Serverless Architectural Patterns and Best Practices

Sascha Möllering, Solutions Architect

October 2018

About me

Sascha Möllering
Solutions Architect
Amazon Web Services EMEA SARL

- 16 years of dev, software architecture, and systems architecture background
- Has written a lot of Java code.
- Enjoys containers and serverless. All day.

Twitter: @sascha242
Email: smoell@amazon.de

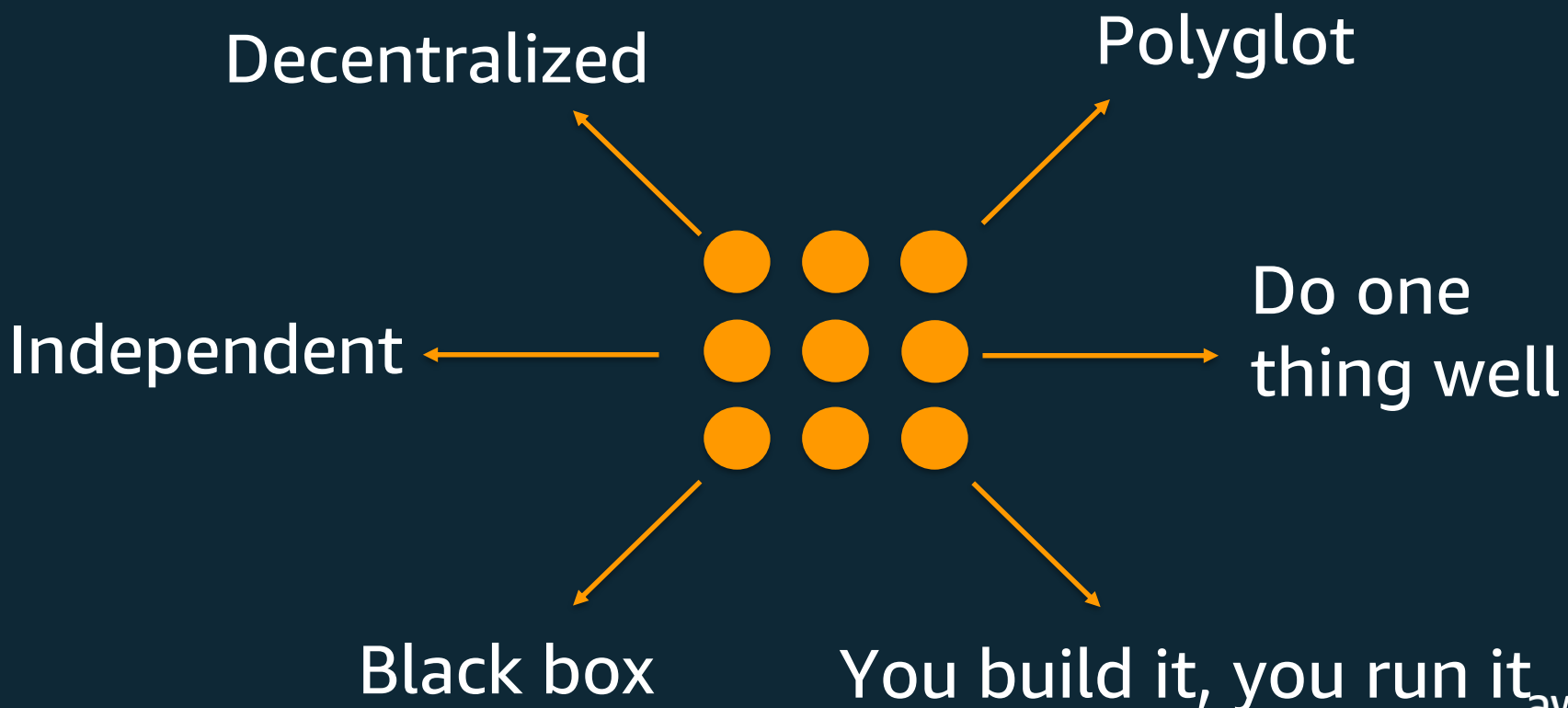
aws

Agenda

- Microservices
- Serverless Foundations
- Web application
- Data Lake
- Stream processing

Microservices

Characteristics of Microservice Architectures



Microservices should be stateless.

Keep state in external systems.

No shared libraries or shared SDKs.

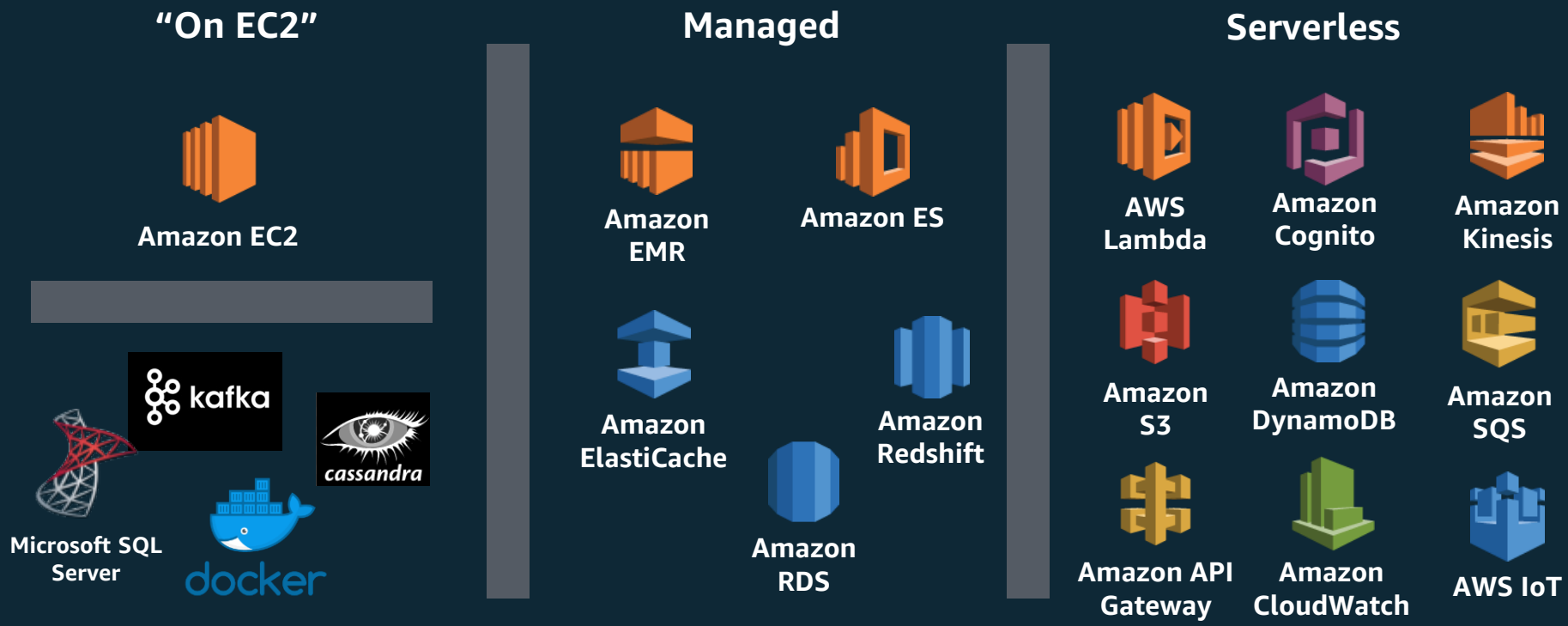
Avoid Host-Affinity.

