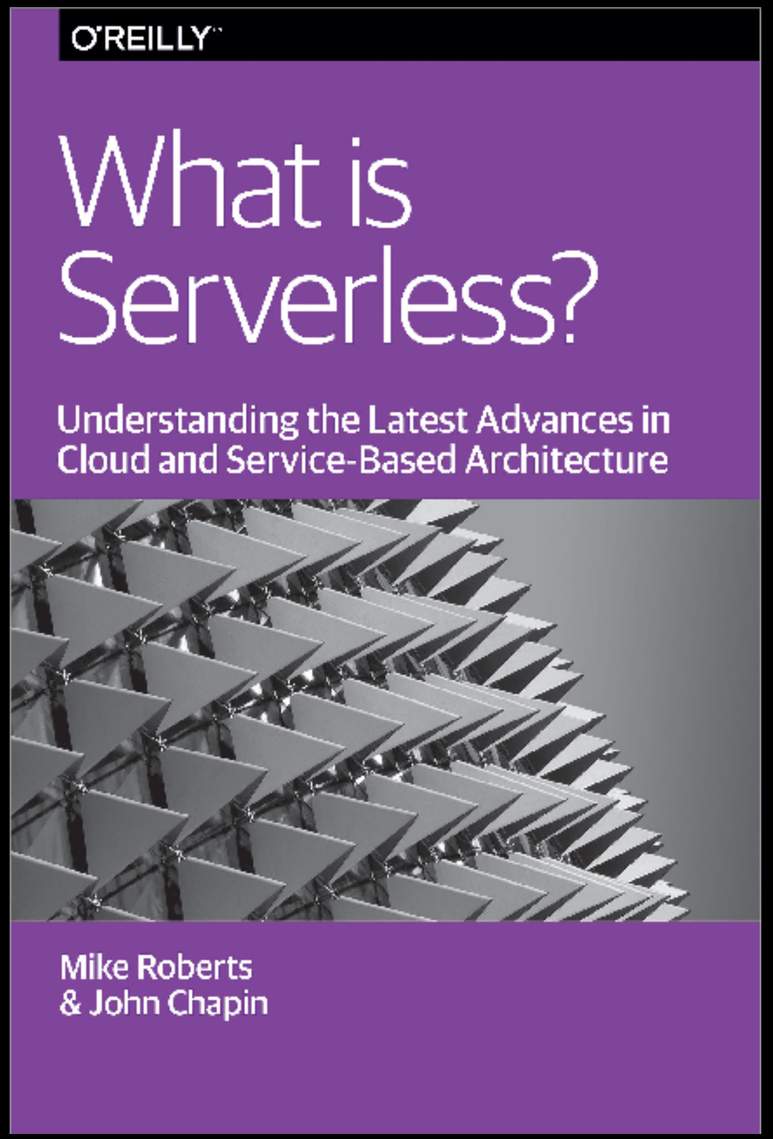
Serverless and Java in the Real World

@johnchapin | john@symphonia.io


O'REILLY
What is Serverless?
Understanding the Latest Advances in Cloud and Service-Based Architecture
Mike Roberts
& John Chapin

Fearless AWS Lambdas

QCon NYC 2017

https://bit.ly/symph-qcon-fearless

Learning Lambda

Mike Roberts
mike@symphonia.io

https://bit.ly/symph-ll

Agenda

1. Choosing Java for Serverless (and AWS Lambda)
2. Structuring a Serverless Java project
3. Logging and metrics
4. Building and deploying
5. Live examples

