

Mismatch Between Read and Write Representations

Record A NEW SALE

Get item EXCHANGE RATE

Get
Mismatch Between Read and Write Representations

SMP Stock Market

Record
A NEW SALE

Get item
EXCHANGE RATE
CQRS - Command and Query Responsibility Segregation

SMP Stock Market
CQRS - Command and Query Responsibility Segregation

SMP Stock Market WRITE

SMP Stock Market READ

WRITE

READ
CQRS - Command and Query Responsibility Segregation

SMP Stock Market WRITE

SMP Stock Market READ

WRITE

? READ

Event Sourcing to the Rescue

SMP Stock Market
WRITE

Event Store

Event Handler OR Projection Engine

SMP Stock Market
READ

Publish event
Cloud services make it magic! (again)

An example with Azure shiny icons
Migrate Legacy Service
Strangler Facade

Let's start!

SMP Stock Market

SMP Stock Market CQRS Facade
Strangler Facade

Let's start!

Strangler Facade

SMP Stock Market

SMP Stock Market

CQRS Facade
Strangler Facade

SMP Stock Market

CQRS Facade

Next step
Monolith to Microservices

When new services are highly dependent on legacy...
Anti-Corruption Layer

When new services are highly dependent on legacy... And you want to protect them
Improve Client-side API
Gateway Aggregation

SMP Stock Market

New Sale (exchange rate)

New Sale (player trade)

SMP Marketplace
SMP Market
General Purpose API
Backend For Frontend

SMP Market
Mobile BFF

SMP Market
Desktop BFF
Compared to the single data model used in CRUD-based systems, the use of separate query and update models for the data in CQRS-based systems simplifies design and implementation. However, one disadvantage is that unlike CRUD designs, CQRS code can’t automatically be generated using scaffold mechanisms.

The query model for reading data and the update model for writing data can access the same physical store, perhaps by using SQL views or by generating projections on the fly. However, it’s common to separate the data into different physical stores to maximize performance, scalability, and security, as shown in the next figure.

The read store can be a read-only replica of the write store, or the read and write stores can have a different structure altogether. Using multiple read-only replicas of the read store can greatly increase query performance and application UI responsiveness, especially in distributed scenarios where read-only replicas are located close to the application instances. Some database systems, such as databases like PostgreSQL or some RDBMs, include additional features such as Hot Standby replicas, to
Transform a legacy application with Kubernetes & Istio (by David Gageot, Google)

Compared to the single data model used in CRUD-based systems, the use of separate query and update models for the data in CQRS-based systems simplifies design and implementation. However, one disadvantage is that unlike CRUD designs, CQRS code can’t automatically be generated using scaffold mechanisms.

The query model for reading data and the update model for writing data can be perhaps by using SQL views or by generating projections on the fly. However, different physical stores to maximize performance, scalability, and security.

The read store can be a read-only replica of the write store, or the read and structure altogether. Using multiple read-only replicas of the read store can application UI responsiveness, especially in distributed scenarios where read application instances. Some database systems, like RDBMS, don’t include additional
Code Quality Measurement: WTFs/Minute

Good Code

Bad Code

WTF is this shit?
Anti-Corruption Layer
Gateway Offloading
Aggregation
API Gateway
Load Balancer
Circuit Breaker
Strangler Facade
QRS
Event Sourcing
Sidecar
Bulk Request
Backend For Frontend
Q/A: How to implement business transactions? “Distributed Sagas” pattern to the rescue

https://www.youtube.com/watch?v=0UTOLRTwOX0