Use mechanisms for registration.

Use lightweight protocols for communication.

Serverless Foundations

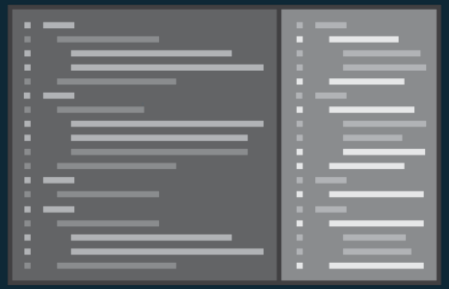
Spectrum of AWS offerings



aws

Serverless means…

- No servers to provision or manage
- Scales with usage
- Never pay for idle
- Built-in High-Availability and Disaster Recovery



Serverless applications

EVENT SOURCE

FUNCTION

SERVICES (ANYTHING)

Changes in data state

Requests to endpoints

Changes in resource state

Node.js
Python
Java
C#
Go

aws

Lambda considerations and best practices

AWS Lambda is stateless—architect accordingly

- Assume no affinity with underlying compute infrastructure
- Local filesystem access and child process may not extend beyond the lifetime of the Lambda request

aws

Lambda considerations and best practices

Can your Lambda functions survive the cold?

- Instantiate AWS clients and database clients outside the scope of the handler to take advantage of container re-use.
- Schedule with CloudWatch Events for warmth
- ENIs for VPC support are attached during cold start

Executes during cold start

```
import sys
import logging
import rds_config
import pymysql

rds_host = "rds-instance"
db_name = rds_config.db_name
try:
        conn = pymysql.connect(
except:
        logger.error("ERROR:
def handler(event, context):
        with conn.cursor() as cur:
```

Executes with each invocation

aws

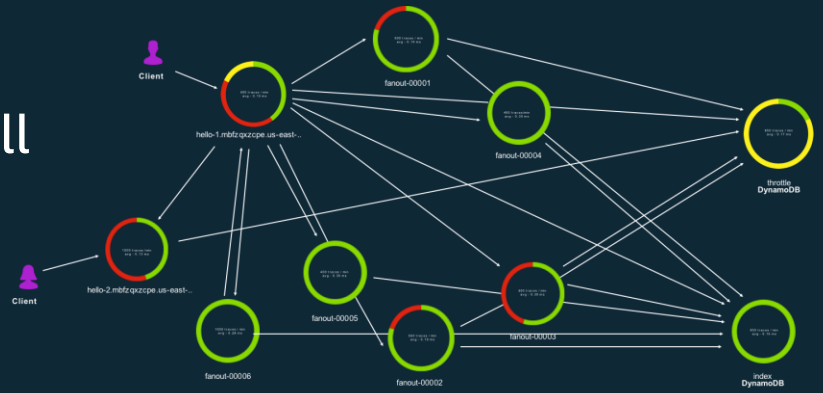
Lambda Best Practices

- Minimize package size to necessities
- Separate the Lambda handler from core logic
- Use Environment Variables to modify operational behavior
- Self-contain dependencies in your function package
- Leverage “Max Memory Used” to right-size your functions
- Delete large unused functions (75GB limit)

aws

AWS X-Ray Integration with Serverless

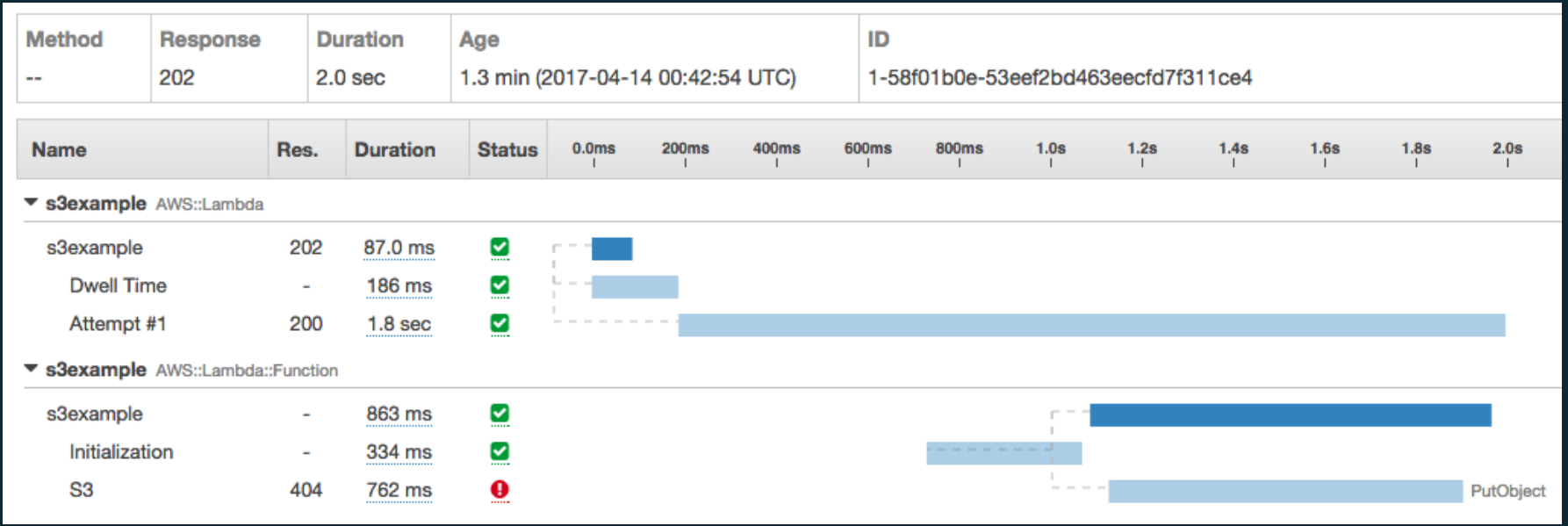
- Lambda instruments incoming requests for all supported languages
- Lambda runs the X-Ray daemon on all languages with an SDK




```
var AWSXRay = require('aws-xray-sdk-core');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
S3Client = AWS.S3();
```


aws

AWS X-Ray Trace Example



Meet SAM!

aws

AWS Serverless Application Model (SAM)

- CloudFormation extension optimized for serverless
- New serverless resource types: functions, APIs, and tables
- Supports anything CloudFormation supports
- Open specification (Apache 2.0)

https://github.com/awslabs/serverless-application-model

aws

SAM Local

- Develop and test Lambda locally
- Invoke functions with mock serverless events
- Local template validation
- Local API Gateway with hot reloading