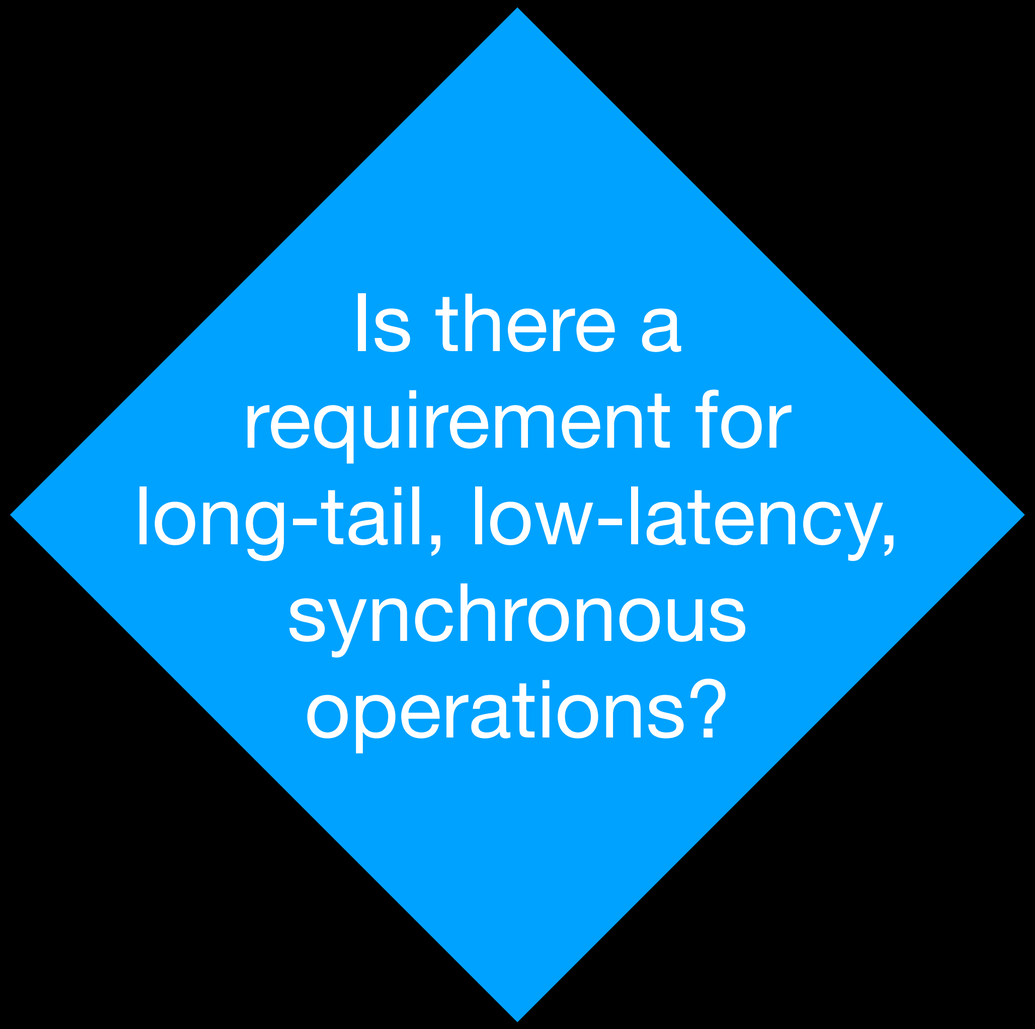
Choosing Java for Serverless (and AWS Lambda)

AWS Lambda runtimes

- Node.js
- Python
