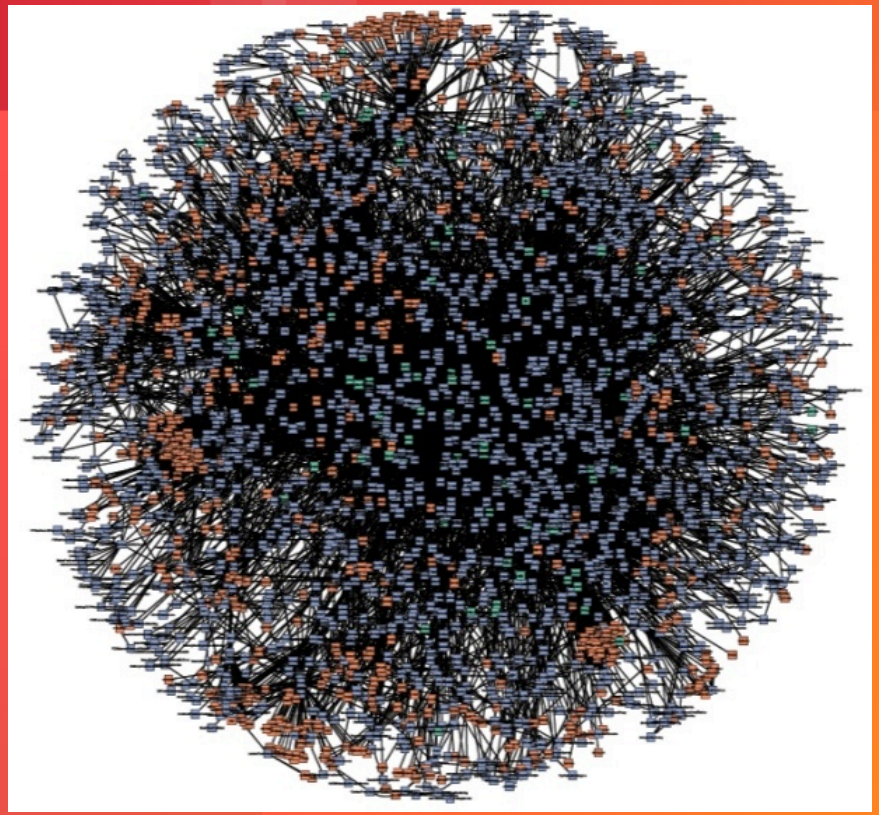
Data design and modeling for microservices

Ian Robinson | July 2018
ianrob@amazon.com

What to Expect from the Session

- Microservices at Amazon
  - Overview and Challenges
  - Key Elements and Benefits
  - Two Pizza Teams
- Data Architecture Challenges
  - Transactions and Rollbacks
  - Streams
  - Master Data Management
- Choosing a Data Store
- Aggregation

Microservices at Amazon



Service-Oriented Architecture (SOA)

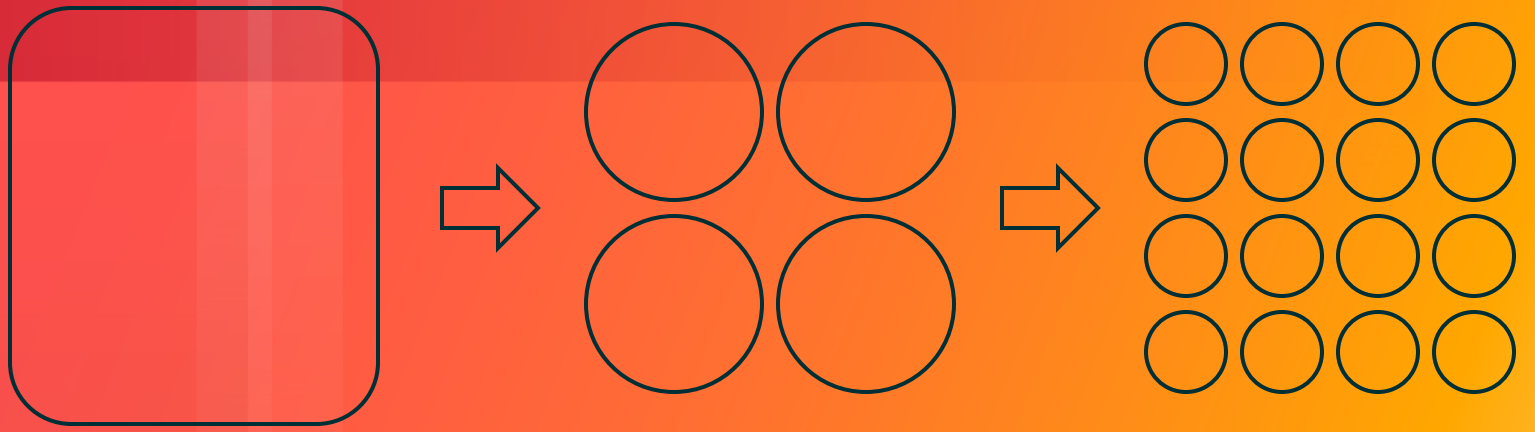
Single-purpose

Connect only through APIs

Connect over HTTPS

“Microservices”

Monolithic vs. SOA vs. Microservices



Monolithic
Single Unit

SOA
Coarse-grained

Microservices
Fine-grained

Monolithic vs. SOA vs. Microservices

Microservices:

- Many very small components
- Business logic lives inside of single service domain
- Simple wire protocols(HTTP with XML/JSON)
- API driven with SDKs/Clients

SOA:

- Fewer more sophisticated components
- Business logic can live across domains
- Enterprise Service Bus like layers between services
- Middleware

