Fully Serverless

Pros and Cons - A Case Study

June 2018

Stephen Colebourne

OPENGAMMA

goto; amsterdam

Please

Ask questions through the app

Rate Session

Thank you!

@GOTOamst

gotoams.nl

Stephen Colebourne

OPENGAMMA

- Engineering Lead at OpenGamma
- Worked at OpenGamma for 8 years
- Creator of Joda Time, `java.time` and many others
- Java Champion, blogger

OpenGamma

OPENGAMMA

- Fintech startup dedicated to finance industry analytics
- Based in London and NY
- Modern SaaS products developed on the JVM, primarily Java
- Used by banks, hedge funds, pension funds, other vendors
- New platform built on AWS starting early 2017
- Underpinned by our award-winning open source analytics library, Strata (strata.opengamma.io)

Contents

OPENGAMMA

- Introduction
- Data services
- Margin service
- Analytics service
- Ops/Infrastructure
- Conclusions

Introduction

What is serverless?

OPENGAMMA

- Many different definitions
- For us it means the infrastructure is invisible
  - We don’t know where our code runs
  - We have no control over where our code runs

AWS compute

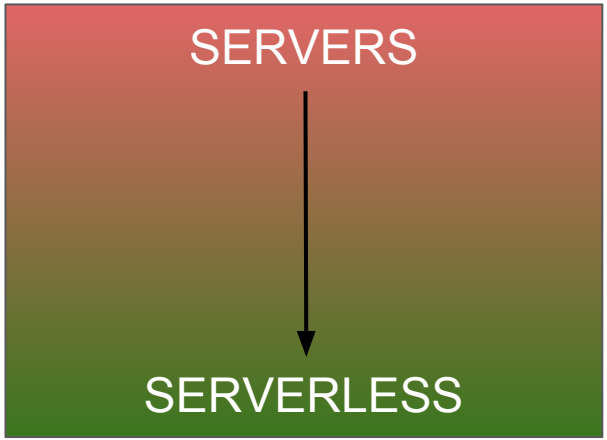
OPENGAMMA

- EC2 - IaaS
- Elastic Container Service - CaaS
- Elastic Beanstalk - PaaS
- Batch
- Lambda - FaaS

AWS compute

OPENGAMMA

- EC2 - IaaS
- Elastic Container Service - CaaS
- Elastic Beanstalk - PaaS
- Batch
- Lambda - FaaS



AWS Lambda

OPENGAMMA

- Simple interface, not unlike Servlets
- Once method completes, the Lambda terminates

```
public interface RequestStreamHandler {
  public void handleRequest(
       InputStream input,
       OutputStream output,
       Context context) throws IOException;
}
```

AWS Lambda

OPENGAMMA

- Package it up as a jar-with-dependencies
- Upload it to AWS to be run

```
public class MyLambda implements RequestStreamHandler {
  public void handleRequest(
      InputStream input,
      OutputStream output,
      Context context) throws IOException {
    return “Hello world”;
  }
}
```

Other AWS services

OPENGAMMA

- Lambda provides compute, also need...
- DynamoDB - NoSQL data store
- API Gateway - REST APIs
- S3 - File storage
- CloudWatch - Logging
- ...

OpenGamma systems

OPENGAMMA

- Analytics Service
  - Batch calculations
  - Web interface
- Margin Service
  - Stateless on-demand calculations
  - REST API and Web interface
- Data Services
  - Download & Process industry data
  - API for Lookup