- Java (and Scala, Clojure, Kotlin...)
- Go
- C#
- Anything else you want, via a Node.js shim

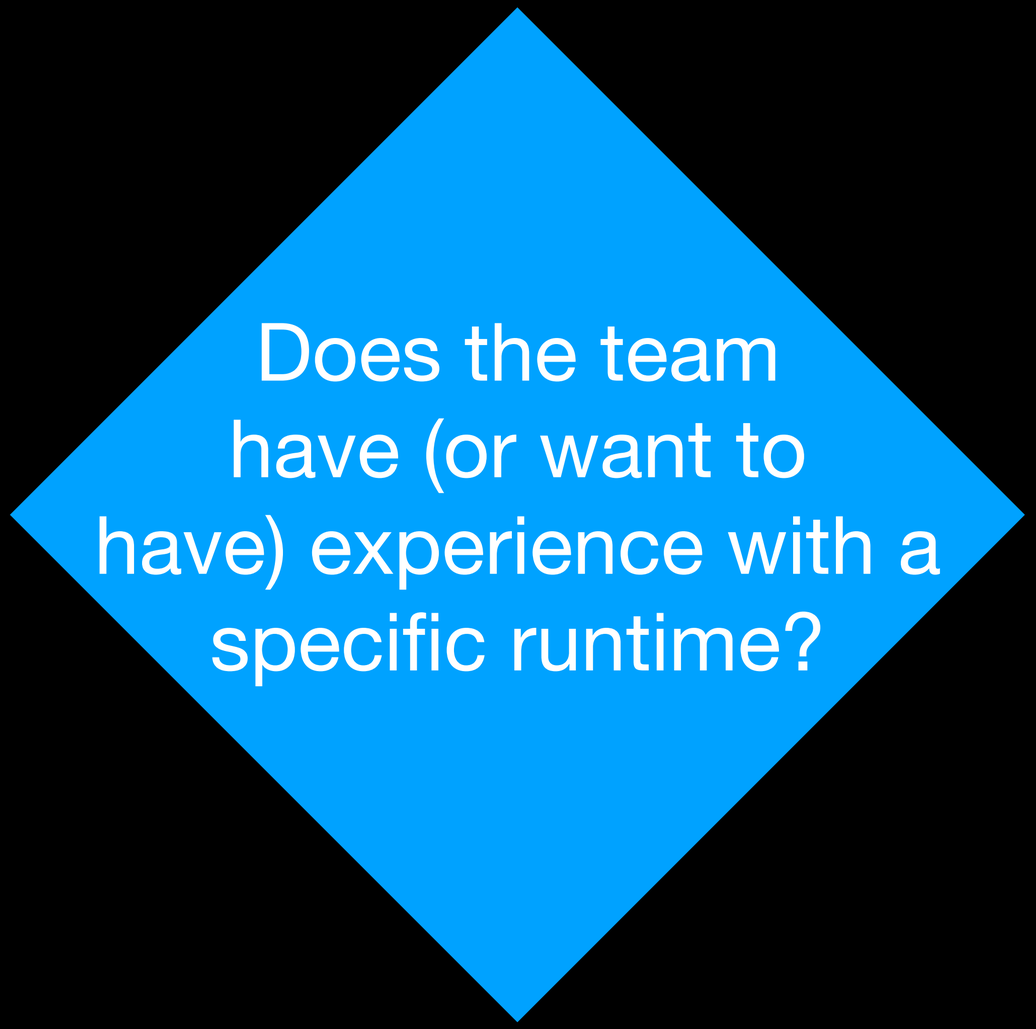
How do we choose a runtime?