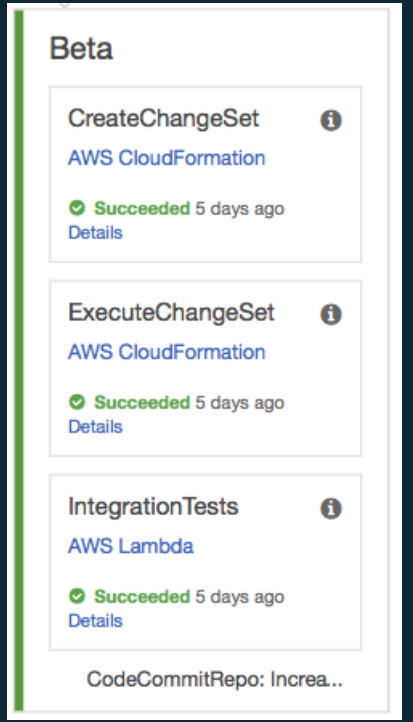
https://github.com/awslabs/aws-sam-local

aws

Delivery via CodePipeline

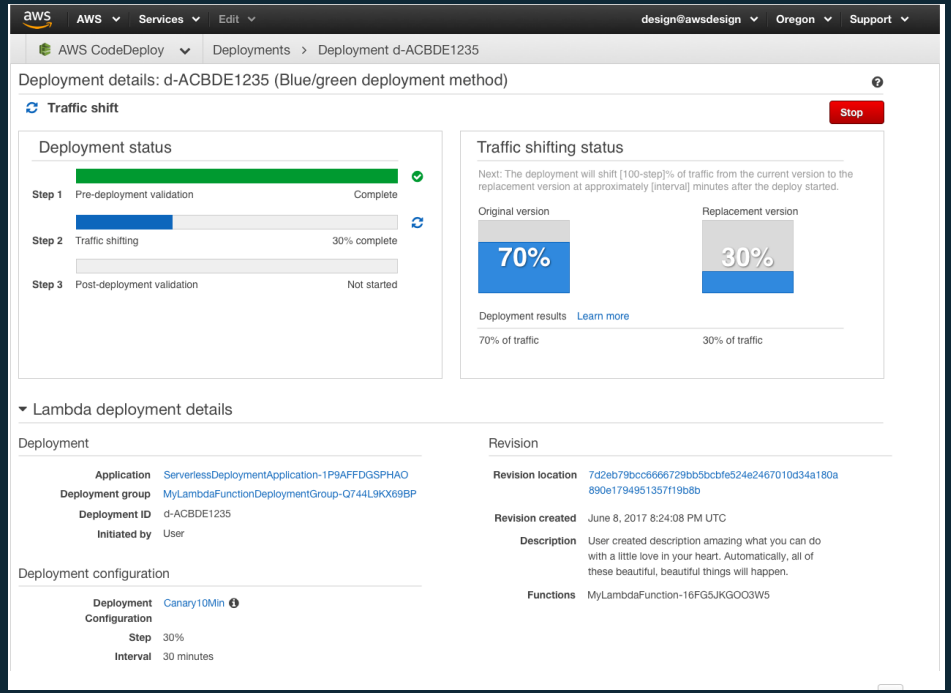
Pipeline flow:

1. Commit code to source code repository
2. Package/test in CodeBuild
3. CloudFormation actions in CodePipeline to create or update stacks via SAM templates
   - Optional: Make use of ChangeSets
4. Make use of specific stage/environment parameter files to pass in Lambda variables
5. Test our application between stages/environments
   - Optional: Make use of manual approvals



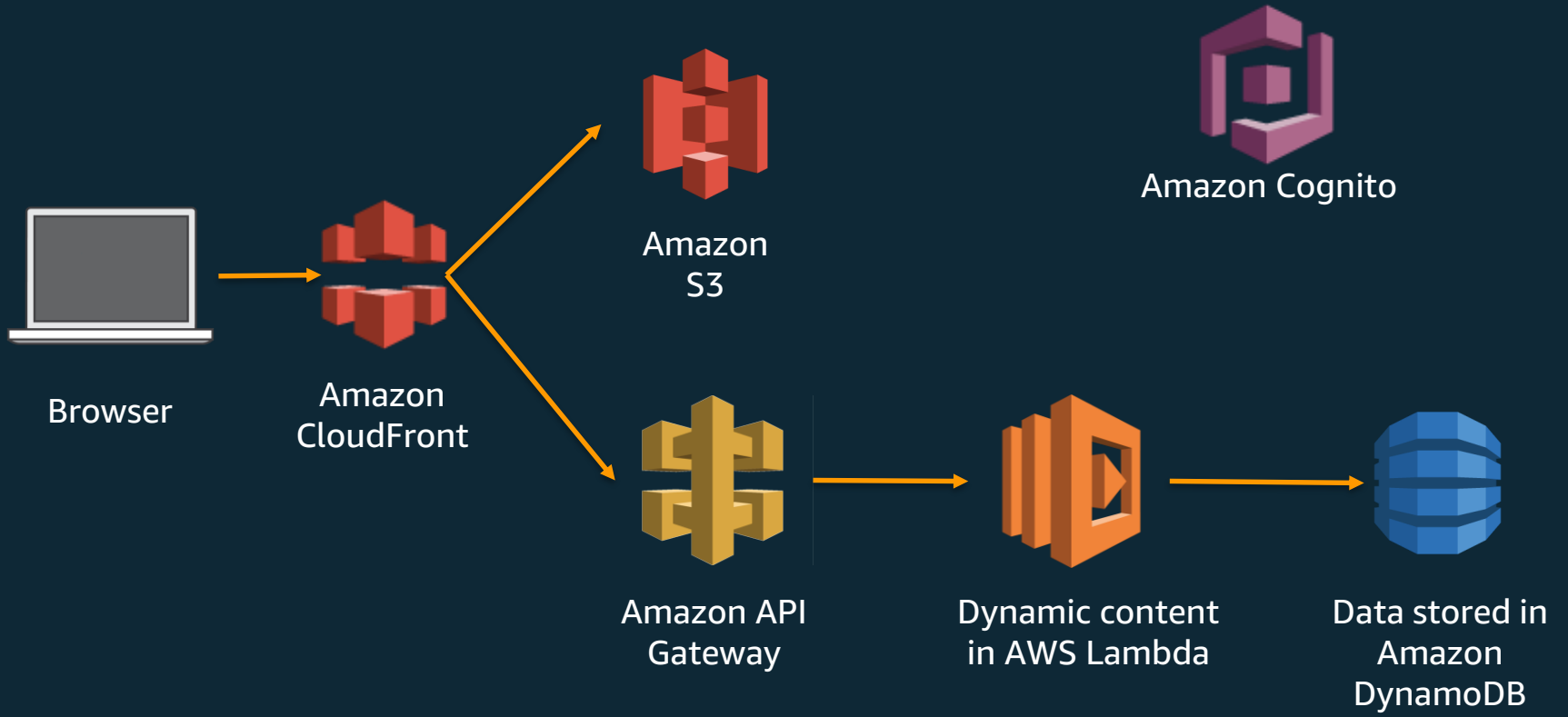
AWS CodeDeploy and Lambda Canary Deployments

- Direct a portion of traffic to a new version
- Monitor stability with CloudWatch
- Initiate rollback if needed
- Incorporate into your SAM templates