Data services

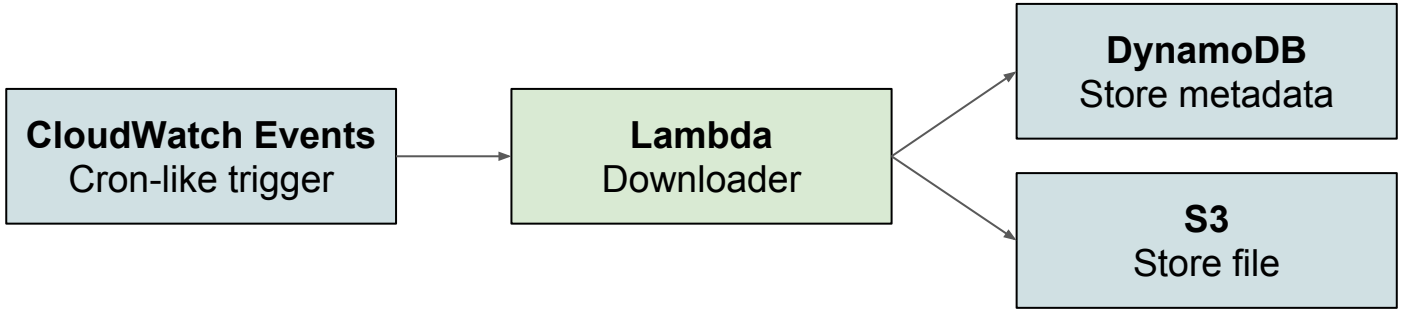
Data Services

OPENGAMMA

- Reference data and market data used in calculations
- Downloaded from key industry data providers
- Multi-stage processing pipeline
  - Download data files from provider to S3
  - Process data files to create Java objects
- Expose data to calculations

Data Services - Downloaders

OPENGAMMA



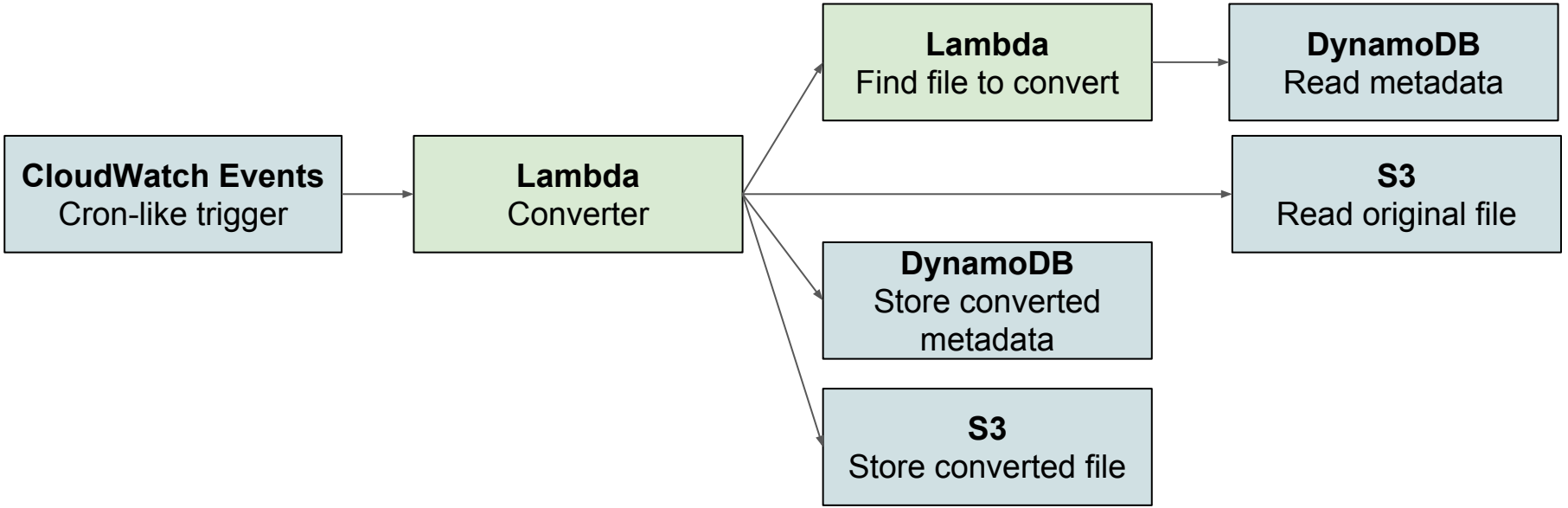
Triggering a Lambda

OPENGAMMA

- Periodic / Cron-like
- Store a file in S3
- Update a DynamoDB table
- External API call
- From a Queue
- Directly, such as from another Lambda
- ...

