seeka.TV
AND YOU SHALL FINDA

COLUMBIA UNIVERSITY

Building and Teaching a Complete Serverless Solution

Donald F. Ferguson

Professor of Professional Practice, Dept. of Computer Science, Columbia University
CTO and Co-Founder, Seeka TV
(dff@cs.columbia.edu, donald.ferguson@seeka.tv, dff9@columbia.edu)

Third International Workshop On Serverless Computing (WoSC) 2018,
San Francisco, CA. 02-Jul-2018
(https://www.serverlesscomputing.org/wosc3/)

The Solution

Two elements to the company

- Multi-tenant platform for managing and enabling access to content.
- A “channel” focused on enabling independent series producers to build and audience.

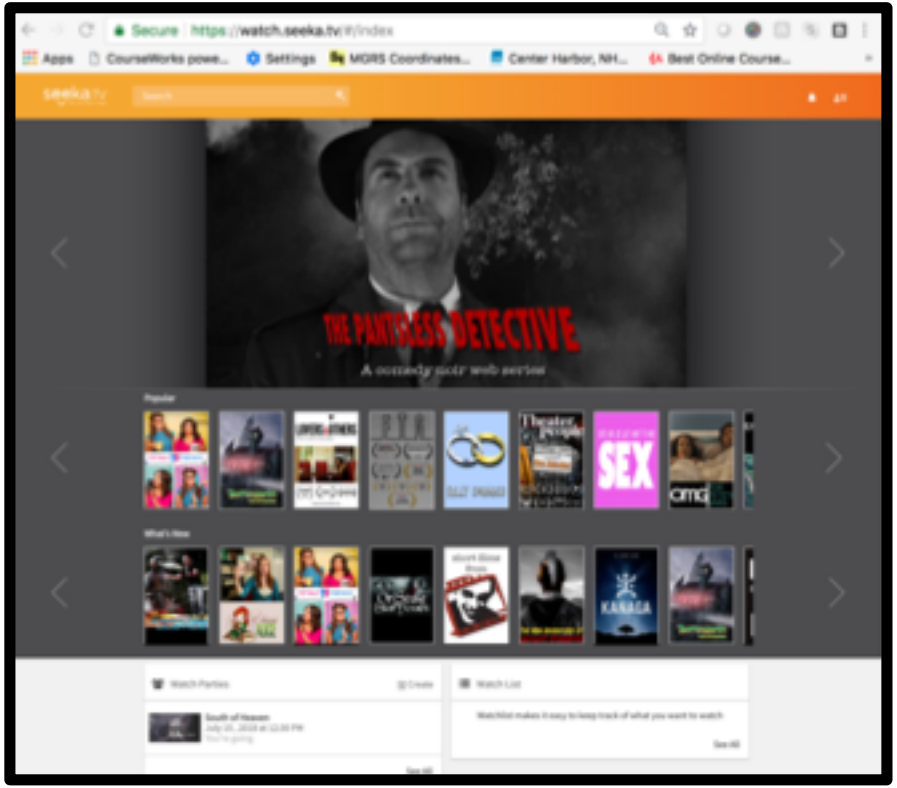
End-Users

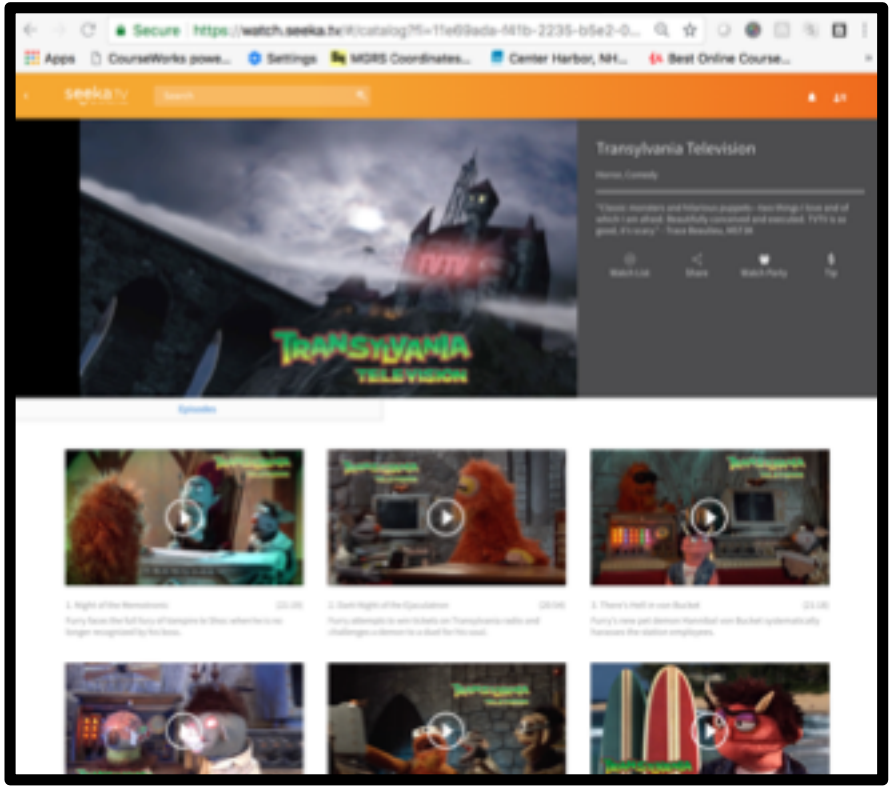
- Browse and search.
- Access (watch content)
- Interact
  - Comment.
  - Simultaneous viewing and chat.
  - Social media integration.
  - Rate.
  - Pay/Tip

Providers

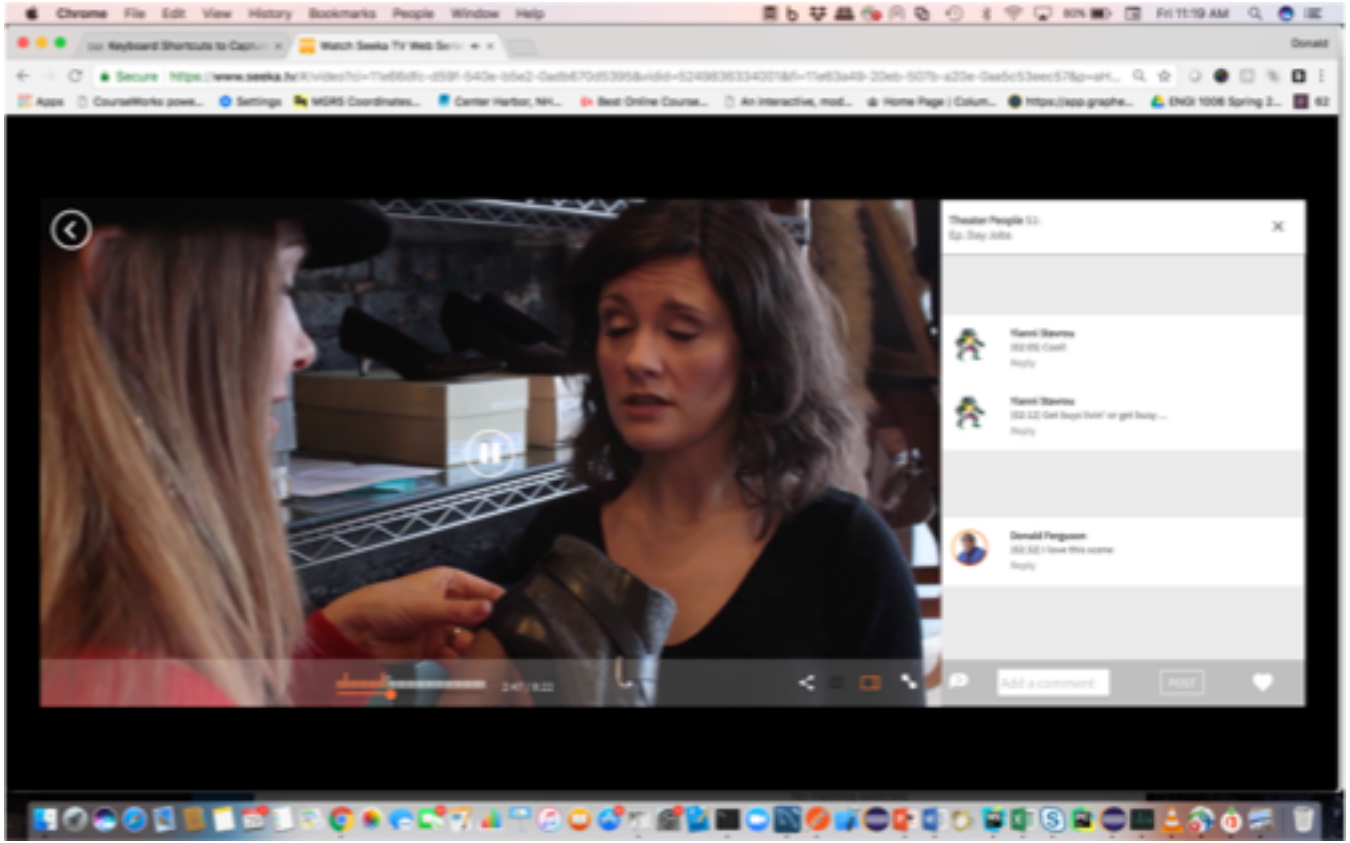
- Manage content and metadata.
- Control access.
- Pay-Wall integration.
- Schedule and control placement.
  - Private channel.
  - Web sites.
  - Social media.
- Statistics and analysis.