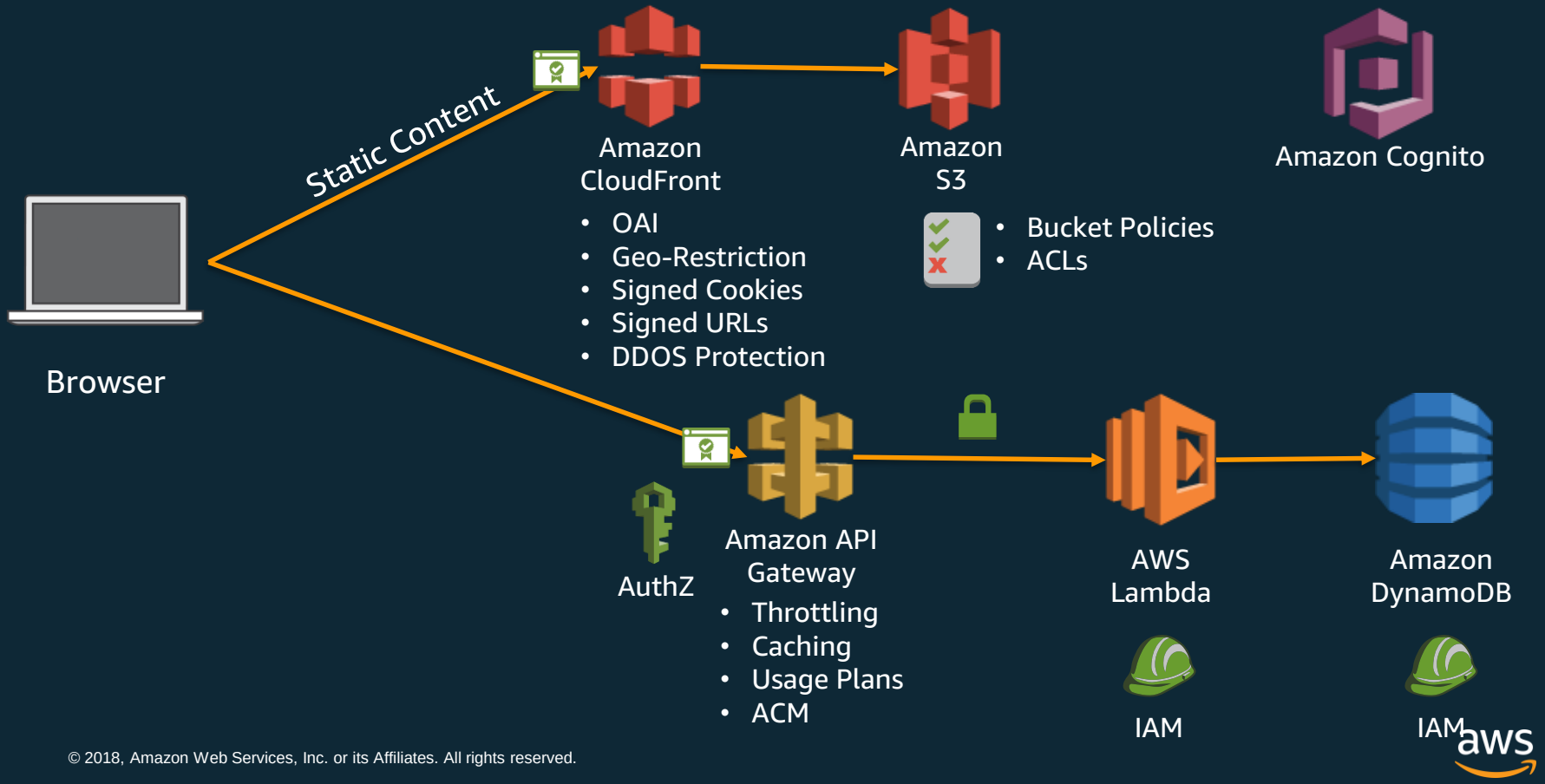


Pattern 1: Web App/Microservice/API

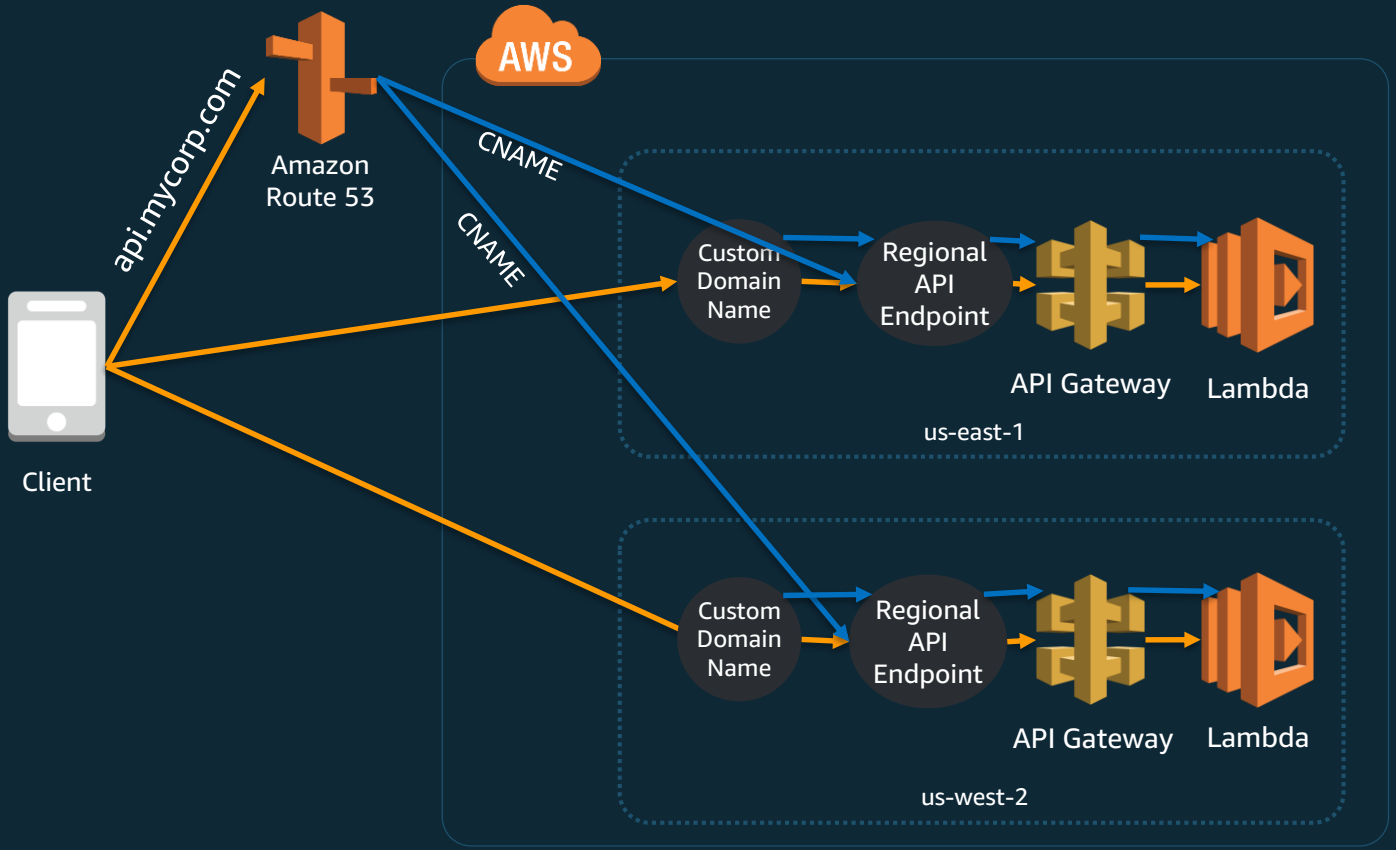
Web application



Serverless web app security



Multi-Region with API Gateway



Useful Frameworks for Serverless Web Apps

- AWS Chalice

  Python Serverless Framework

  https://github.com/aws/chalice

  Familiar decorator-based api similar to Flask/Bottle

  Similar to 3rd Party frameworks, Zappa or Claudia.js

- AWS Serverless Express

  Run Node.js Express apps

  https://github.com/awslabs/aws-serverless-express

- Java - HttpServlet, Spring, Spark and Jersey

  https://github.com/awslabs/aws-serverless-java-container

aws

Pattern 2: Data Lake

Serverless Data Lake Characteristics

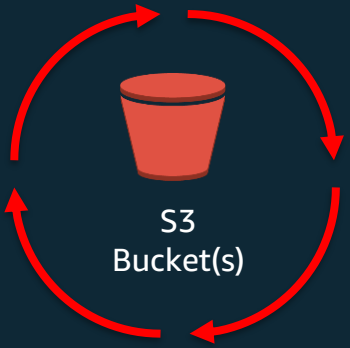
- Collect/Store/Process/Consume and Analyze all organizational data
- Structured/Semi-Structured/Unstructured data
- AI/ML and BI/Analytical use cases
- Fast automated ingestion
- Schema on Read
- Complementary to EDW
- Decoupled Compute and Storage