Microservice Challenges

Distributed computing is hard

Transactions

- Multiple Databases across multiple services

Eventual Consistency

Lots of moving parts

Service discovery

Increase coordination

Increase message routing

Key Elements of Microservices…

Some core concepts are common to all services

- Service registration, discovery, wiring, administration
- State management
- Service metadata
- Service versioning
- Caching

Low Friction Deployment

Automated Management and Monitoring

Key Elements of Microservices…

Eliminates any long-term commitment to a technology stack

Polyglot ecosystem

Polyglot persistence

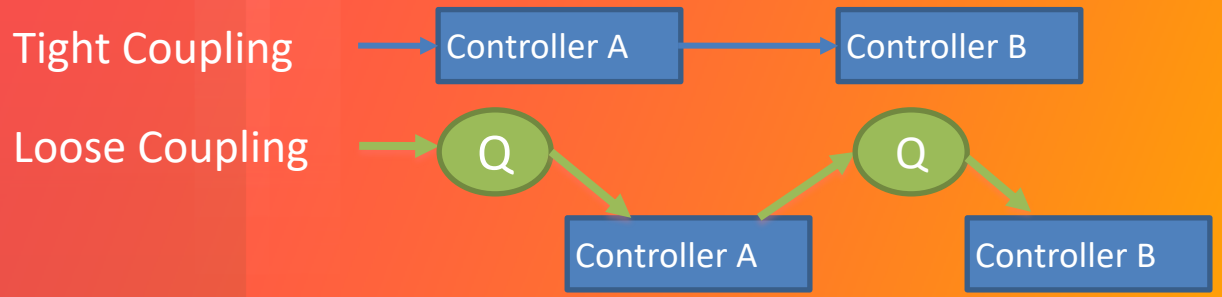
- Decompose Databases
- Database per microservice pattern

Allows easy use of Canary and Blue-Green deployments

Key Elements of Microservices…

Each microservice is:

- Elastic: scales up or down independently of other services
- Resilient: services provide fault isolation boundaries
- Composable: uniform APIs for each service
- Minimal: highly cohesive set of entities
- Complete: loosely coupled with other services



Microservices Benefits

- Fast to develop
- Rapid deployment
- Parallel development & deployment
- Closely integrated with DevOps
  - Now "DevSecOps"
- Improved scalability, availability & fault tolerance
- More closely aligned to business domain

Principles of the Two Pizza Team

Two-pizza teams

Full ownership

Full accountability

Aligned incentives

“DevOps”

How do Two Pizza Teams work?

We call them “Service teams”

Own the “primitives” they build:

- Product planning (roadmap)
- Development work
- Operational/Client support work

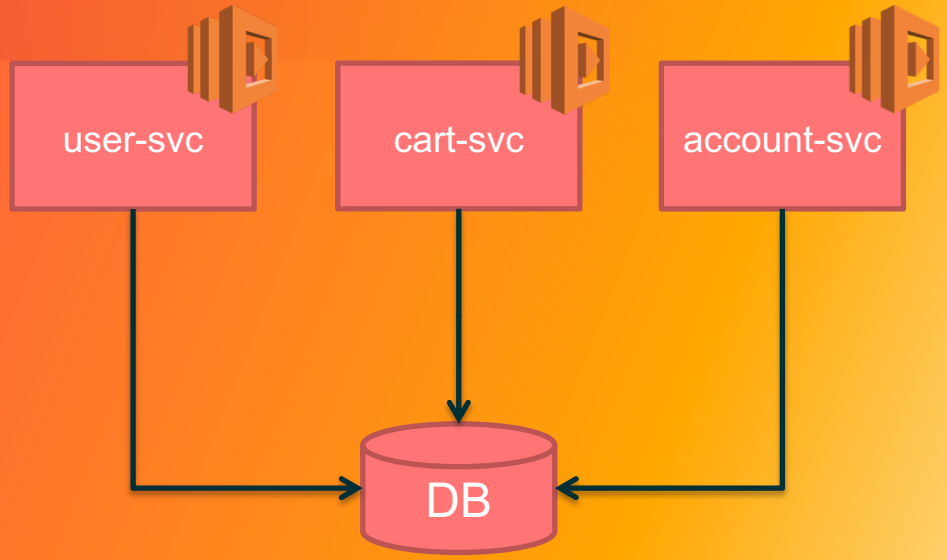
“You build it, you run it”

Part of a larger concentrated org (Amazon.com, AWS, Prime, etc)