Data Services - Converters

OPENGAMMA



Our standard data pattern

OPENGAMMA

- Metadata in DynamoDB
- Data in S3
- Data too large for storing in DynamoDB
  - Limit is 400kb per “row”
- Lambda invocation sends back metadata, not data
  - Limit is 128Kb from triggers (6Mb in request/response mode)
  - Better for caller to directly query S3

Our standard data pattern

OPENGAMMA

- Calls to/from one DynamoDB table restricted
- Call from Lambda to another Lambda to perform query
- Packaged up as a nice Java API
  - Hides Lambda and S3 calls

```
IndustryDataService svc = IndustryDataService.remote();
IndustryDataRequest req = …
IndustryDataResponse rsp = svc.query(request);
byte[] bytes = svc.fileAsBytes(rsp.getFileLocation());
```

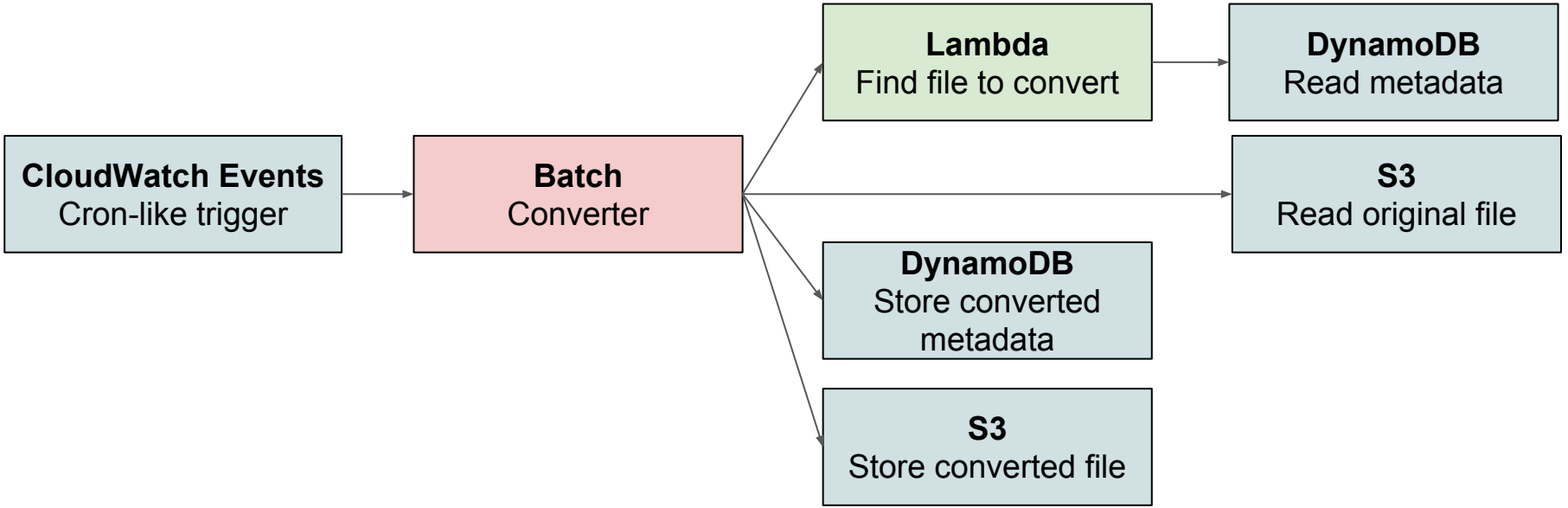
Escaping the limits

OPENGAMMA

- Lambda limits:
  - 5 minute timeout
  - Max 3Gb memory
  - 512Mb disk space
  - 1024 threads + processes
  - 1000 instances (can be increased)
  - 50Mb jar-with-dependencies

Data Services - Converters

OPENGAMMA



Escaping the limits

OPENGAMMA

- Batch intended for larger/longer data processing
- Submit a job, will be processed at some point
- Serverless - runs in Docker on EC2, but auto-scaling
- Can trigger Batch from CloudWatch Events
  - Recent AWS enhancement

Lambda vs Batch

OPENGAMMA

- Prefer Lambda
  - Easier to use and manage
- Batch is useful fallback
  - But only if time to start job doesn’t matter
  - Will need a “fetch and run” docker image to run jar file

Lambda vs Batch

OPENGAMMA

- Prefer Lambda
  - Easier to use and manage
- Batch is useful fallback
  - But only if time to start job doesn’t matter
  - Will need a “fetch and run” docker image to run jar file
- What if time to start job does matter?
  - There is no serverless compute solution available