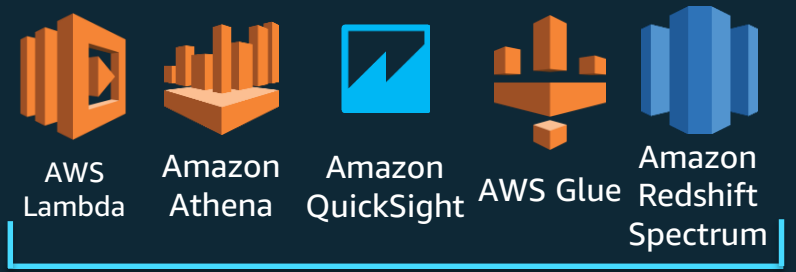
AWS Serverless Data Lake

Catalog & Search

API/UI

Ingest





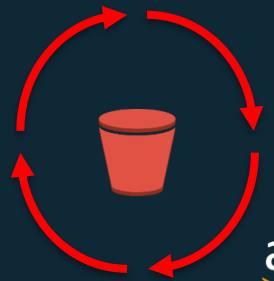
Analytics & Processing

Security & Auditing

aws

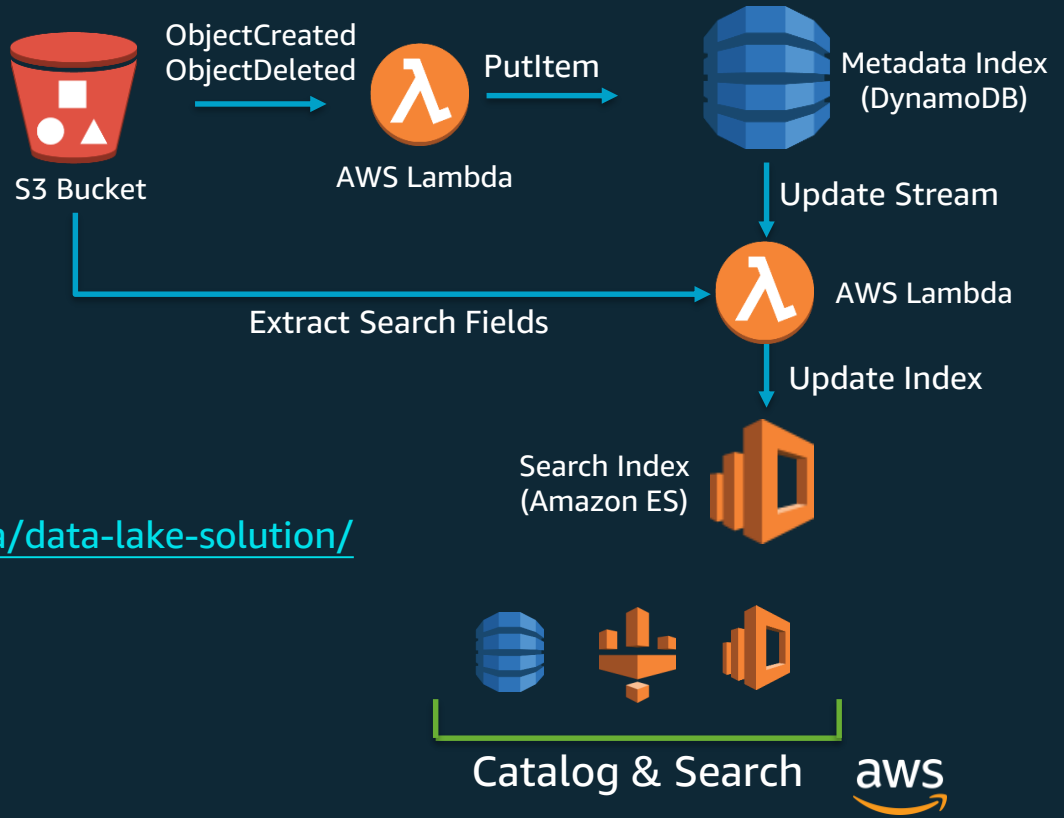
The Foundation…S3

- No need to run compute clusters for storage
- Virtually unlimited number of objects and volume
- Very high bandwidth – no aggregate throughput limit
- Multiple storage classes
- Versioning
- Encryption