Data Architecture Challenges

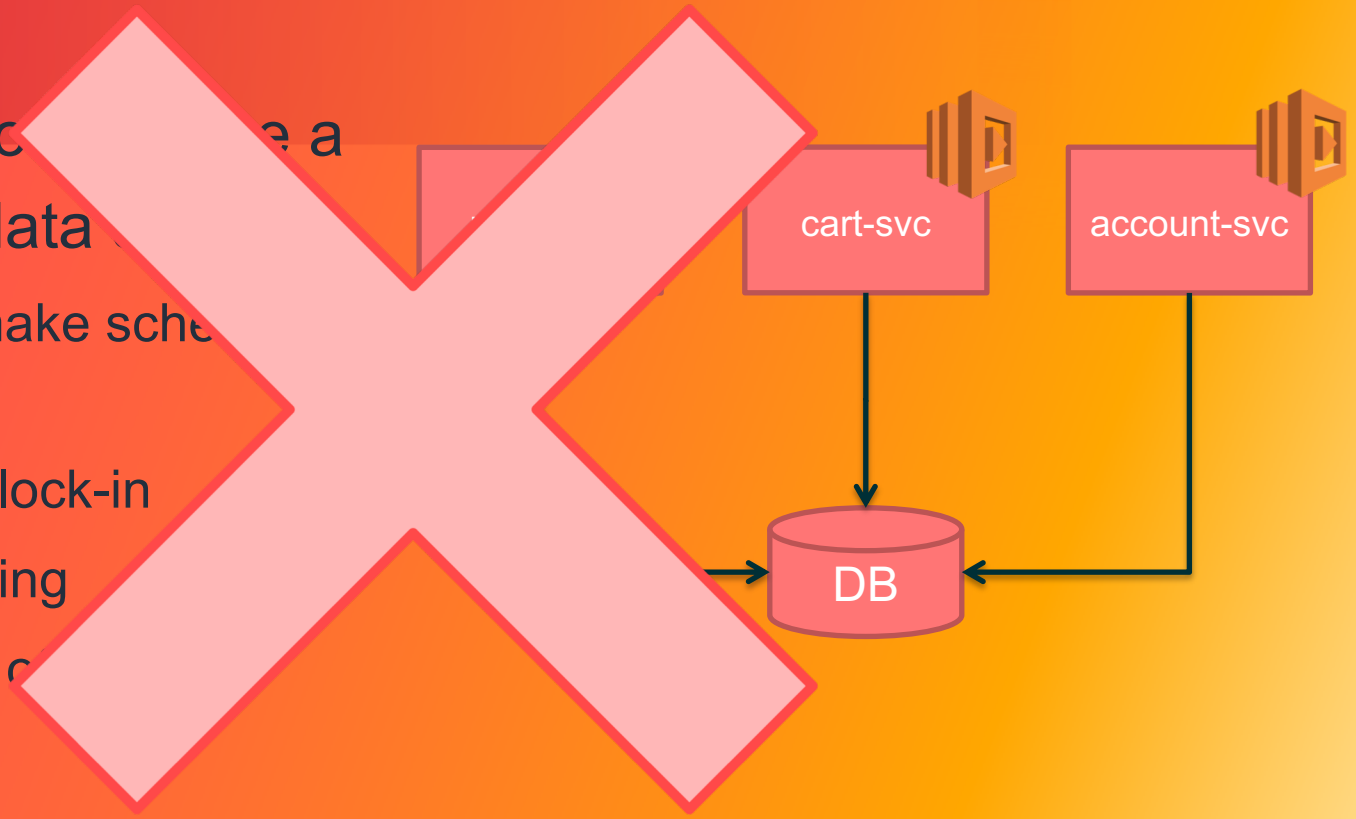
Challenge: Centralized Database

- Applications often have a **monolithic** data store
  - Difficult to make schema changes
  - Technology lock-in
  - Vertical scaling
  - Single point of failure



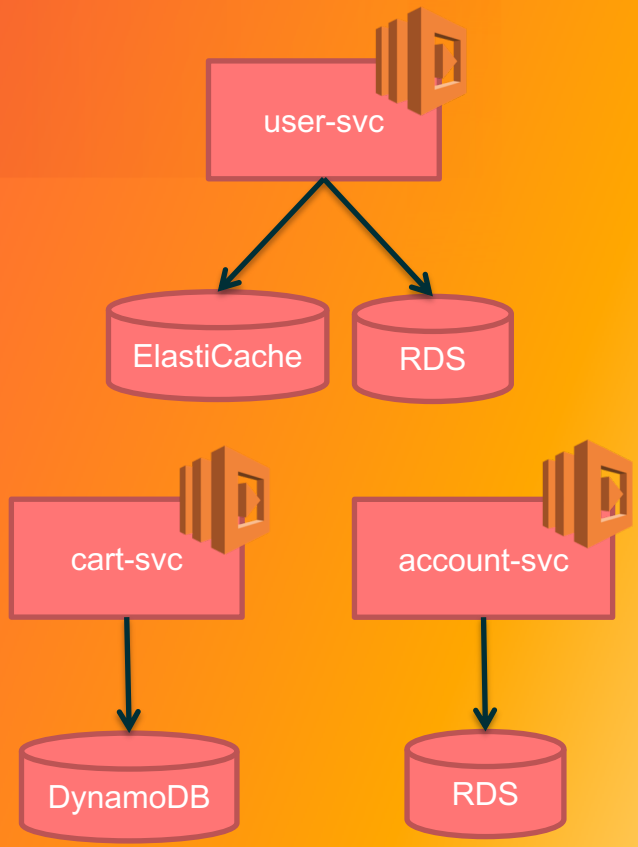
Centralized Database – Anti-pattern

- Applications c e a **monolithic** data
  - Difficult to make sche changes
  - Technology lock-in
  - Vertical scaling
  - Single point c