Lambda vs Traditional server

OPENGAMMA

- 4 services, formed of a total of 23 Lambda & 2 Batch
- Cron triggering via infrastructure
- Each Lambda performing a small isolated task

- 1 server instance running continuously
- Java code would perform cron triggering
- Good software discipline needed, need to avoid rogue tasks

Margin Service

Margin Service

OPENGAMMA

- Financial calculation
  - Input - a portfolio of trades
  - Output - the margin, detailed breakdown, any errors
- Stateless, run on-demand
- REST API
  - Programmatic access
  - Web interface

Margin Service

OPENGAMMA

- First implementation on Elastic Beanstalk (EB)
- Rewritten to use some Lambda
- Then reworked to use Lambda only

Margin Calculations - Data

OPENGAMMA

- Calculations use small subset of large data set
- On EB, large data not such a big problem
  - Load full set of market data into memory at startup
  - Same set of data used for every request - excellent performance

Margin Calculations - Move to Lambda

OPENGAMMA

- Cannot load the full data set
  - Too slow to do on each lambda invocation
  - Uses too much memory (>4GB in some cases)
- Only load the data needed for the calculation
  - Data must be pre-processed into small subsets
- Significant complexity in pre-processing
- No possibility of caching between requests
  - Each calculation needs a slightly different subset of the data

Margin Calculations - Migration impact

OPENGAMMA

- Effectively a rewrite to manage data better
- Key business benefit was being able to process any date
  - Not just the current date
- Greater latency, but acceptable
  - Lambda has to load data each time it is invoked

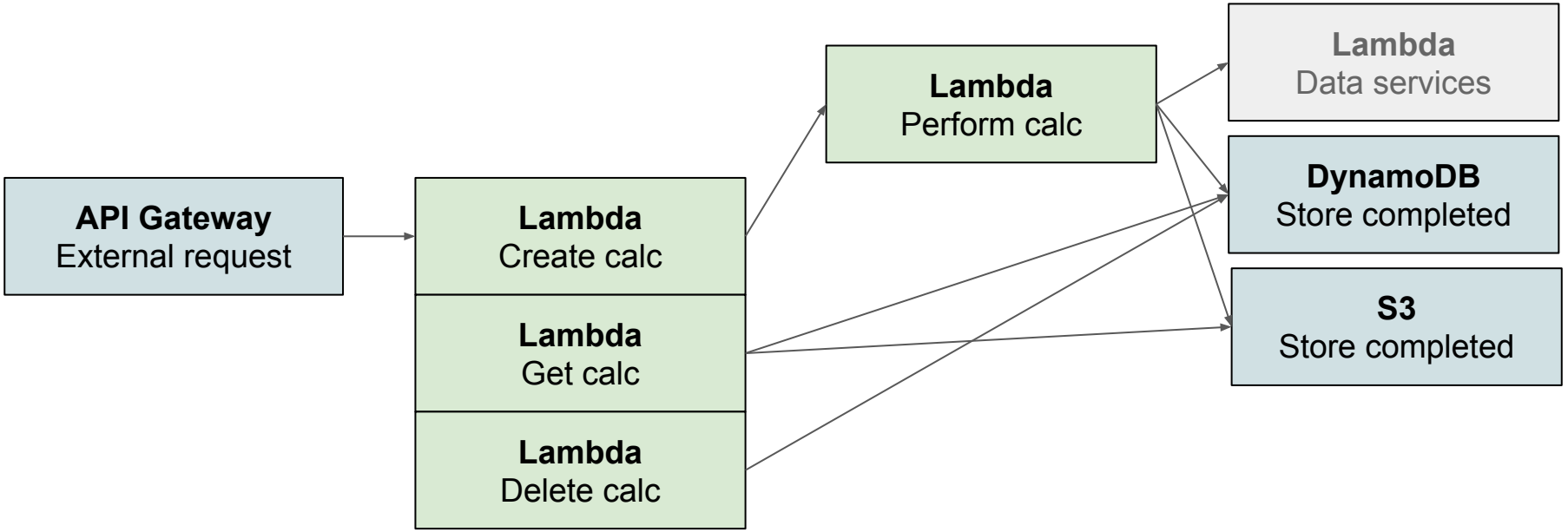
Margin Calculations - Scaling

OPENGAMMA

- Scalability is a headline feature of Lambda
- It really works!
  - Scales transparently in response to spikes
  - Scales down to zero when idle
- No silver bullet
  - A system is only as scalable as its least scalable component
  - DynamoDB and S3 have caused us problems