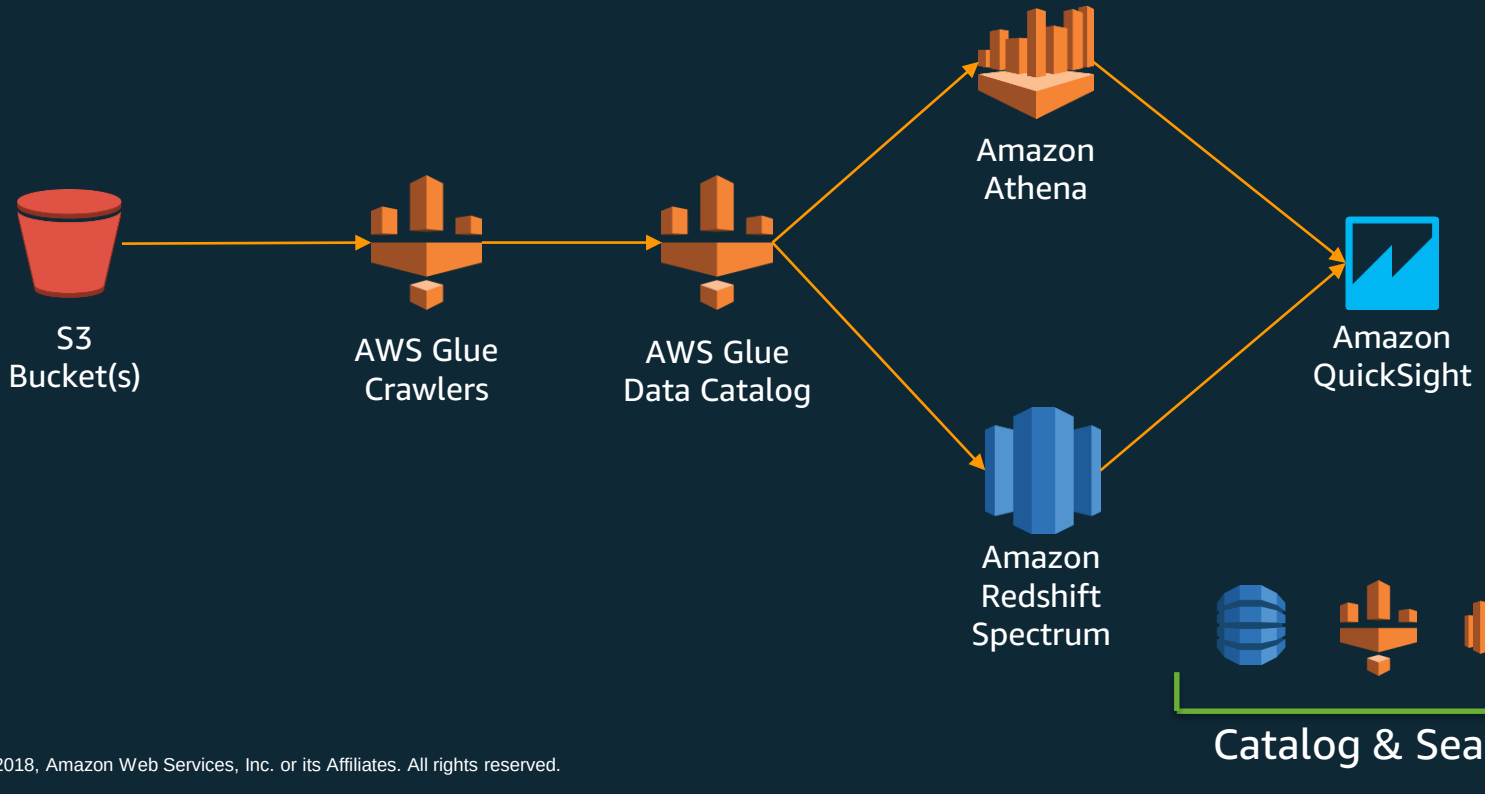
Search and Data Catalog

- DynamoDB as Metadata repository
- Amazon Elasticsearch Service



https://aws.amazon.com/answers/big-data/data-lake-solution/

Instantly query your data lake on Amazon S3



Analytics and Processing

- Amazon QuickSight
- Amazon Athena
- AWS Lambda
- Predictive Analytics
- Amazon EMR
- AWS Glue (ETL)

Athena – Serverless Interactive Query Service

```
SELECT gram, year, sum(count) FROM ngram
WHERE gram = 'just say no'
GROUP BY gram,year ORDER BY year ASC;
```

44.66 seconds…Data scanned: 169.53GB

Cost: $5/TB or $0.005/GB = $0.85

Athena – Best Practices

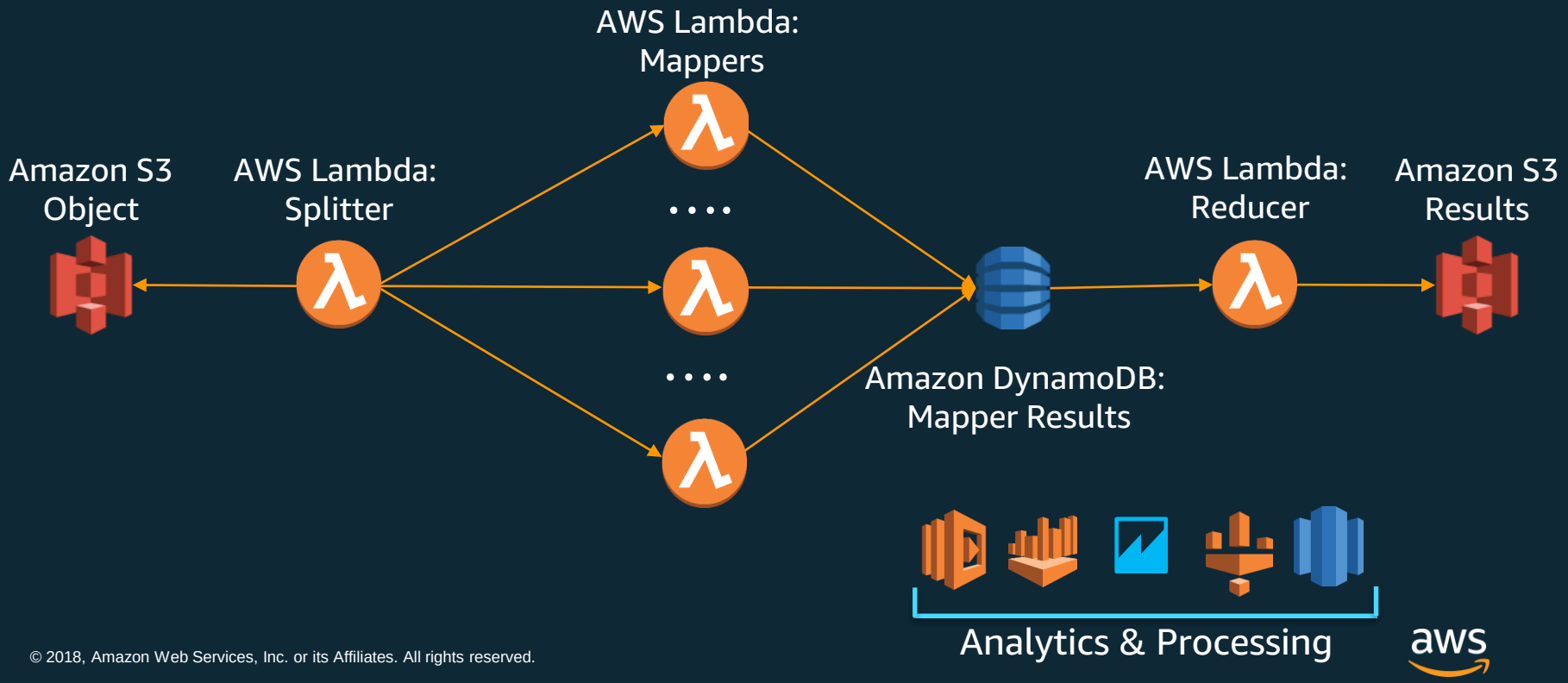
- Partition data

  s3://bucket/flight/parquet/year=1991/month=1/day=2/
- Columnar formats – Apache Parquet, AVRO, ORC
- Optimize file sizes
- Compress files with splittable compression (bzip2)