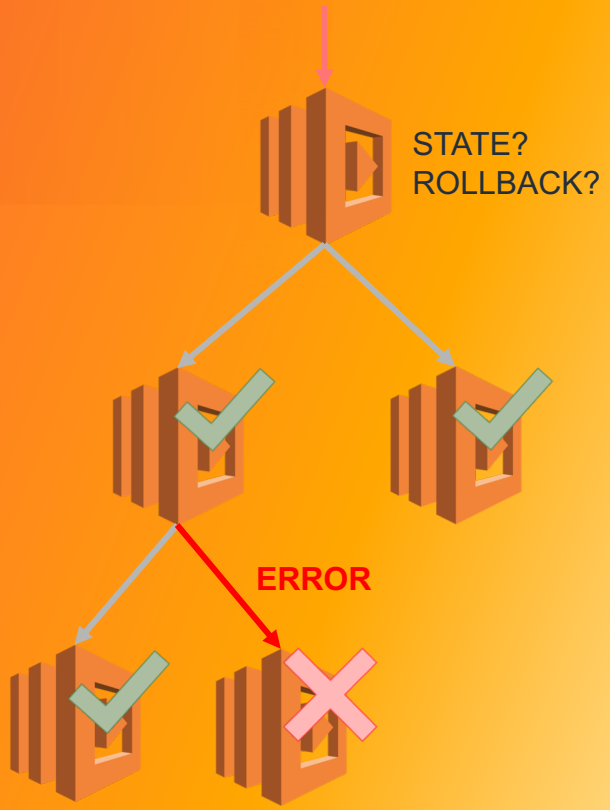
Decentralized Data Stores

- **Polyglot** Persistence
- Each service chooses it's data store technology
- Low impact schema changes
- Independent scalability
- Data is gated through the service API

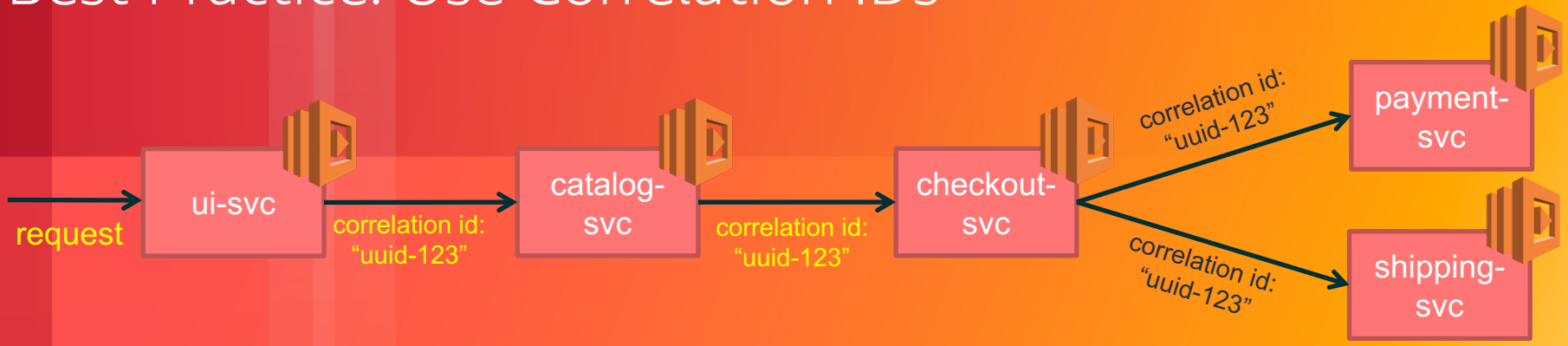


Challenge: Transactional Integrity

- Polyglot persistence generally translates into **eventual consistency**
- **Asynchronous calls** allow non-blocking, but returns need to be handled properly
- How about **transactional integrity**?
  - Event-sourcing – Capture changes as sequence of events
  - Staged commit
  - Rollback on failure



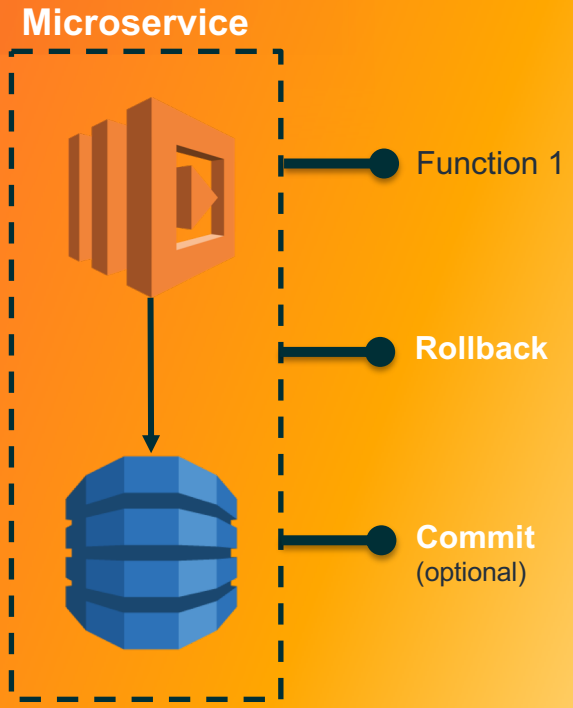
Best Practice: Use Correlation IDs




```
09-02-2015 15:03:24 ui-svc INFO [uuid-123] ……
09-02-2015 15:03:25 catalog-svc INFO [uuid-123] ……
09-02-2015 15:03:26 checkout-svc ERROR [uuid-123] ……
09-02-2015 15:03:27 payment-svc INFO [uuid-123] ……
09-02-2015 15:03:27 shipping-svc INFO [uuid-123] ……
```

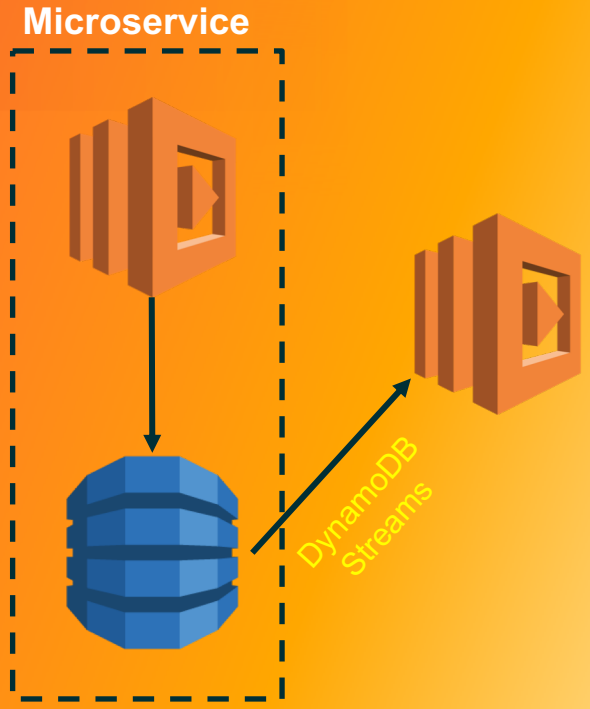
Best Practice: Microservice owns Rollback