Business/Technical Decision



Probably not Java (or C#).

Team Decision



Make the team happy.

The best use case for Serverless Java

Kinesis processing

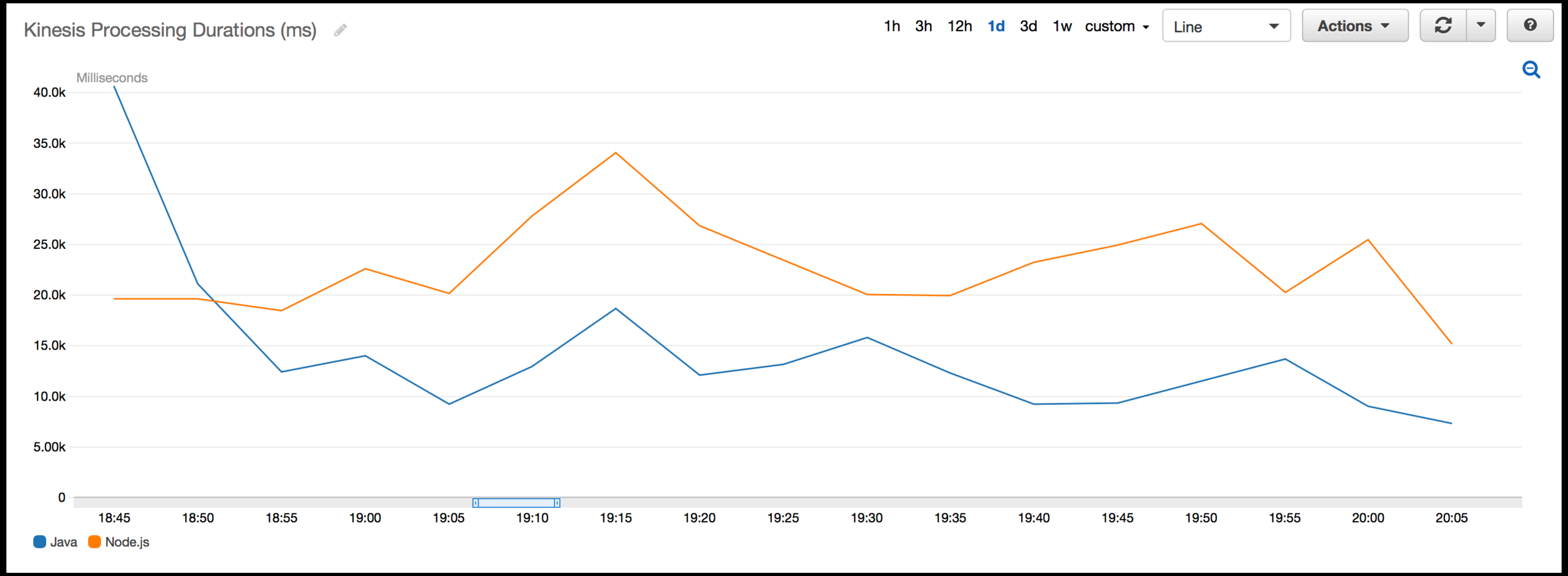
Latency tolerant

Regular, frequent invocations