https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/

aws

Serverless batch processing

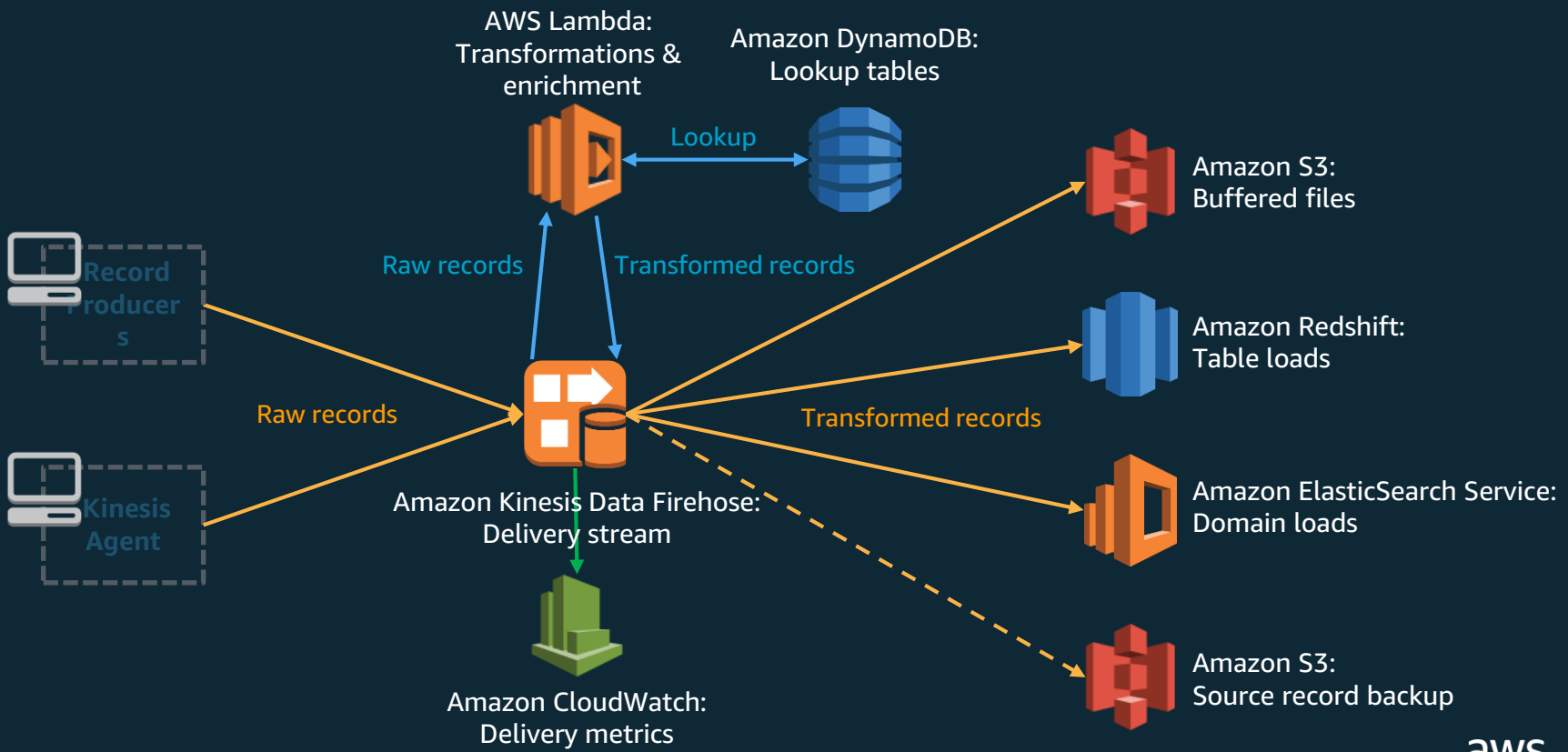


Pattern 3: Stream Processing

Stream processing characteristics

- High ingest rate
- Near real-time processing (low latency from ingest to process)
- Spiky traffic (lots of devices with intermittent network connections)
- Message durability
- Message ordering

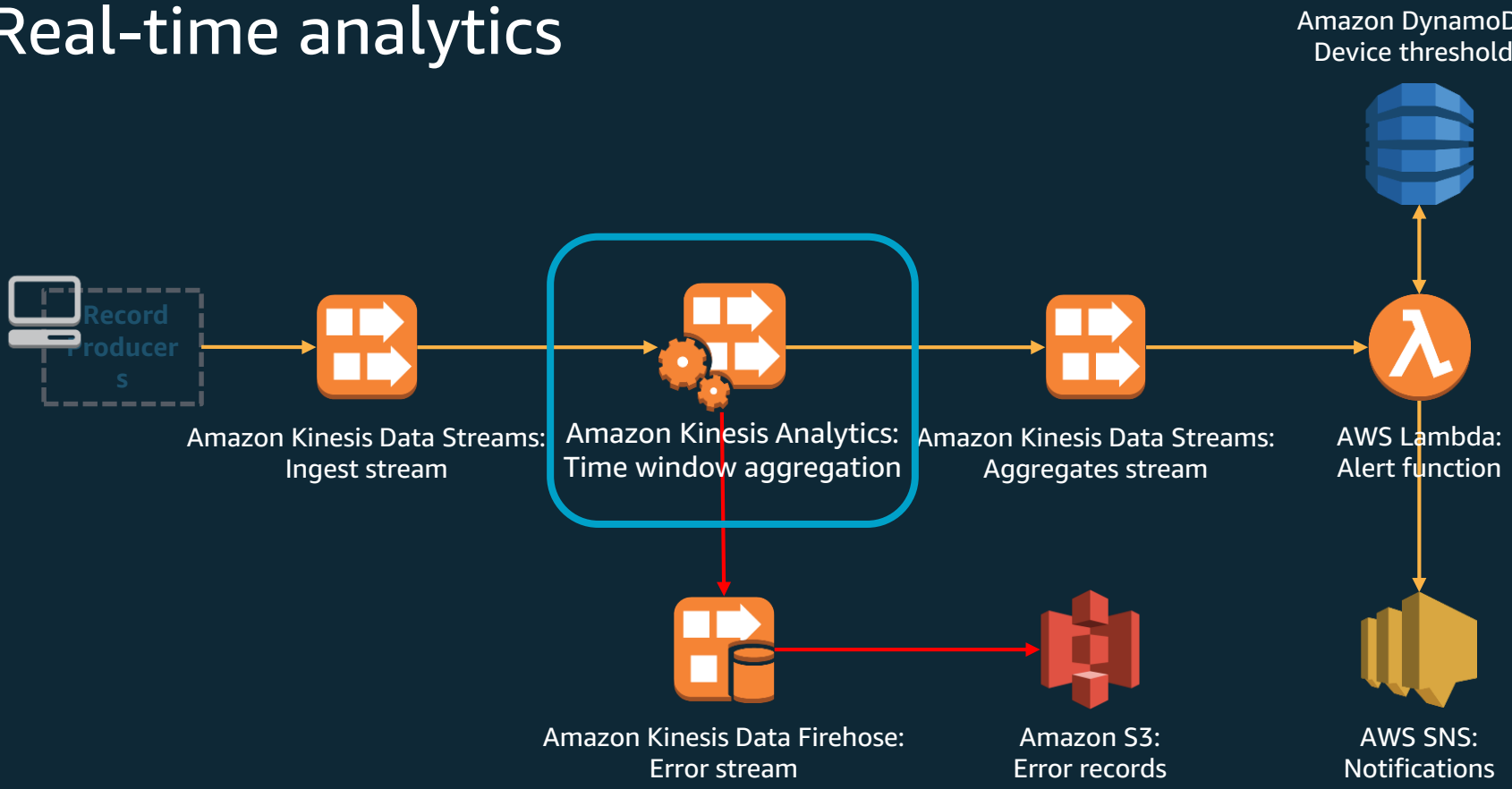
Streaming data ingestion



Best practices

- Tune Firehose buffer size and buffer interval
  - Larger objects = fewer Lambda invocations, fewer S3 PUTs
- Enable compression to reduce storage costs
- Enable Source Record Backup for transformations
  - Recover from transformation errors
- Follow Amazon Redshift Best Practices for Loading Data