- Every microservice should expose it's own "rollback" method
- This method could just **rollback** changes, or trigger **subsequent actions**
  - Could send a notification
- If you implement **staged commit**, also expose a commit function



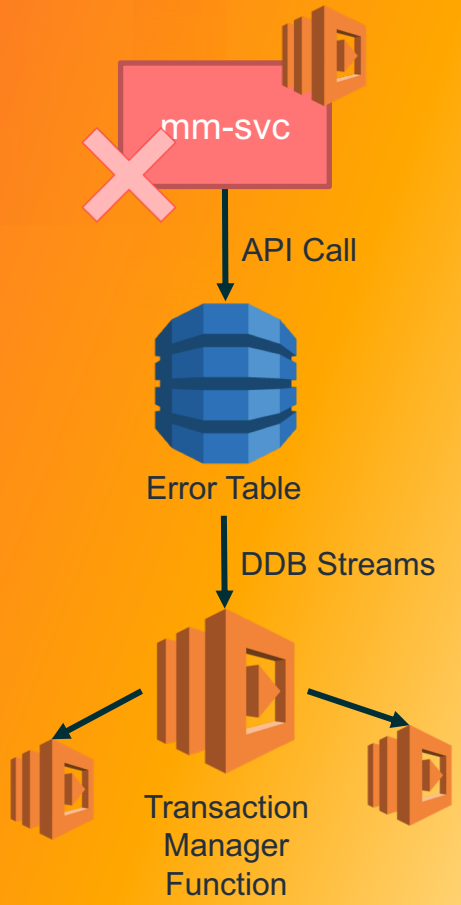
Event-Driven: DynamoDB Streams

- If async, consider event-driven approach with **DynamoDB Streams**
- Don’t need to manage function execution failure, DDB Streams **automatically retries** until successful
- **“Attach”** yourself to the data of interest