Not particularly bursty

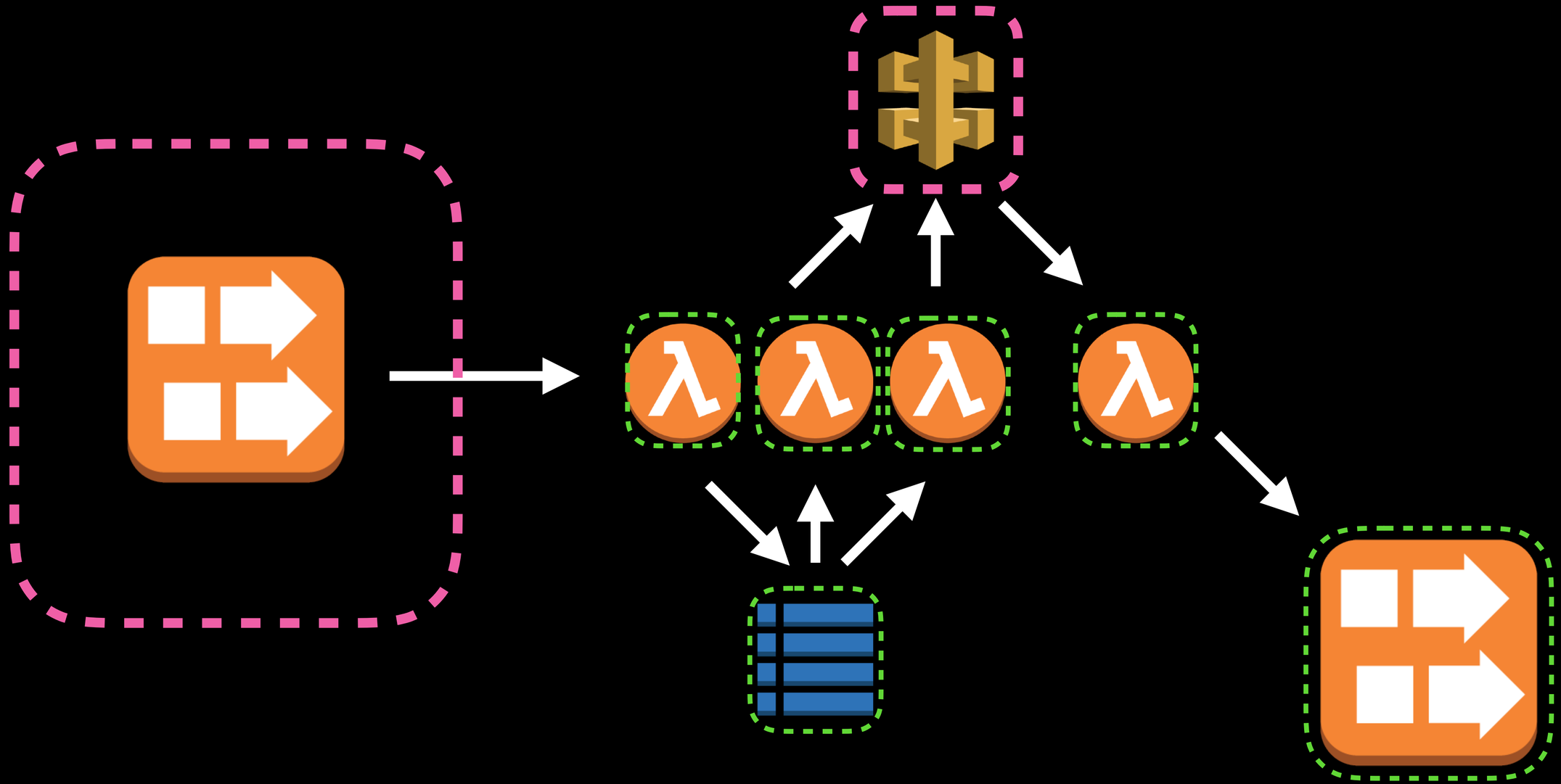
Computationally intensive

Asynchronous, high-throughput



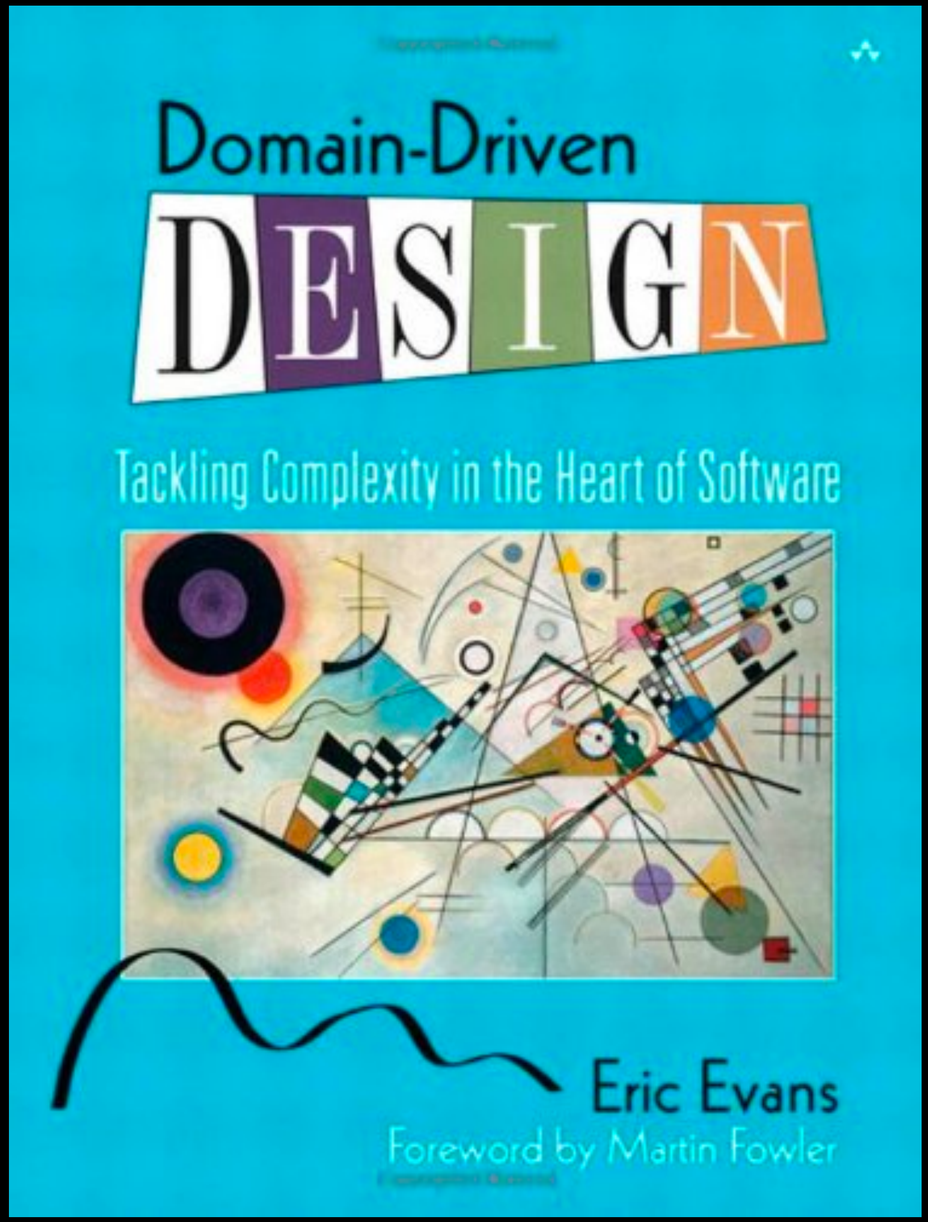
Structuring a Serverless Java Project

What is a project?