Margin calculation

OPENGAMMA



Polling with Lambdas

OPENGAMMA

- Margin calculation can be slow
- Client creates a calculation and polls for the result
- Lambda has no convenient way to handle this
  - Caller invokes a Lambda to create the calculation
  - When that returns “calculation created” it is finished
  - Must call a second async Lambda to perform the actual calculation
- Normally a simple coding problem to run background task
- Requires infrastructure changes with Lambda

Margin Service - REST API

OPENGAMMA

- API defined in API Gateway
  - Provides some throttling/security benefits
- Request handling logic in a Lambda

Serverless REST APIs

OPENGAMMA

- API Gateway + Lambda a good fit for our case
  - Stateless web app
  - All state is in the data stores
- But difficult to use
  - API Gateway configuration is complex & verbose
  - Integration with Lambda is low level

Serverless REST APIs - libraries

OPENGAMMA

- No good choices for JVM when we needed them
- One Lambda for each endpoint is overly complex
- Created our own
  - Inspired by JAX-RS - one annotated method per endpoint
  - All requests dispatched to one lambda
  - Library code routes request to method

Serverless REST APIs - libraries

OPENGAMMA

- More choices now
  - AWS SAM (still quite low level)
  - aws-serverless-java-container (built on AWS SAM)
    - Embed Spring, Spark Java, Jersey apps in a lambda
  - Osiris (www.osiris.ws)

Serverless REST APIs - Challenges

OPENGAMMA

- Cold Start
  - JVM Lambdas take time to start
  - Lambdas are reused, typically live for a few minutes
  - Built a keep-alive mechanism to take advantage of this
    - A hack, but effective
- Routing all endpoints through one Lambda helps

Serverless REST APIs - Challenges

OPENGAMMA

- Development workflow
  - Need to deploy to AWS to test
  - Hard to debug
  - Contention over shared dev environment
  - Rebuilt API by hand using Spark Java in to run locally
    - Ideally this should be provided by a library

Lambda vs Traditional server

OPENGAMMA

- 9 Lambda, split because of polling
- Requires pre-processing to get data manageable
- Scales very well, at the expense of latency

- Would need more than one server
- Would still have needed data pre-processing
- Would have had fixed scaling limits

Analytics Service

Analytics Service

OPENGAMMA

- Web interface - API Gateway + Lambda
- Daily batch calculations - AWS Batch
- Client portfolio data and calculation results persisted daily

Analytics Service - Data Security

OPENGAMMA

- Inputs to calculations include client portfolios
- Calculation results include position data
- Data is *highly* sensitive
- Leaking client data would be a disaster
- Sending data to the wrong user would be a disaster