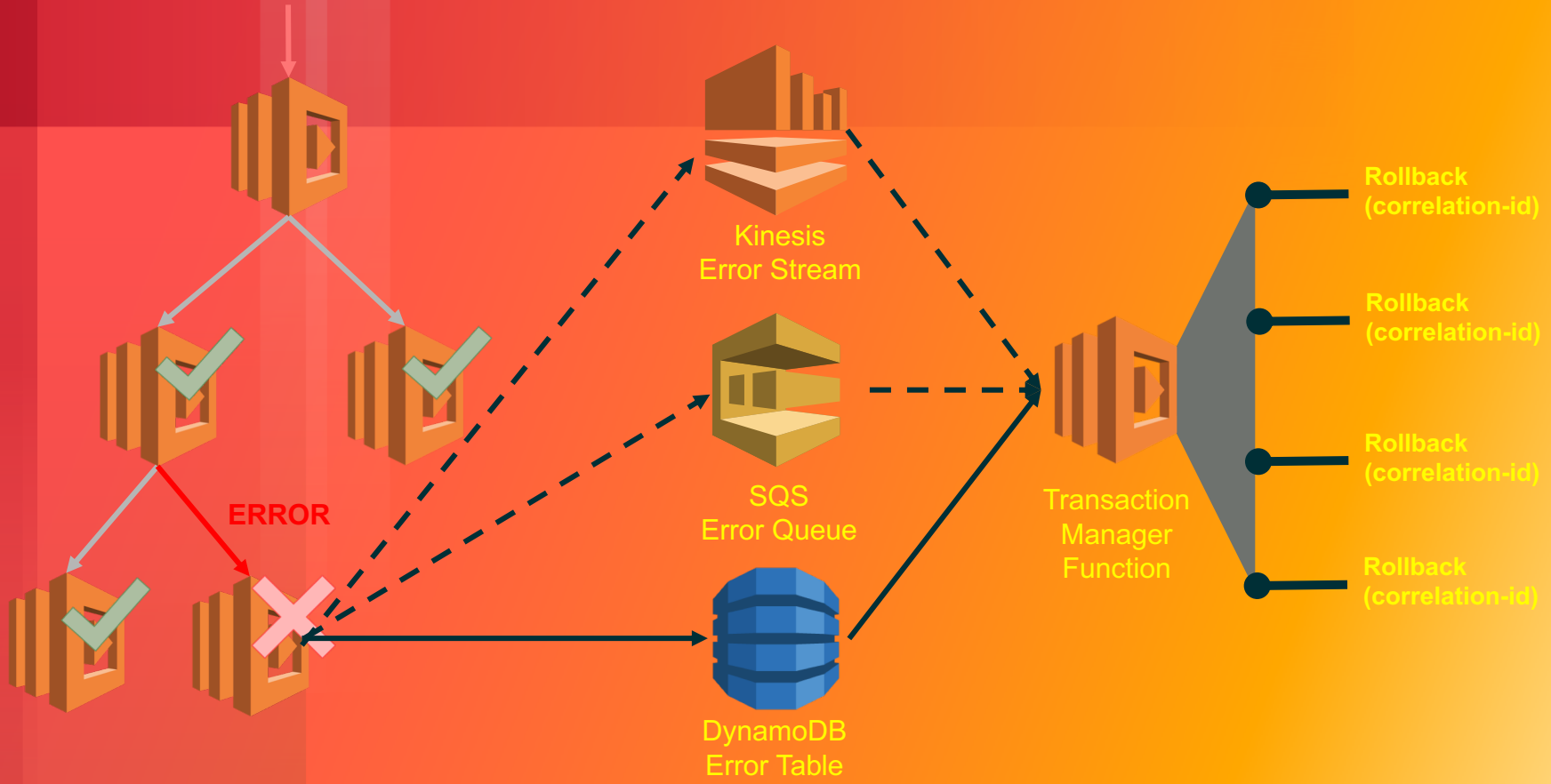


Challenge: Report Errors / Rollback

- What if functions fail? (business logic failure, not code failure)
- Create a **"Transaction Manager"** microservice that notifies all relevant microservices to rollback or take action
- DynamoDB is the **trigger** for the clean-up function (could be SQS, Kinesis etc.)
- Use **Correlation ID** to identify relations

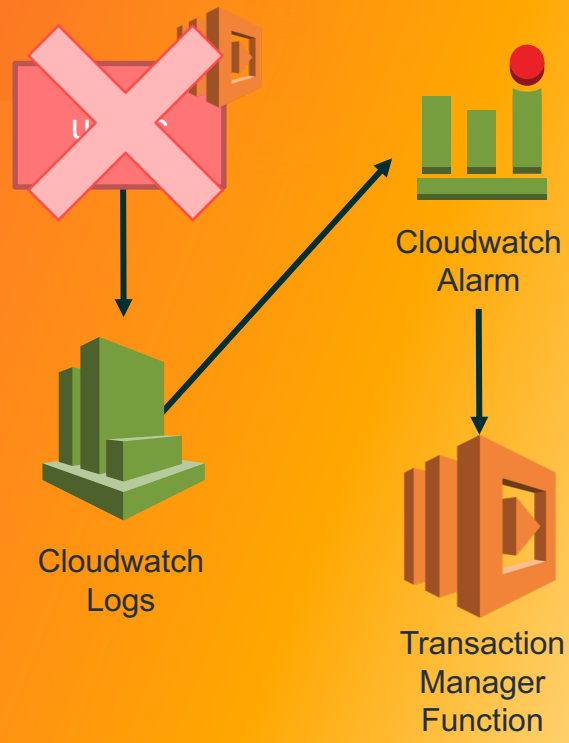


Challenge: Report Errors / Rollback



Challenge: Code Error

- Lambda Execution Error because of **faulty code**
- Leverage **Cloudwatch Logs** to process error message and call Transaction Manager
- Set **Cloudwatch Logs Metric Filter** to look for Error/Exception and call Lambda Handler upon Alarm state

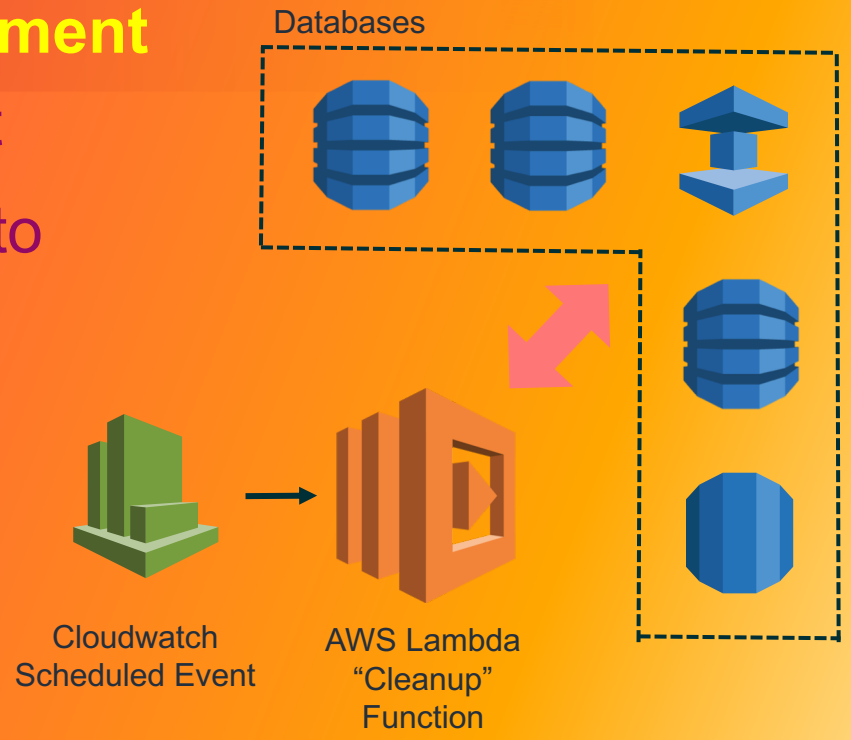


Beware: Stream Model with AWS Lambda

- DynamoDB Streams and Kinesis streams directly work with AWS Lambda. However AWS Lambda needs to **acknowledge processing** the message correctly
- If Lambda fails to process the message, the stream horizon will not be moved forward, creating a “jam”
- **Solution**: Monitor AWS Lambda **Error Cloudwatch Metric** and react when error rate of same “Correlation ID” keeps increasing

MDM – Keep Data Consistent

- Perform **Master Data Management** (MDM) to keep data consistent
- Create AWS Lambda function to **check consistencies** across microservices and “cleanup”
- Create a **Cloudwatch Event** to schedule the function (e.g. hourly basis)