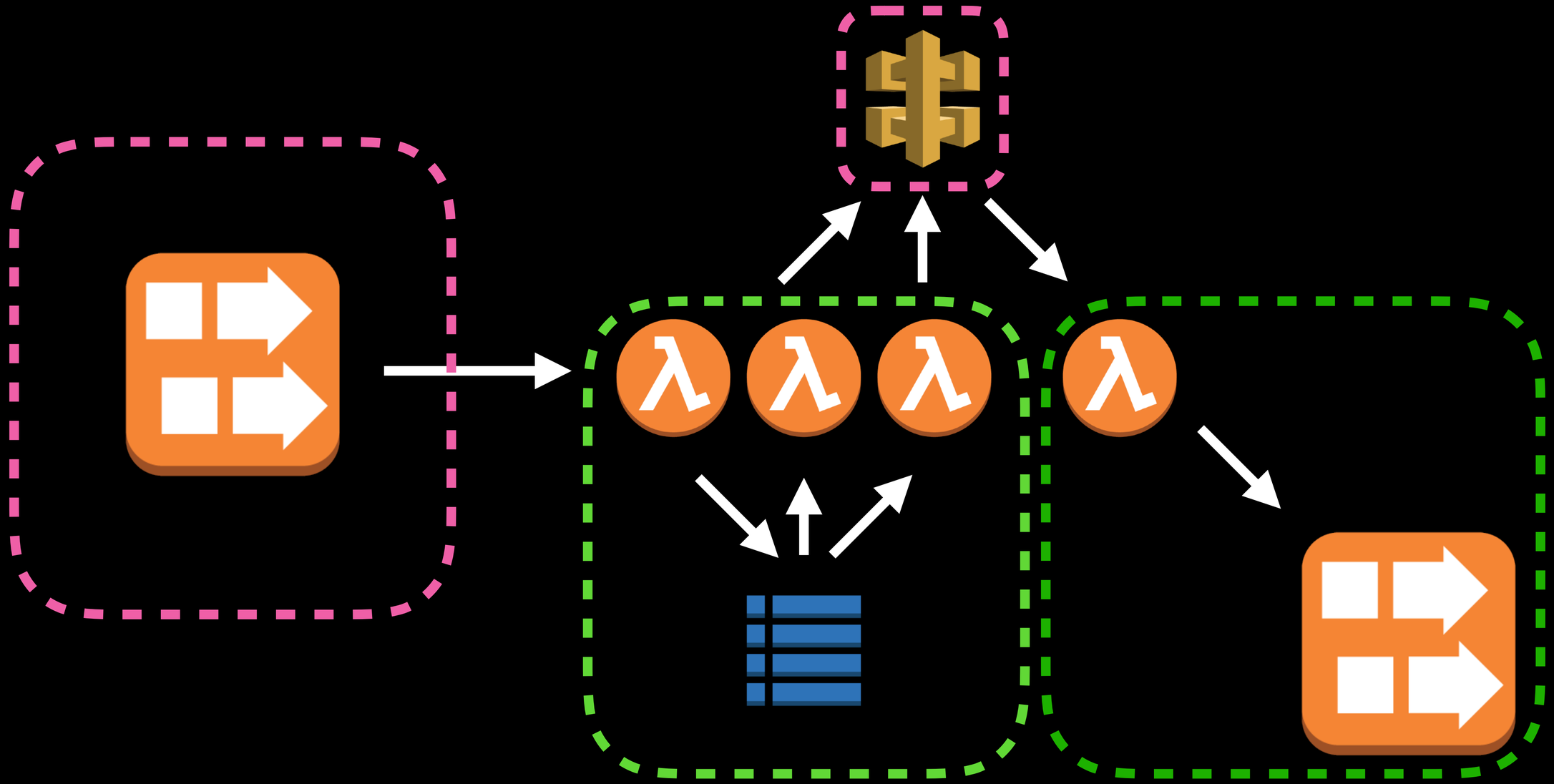
Domain-Driven Design

Eric Evans, 2003



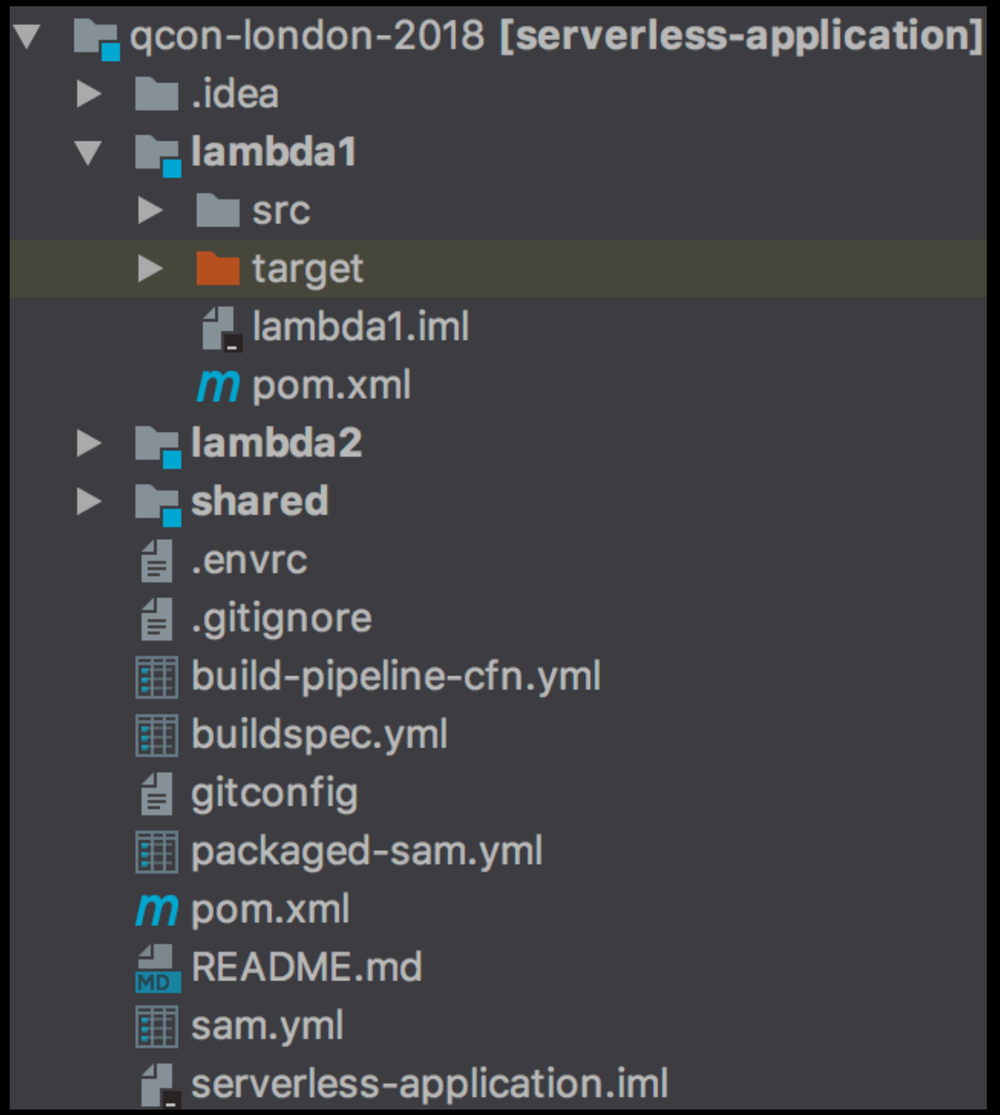
One service = one project

- Service = bounded context (probably)
- A "service" in the AWS Serverless world might be made up of several Lambda functions, and some associated infrastructure.
- The "backbone" infrastructure of the system may not belong to a specific service (but should be owned by a single team).



Multi-module Maven project

- Hierarchical POM files
  - Parent
  - Lambdas
  - Libraries
- AWS Bill of Materials (BOM) at top-level



The Lambda diet

Fewer classes = faster startup

- Ruthlessly cull dependencies

- AWS libraries can be bloated!

mvn dependency:tree, sbt dependencyStats

Other useful libraries

- https://github.com/aws/aws-lambda-java-libs
  - Recently updated, more events and Log4J2 support!
- https://github.com/symphoniacloud/lambda-monitoring
  - Logging + metrics, PRs welcome!

Logging and Metrics

The Present and Future of Serverless Observability

- Yan Cui, yesterday here at QCon

Logging

- System.out/err goes to CloudWatch Logs
- One “log group” per Lambda (by default)
- Within “log group”, one “log stream” per container
- From CloudWatch, can aggregate/forward

System.out.nope

- System.out.println is bad for the normal reasons
- *Real* logging is better
- Lambda runtime can add RequestId to Log4J logs
- NEW! aws-lambda-java-log4j uses Log4J 2

lambda-logging

- SLF4J + Logback
- Sane default configuration w/ AWS RequestId
- Open Source (Apache 2 license)
  - io.symphonia/lambda-logging “1.0.1”
- - github.com/symphoniacloud/lambda-monitoring/

```java
package io.symphonia;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LoggingLambda {

    Logger LOG = LoggerFactory.getLogger(LoggingLambda.class);

    public void handler(String input) {
        LOG.info("Hello, {}", input);
    }
}
```

```
START RequestId: 084c7cbf Version: $LATEST
[2017-04-02 00:32:10.486] 084c7cbf INFO i.s.LoggingLambda - Hello, John
END RequestId: 084c7cbf
REPORT RequestId: 084c7cbf Duration: 1.52 ms Billed Duration: 100 ms …
```