Real-time analytics



Amazon Kinesis Analytics

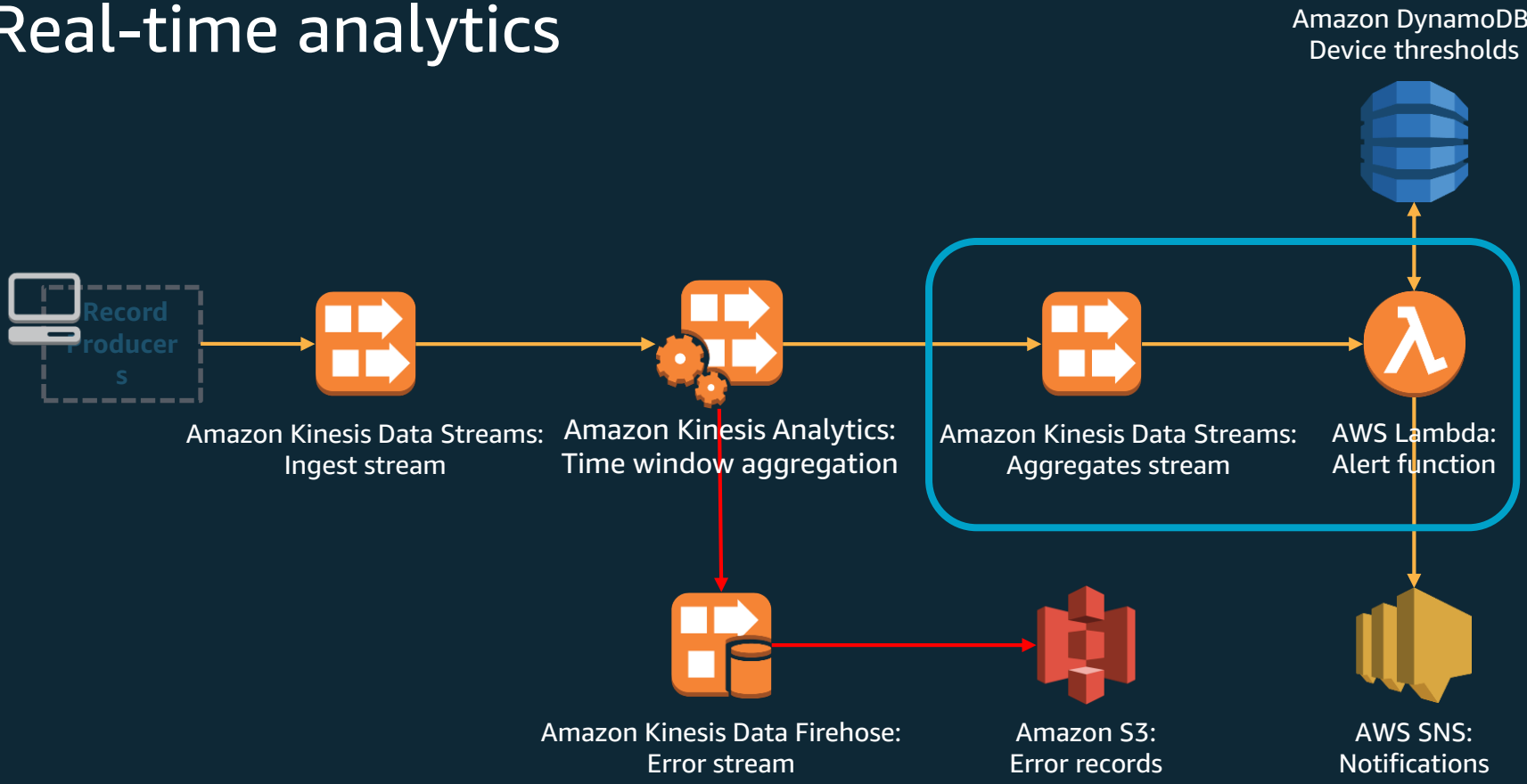

```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE) as "window_ts",
SUM("measurement") as "sample_sum",
COUNT(*) AS "sample_count"
FROM "SOURCE_SQL_STREAM_001"
GROUP BY "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE);
```

Aggregation

1 minute tumbling window

aws

Real-time analytics

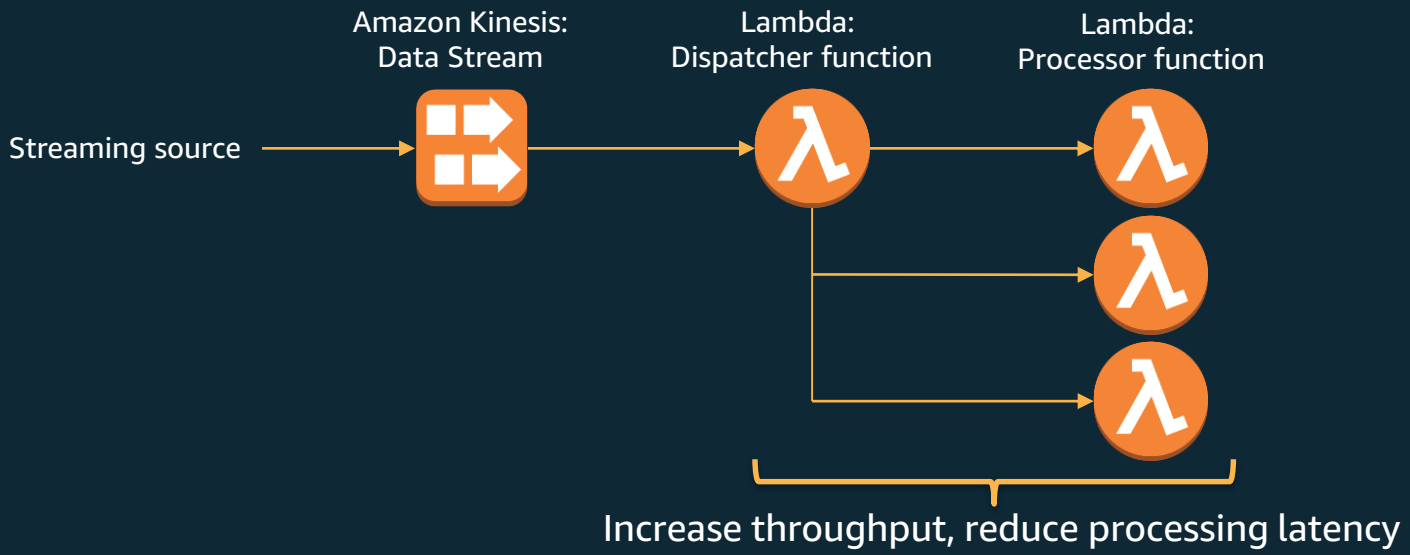


Amazon Kinesis Streams and AWS Lambda

- Number of Amazon Kinesis Streams shards corresponds to concurrent invocations of Lambda function
- Batch size sets maximum number of records per Lambda function invocation

Fan-out pattern

Fan-out pattern trades strict message ordering vs higher throughput & lower latency



Best practices

- Tune batch size when Lambda is triggered by Amazon Kinesis Streams
    - Higher batch size = fewer Lambda invocations
- Tune memory setting for your Lambda function
    - Higher memory = shorter execution time
- Use Kinesis Producer Library (KPL) to batch messages and saturate Amazon Kinesis Stream capacity

aws

Go build something!

Further Reading

Optimizing Enterprise Economics with Serverless Architectures
https://d0.awsstatic.com/whitepapers/optimizing-enterprise-economics-serverless-architectures.pdf

Serverless Architectures with AWS Lambda
https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf

Serverless Applications Lens - AWS Well-Architected Framework
https://d1.awsstatic.com/whitepapers/architecture/AWS-Serverless-Applications-Lens.pdf

Streaming Data Solutions on AWS with Amazon Kinesis
https://d1.awsstatic.com/whitepapers/whitepaper-streaming-data-solutions-on-aws-with-amazon-kinesis.pdf

AWS Serverless Multi-Tier Architectures
https://d1.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Archiectures.pdf

Thank you!