Choosing a Datastore

Storage & DB options in AWS

In-Memory

Amazon ElastiCache

NoSQL

Amazon DynamoDB

SQL

Amazon RDS

Amazon Redshift

Object

Amazon S3

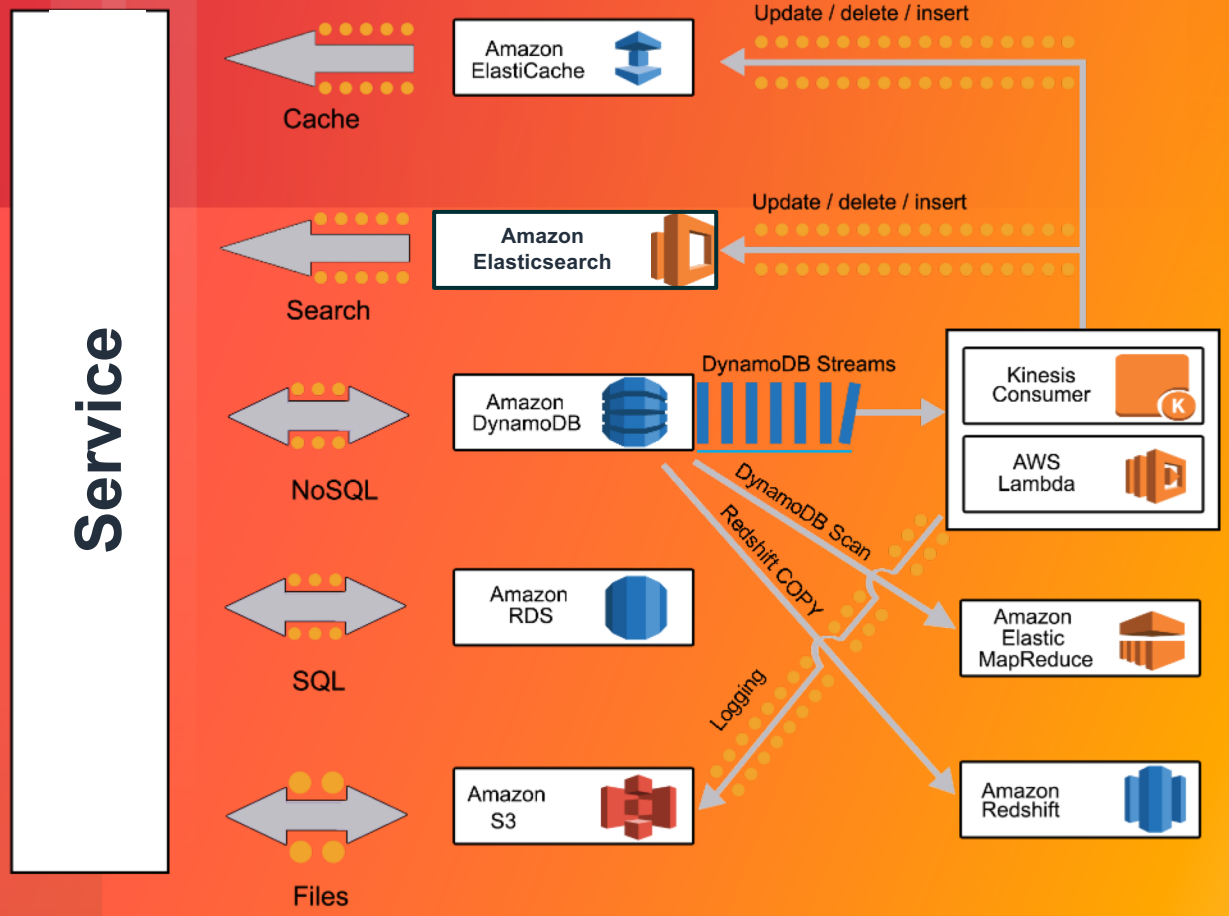
Amazon Glacier

Search

Amazon Elasticsearch Service

Streaming

Amazon Kinesis



Challenge: What Service to Use?

- Many problems can be solved with NoSQL, RDBMS or even in-memory cache technologies
- Non-functional requirements can help identify appropriate services
- **Solution**: Classify your organizations non-functional requirements and map them to service capabilities

Determine Your Non-Functional Requirements

This is only an example. Your company's classifications will be different

Requirement				
Latency	> 1s	200 ms -1s	20 ms – 200 ms	< 20 ms
Durability	99.99	99.999	99.9999	> 99.9999
Storage Scale	< 256 GB	256 GB – 1 TB	1 TB – 16 TB	> 16 TB
Availability	99	99.9	99.95	> 99.95
Data Class	Public	Important	Secret	Top Secret
Recoverability	12 – 24 hours	1 – 12 hours	5 mins – 1 hour	< 5 mins
Skills	None	Average	Good	Expert

There will be other requirements such as regulatory compliance too.