CloudWatch Metrics

- No built-in business metrics
- Lambda platform metrics
  - Errors, Duration, Invocations, Throttles
- Naive metrics collection approach is dangerous!
  - Cloudwatch has account-level API limits 🔥

CloudWatch Metric Filters

- Built into Cloudwatch! Scalable!
- Scrape Cloudwatch Logs data using special (finicky) patterns
- Generates and posts Cloudwatch metrics in batch

lambda-metrics

- Codahale metrics and lambda-logging
- Maven plugin builds Metric Filters to scrape logs, post to CloudWatch Metrics
- Open Source (Apache 2 license)
  - io.symphonia/lambda-metrics “1.0.1”
  - github.com/symphoniacloud/lambda-monitoring

```java
package io.symphonia;

import com.codahale.metrics.Counter;
import io.symphonia.lambda.annotations.CloudwatchMetric;
import io.symphonia.lambda.metrics.LambdaMetricSet;
import org.slf4j.*;

public class MetricLambda {

    Logger LOG = LoggerFactory.getLogger(MetricLambda.class);

    private class Metrics extends LambdaMetricSet {
        @CloudwatchMetric
        Counter inputBytes = new Counter();
    }

    public void handler(String input) {
        Metrics metrics = new Metrics();
        metrics.inputBytes.inc(input.length());
        metrics.report(LOG);
    }
}
```

```
START RequestId: 084c7cbf Version: $LATEST
084c7cbf METRIC i.s.Lambda type COUNTER name \
   io.symphonia.Lambda.Metrics/inputBytes count 3
END RequestId: 084c7cbf
REPORT RequestId: 084c7cbf Duration: 1.52 ms Billed \
   Duration: 100 ms …
```

```
[request_id, level=METRIC, logger, type_label,
 type=COUNTER, name_label,
 name="io.symphonia.Lambda.Metrics/inputBytes",
 count_label, count]
```

```
START RequestId: 084c7cbf Version: $LATEST
084c7cbf METRIC i.s.Lambda type COUNTER name \
   io.symphonia.Lambda.Metrics/inputBytes count 3
END RequestId: 084c7cbf
REPORT RequestId: 084c7cbf Duration: 1.52 ms Billed \
   Duration: 100 ms …
```

```
[request_id, level=METRIC, logger, type_label,
 type=COUNTER, name_label,
 name="io.symphonia.Lambda.Metrics/inputBytes",
 count_label, count]
```

```
START RequestId: 084c7cbf Version: $LATEST
084c7cbf METRIC i.s.Lambda type COUNTER name \
   io.symphonia.Lambda.Metrics/inputBytes count 3
END RequestId: 084c7cbf
REPORT RequestId: 084c7cbf Duration: 1.52 ms Billed \
   Duration: 100 ms …
```

```
[request_id, level=METRIC, logger, type_label,
 type=COUNTER, name_label,
 name="io.symphonia.Lambda.Metrics/inputBytes",
 count_label, count]
```

Building and Deploying

Continuous Delivery in an Ephemeral World

O'Reilly SACON NYC 2018

https://conferences.oreilly.com/software-architecture/sa-ny/public/schedule/detail/63860

Build pipeline

- Infrastructure as code!
  - AWS CodePipeline / CodeBuild
- Separate repository
  - e.g., "serverless-app-build" repository
- Cache dependencies (via CodeBuild caching)

Application and infrastructure

- **Continuously deploy code *and* infrastructure**
- Alongside the application source code
  - **buildspec.yml**
  - **sam.yml**

SAM vs Serverless Framework

- Serverless Application Model
  - AWS-specific
  - Different flavor of CloudFormation
- Serverless Framework
  - 3rd-party, supports multiple cloud vendors
  - CloudFormation under the hood (for AWS)
  - Plugin architecture

Reproducible builds

- Maven-produced JAR files are non-deterministic
  - Datetime stamp in "pom.properties"
  - Other timestamps, file ordering, etc
- SAM relies on file hashes to detect updates

Examples

https://github.com/symphoniacloud/qcon-london-2018

Flash sale!

http://bit.ly/symph-qcon-2018-demo

In conclusion

- For the right use-case, choose the Java runtime!
- Services = multi-module Maven projects
- Log and collect metrics correctly and scalably
- Infrastructure as code, continous delivery are paramount

Questions?

@johnchapin | john@symphonia.io