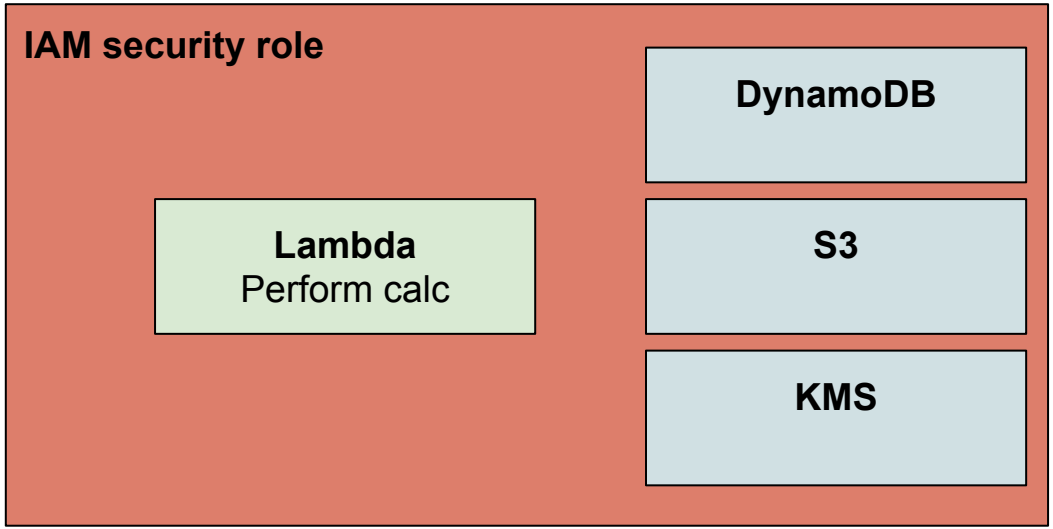
Analytics Service - Tenant Segregation

OPENGAMMA

- Data for each client is encrypted with a different key
  - Using S3 server-side encryption and KMS
- Any process handles exactly one client
  - Only knows about one set of credentials - STS
  - Can only access one encryption key
- Security threaded through architecture (defense in depth)
  - Provided by both infrastructure and code
  - Enforced by trusted AWS security primitives

Analytics Service - Data Isolation

OPENGAMMA



Lambda vs Traditional server

OPENGAMMA

- Each Lambda runs as a Role, simple effective tenant
- Batch works well for daily overnight calculations

- Would need more than one server
- Secure tenant segregation would require excellent coding or lots of servers

Ops

Continuous Integration / Deployment

OPENGAMMA

- Pipeline deploys directly to AWS
- Deployed versions controlled via files in GitHub
- Dev normally deployed after each PR merged to master
- Prod deployed from a tag
  - Tag code
  - Test tagged code
  - Update prod versions file with new tag
  - Tagged version is deployed

CI / CD Pipeline

OPENGAMMA

- GitHub
- Shippable - builds and uploads artifacts to S3
- AWS CodePipeline - runs when new artifacts detected
- Lambda - loads versions file and controls deployment

Logging

OPENGAMMA

- Highly distributed design means fragmented logs
- Difficult to find the source of an issue
- Everything goes into AWS CloudWatch Logs
  - Usability isn’t great
- Sumo Logic for log aggregation
  - Lambda writes log events from CloudWatch to Sumo Logic
  - Powerful querying capabilities
  - Structured logging - log JSON to enable querying

Monitoring / Alerts

OPENGAMMA

- CloudWatch metrics
  - High-level metrics, e.g.
    - Failure count for lambdas
    - 5xx errors for API Gateway endpoints
  - Can trigger alerts
  - Hard to find source of failure
  - Not actually very useful

Monitoring / Alerts

OPENGAMMA

- Sumo Logic alerts
  - Driven by log events
  - Queries executed against incoming events

Infrastructure

Infrastructure as Code

OPENGAMMA

- All AWS resources defined in declarative configuration files
- Stored in GitHub
  - Versioned
  - Changes reviewed
- Environment can be rebuilt from scratch
- Uses Terraform
  - AWS CloudFormation is an alternative

Terraform

OPENGAMMA

- Declarative format for defining AWS resources
- Command-line tool to diff and apply changes
- Verbose and fiddly
  - Reflects the underlying AWS model
  - Doesn’t add any higher-level abstractions
- Slightly flaky but does the job

Conclusions

Costs & Benefits

OPENGAMMA

- Serverless architecture has definite costs and benefits
- Depends on the use case which is bigger

Costs

OPENGAMMA

- Building from small simple pieces pushes complexity elsewhere
  - The infrastructure
  - The interactions between components
- No single high-level view of the application
  - Existing analysis tools don’t help much (e.g. IDEs)
- Monitoring and Alerting
  - Highly fragmented system is harder to manage
- Lambdas have restrictive limits

Benefits

OPENGAMMA

- Not thinking about servers is as good as it sounds
- Transparent scaling to handle load
- Scales down to zero when idle
  - Potential large cost savings
  - Lowers the barrier to entry - more use cases are viable
- Great solution for tenants enforced by AWS security
- Simple programming model - everything is a function
- Simple deployment model - jar / zip files

Final Thoughts

OPENGAMMA

- Serverless can be great and it can be painful
  - Depends very much on the use case
- Convergence between services would be welcome
  - FaaS and CaaS are points on a continuum
- The technology is young
  - Platform limitations are likely to be lifted
  - Tooling is bound to improve

goto; amsterdam

Please

Remember to rate this session

Thank you!

@GOTOamst

gotoams.nl