Map Non-Functional Requirements to Services

The information below is not exact and does not represent SLAs

Service	Latency	Durability	Storage	Availability	Recoverability from AZ Failure (RPO, RTO)
RDS	< 100 ms	> 99.8 (EBS)	16 TB	99.95	0s and 90s (MAZ)
Aurora	< 100 ms	> 99.9	64 TB	> 99.95	0s and < 30s (MAZ)
Aurora + ElastiCache	< 1 ms	> 99.9	64 TB	> 99.95	0s and < 30s (MAZ)
DynamoDB	< 10 ms	> 99.9	No Limit	> 99.99	0s and 0s
DynamoDB / DAX	< 1 ms	> 99.9	No Limit	> 99.99	0s and 0s
ElastiCache Redis	< 1 ms	N/A	3.5 TiB	99.95	0s and < 30s (MAZ)
Elasticsearch	< 200 ms	> 99.9	150 TB	99.95	0s and < 30s (Zone Aware)
S3	< 500 ms	99.999999999	No Limit	99.99	0s and 0s

Finalizing Your Data Store Choices

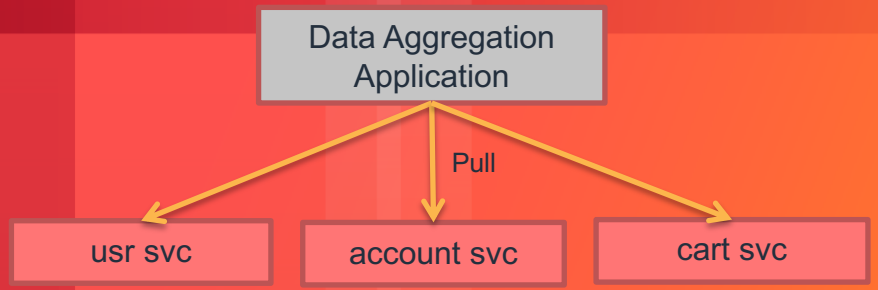
- After mapping your non-functional requirements to services you should have a short list to choose from
- Functional requirements such as geospatial data and query support will refine the list further
- You may institute standards to make data store selection simpler and also make it easier for people to move between teams, e.g Redis over Memcached and PostgreSQL over MySQL. These can still be overridden, but require justification to senior management

Challenge: Reporting and Analytics

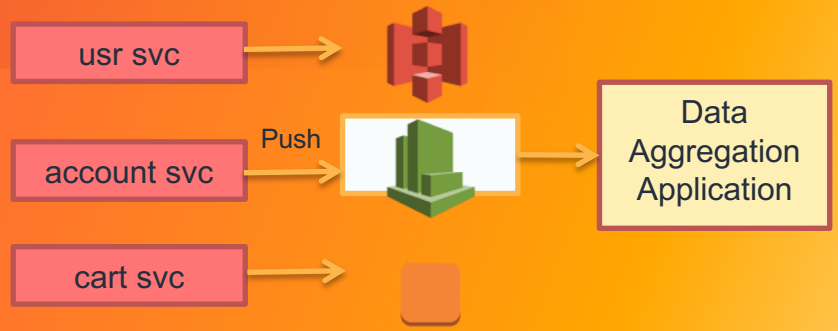
- Data is now spread across a number of isolated polyglot data stores
- Consolidation and aggregation required
- **Solution**: Pull data from required microservices, push data to data aggregation service, use pub/sub. Don’t use a composite service (anti-pattern).

Aggregation

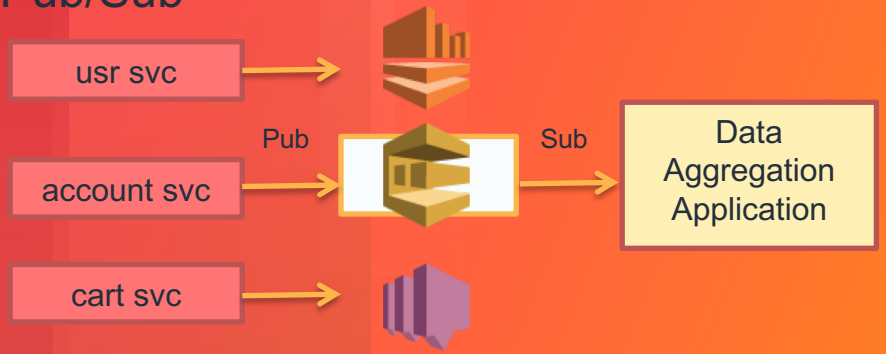
Pull model



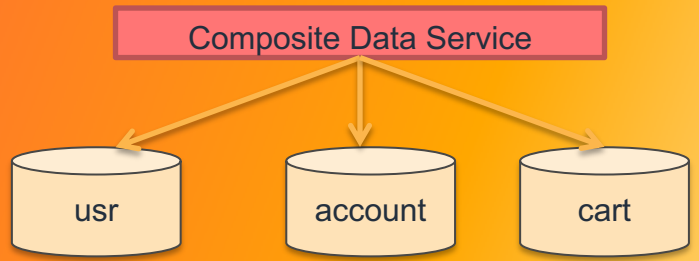
Push model



Pub/Sub



Composite



A Few Thoughts

- Use **Non-Functional Requirements** to help identify the right data store(s) for each microservice
- Use **polyglot persistence** to avoid bottlenecks, schema issues and allow independent scalability (and cache)
- Embrace **eventual consistency** and design fault-tolerant business processes which can recover
- Think ahead and plan your **analytics requirements** as part of the overall architecture

Build your cloud skills with AWS

Use the discount code when booking online and get 20 % off:
MKBERSUM18-1-19P6OP0XN6D3Y

Upcoming Trainings:

Date	Training	Location	Duration	Language
August 28-30	Developing on AWS	Berlin	3 days	English
September 5	Running Container-Enabled Microservices on AWS	Berlin	1 day	English
September 6	Deep Learning on AWS	Berlin	1 day	English
September 18	Building a Serverless Data Lake on AWS	Berlin	1 day	English
October 09-11	Systems Operations on AWS	Berlin	3 days	English
October 23-25	DevOps Engineering on AWS	Berlin	3 days	English

Terms & Conditions: Please note that this discount only applies to Public Schedule Classes sold by AWS in Germany. The promotion expires on 30.09.2018

Thank you!