Best Practices for Working with Data in a Microservices Architecture

Emmanuel Bernard  
Chief Architect, Data

Kim Palko  
Product Manager, Data

Madou Coulibaly  
Solutions Architect, Data

Red Hat
MSA has many benefits

- Freedom to independently develop and deploy services
- Better fault isolation
- Code for different services can be written in different languages
- Continuous integration and continuous delivery
- Easy to understand and modify
- Organized around business capabilities
- Easy to scale and re-use
- Work well with containers
But what about the data?

Martin Fowler: Decentralized Data Management where each microservice encapsulates its own data

Problems with a data store per microservice:
- Not enough ROI for breaking up existing databases and data warehouses
- Copying data can lead to inconsistency
- Security issues around access control
- Difficult to keep a consistent view of data across microservices
From Brownfield Database
to Greenfield Microservice

Emmanuel Bernard
Monolith to microservices
From app velocity
Monolith to microservices
To data velocity
Big bang approach

What about the brownfield app?
One step at a time

- Monolith app
- New registration app (RW)
- User reservations management (RW)

- Virtual DB isolation layer (subset view)
- Virtual DB isolation layer (subset view)

- Monolith RDBMS

JBoss Data Virtualization
Benefits of data virtualization

Data remains centralized
- Monolith still works

Microservice only seeing the subset it is supposed to
- Clear boundaries; Avoid dependency abuse
- Read / write

Step by step evolution preparing for the future
- Virtually choking the monolith before the coup de grace
- Data lineage

JBoss Data Virtualization
Data security
Controlling who uses which data set

More microservices, more demands on your data sources
  ● Who uses what?

Regulatory constraints

Many aspects of securitization including
  ● Restrict access to a subset of the data
  ● Anonymize data
  ● Restrict who has access to specific data
  ● Audit and know who has see what
Approach
A reusable control piece: data firewall

[Diagram showing a monolith app connected to Monolith RDBMS, Super user for call center connected to Virtual DB Adding RBAC, and Audit log connected to Elasticsearch.]
Benefits

Common technology between apps and databases
A la carte restriction capabilities
- Controlled by a different team
Transparent to the microservice app development
Reusable solution across all microservices
Demo part 1
Hotel Reservations and Check-in

Kim Palko
Original Hotel Booking Application

Three-Tier Architecture

Single Application, Single Database
- Booking reservations
- Customer rewards
- Hotel room inventory
Problem: New functionality requirements with tight deadlines

- Teams need to work independently and in-parallel
- Architecture needs to be open to post-relational technology
- GDPR:
  - Need to restrict access to a subset of data
  - Anonymize data (no PII)
  - Audit who has had access to what data

- Mobile check-in
- Keyless entry
- GDPR privacy regulations
Demo Architecture: Functional

- **Monolith app**
  - Reservation Service
    - Virtual Reservation DB
  - Customer Service
    - Virtual Customer DB
  - Inventory Service
    - Virtual Inventory DB
- **Mobile App**
- **Web App**
- **SQL Client**
  - Booking State Service
    - Shared State IMDB
  - Virtual Reporting DB
- **Monolith RDBMS**
Demo part 1

Madou Coulibaly
Solution: Move to microservices architecture
Break up the monolithic database virtually
Demonstrated benefits

Data remains centralized

- Original application continues to work
- Faster time to production vs physically breaking up database
- Data can be migrated over time if necessary
- Can easily augment centralized database with new data sources
- Developers and Operations get along with each other

Security

- Restrict access to subset of the data (by design or by roles)
- Anonymize the data
Sharing state in a stateless app world

Emmanuel Bernard
The challenge of state

Microservices need to scale out (up and down): very elastic

- Scaling state in the app?
- State scaling or compute scaling?

Deploying new version (A/B or canary) with no disruption

- State?

Which state

- Basket, last articles seen, HTTP session etc
Approach
There is a service for that
Benefits

Low latency
Complexity outside the app
Cross data center replication

Load balancer

DC1

Application
Serves customers A-M

Shared state

DC2

Application
Serves customers N-Z

Shared state

Replication

Customers A-Z

Red Hat Data Grid
One service to fail them all

Cluster of microservices with dependencies

- Latency accumulation

What if one goes down
Approach

There is a service for that

Application 1
Application 1 v2
Application 2

Cache
A-F
Cache
G-L
Cache
M-R
Cache
S-Z
Benefits

Externalize infrastructure
  Simpler
  Better hit/miss ratio
Differentiated lifecycle and scaling
A/B or canary testing without performance drop
Surviving the temporary loss of one microservice
  Common requests
Maturing your data microservices approach
CQRS, event sourcing and more
Microservices
Data islands

Full isolation between microservices islands
- Hard to achieve

Adding new services off the same data stream

Scaling different parts independently
- Give flexibility to change (data) tech
- CQRS
Change data capture to the rescue

- Monolith app
- Monolith RDBMS
- Real time DB change (CDC)
- Source Monetary Tx event filter
- Spending dashboard builder
- Spending dashboard aggregate DB
- User <-> device DB
- Debezium
- Source Monetary Tx event filter
- Monetary Tx push notification app
- Spending dashboard builder
- User <-> device DB
Benefits
Of Change Data Capture
Of CQRS

CDC decorrelates the existing database from later consumption by new systems
Lower the load on the database
No update cost on exist apps consuming the database
Opens up “real time” doors
Demo part 2
Hotel Reservations and Check-in

Kim Palko
Problem: Need to share the booking state

Need to share booking state while customer is searching for a room

- Select one room (state that needs to be shared across room inventory)
- Then go to the Customer Service to do credit card checking
- then the Reservation Service to create the reservation

Assure that all the microservices are stateless

Every step across the application will be stored in a Data Grid and saved

The state can be shared across Web and Mobile UI’s
Problem: Need to test new functionality with limited availability before rolling out globally

A/B Testing

- Highlight rooms with a living area
  - Change button color and add an icon
- 2 different versions of the application:
  - one showing the new screen and one without
  - Part of customers have the new screen
- After one week trial, make a decision which version to keep in production
Demo part 2
Hotel Reservations and Check-in

Madou Coulibaly
Solution: Share state with an in-memory data grid
Benefits demonstrated

Shared State
- Share booking state across microservices and different UI’s
- Reliability of the shared data (distributed)
- Low latency
- Scalability (out and down)

A/B Testing
- Deploying a new version of the application re-using the currently used state
- Upgrading an app with no down time
Demo Architecture: Projects
Summary

- When moving a monolithic application to a microservices architecture
  - take a pragmatic approach and break up large data sources logically
  - option to move data physically over time
- Architect for security up front
- Delegate data handling to specialized services (i.e. out of the app)
  - Don’t try to implement caching, shared memory, data virtualization etc.
- Red Hat can help you manage data in a MSA, starting where you are today
- Take your microservices evolution as a journey
References

- Re-create this demo yourself
  - https://github.com/mcouliba/hotel-booking

- Free download book:
  - Migrating to Microservice Databases: From Relational Monolith to Distributed Data
    by Edson Yanaga, Red Hat (with Forward by Emmanuel Bernard)

  by Christian Posta, Red Hat
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos

facebook.com/redhatinc
twitter.com/RedHat