The Solution

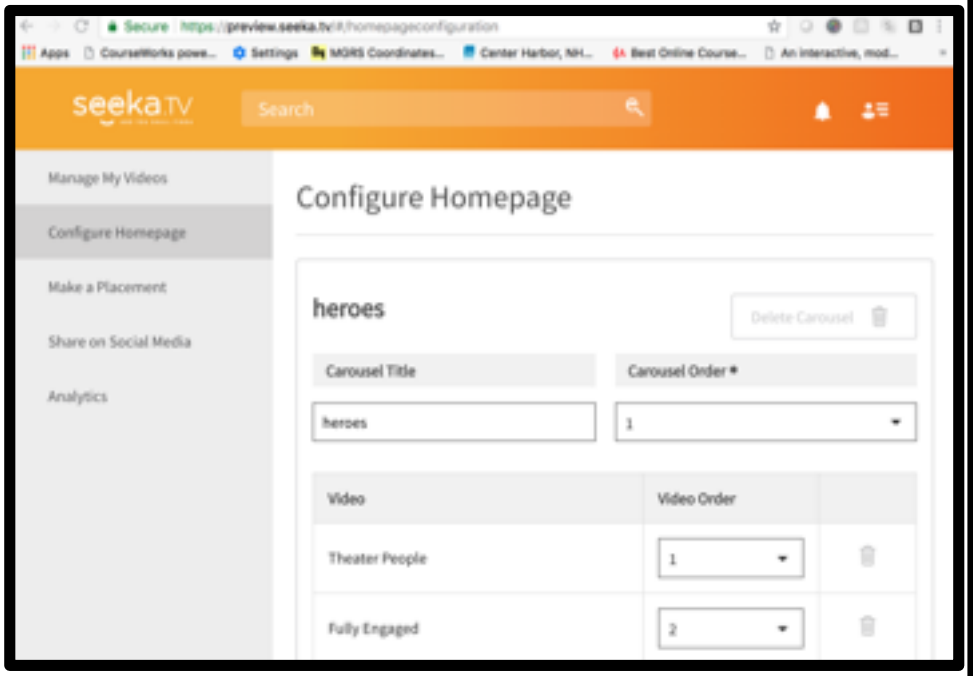


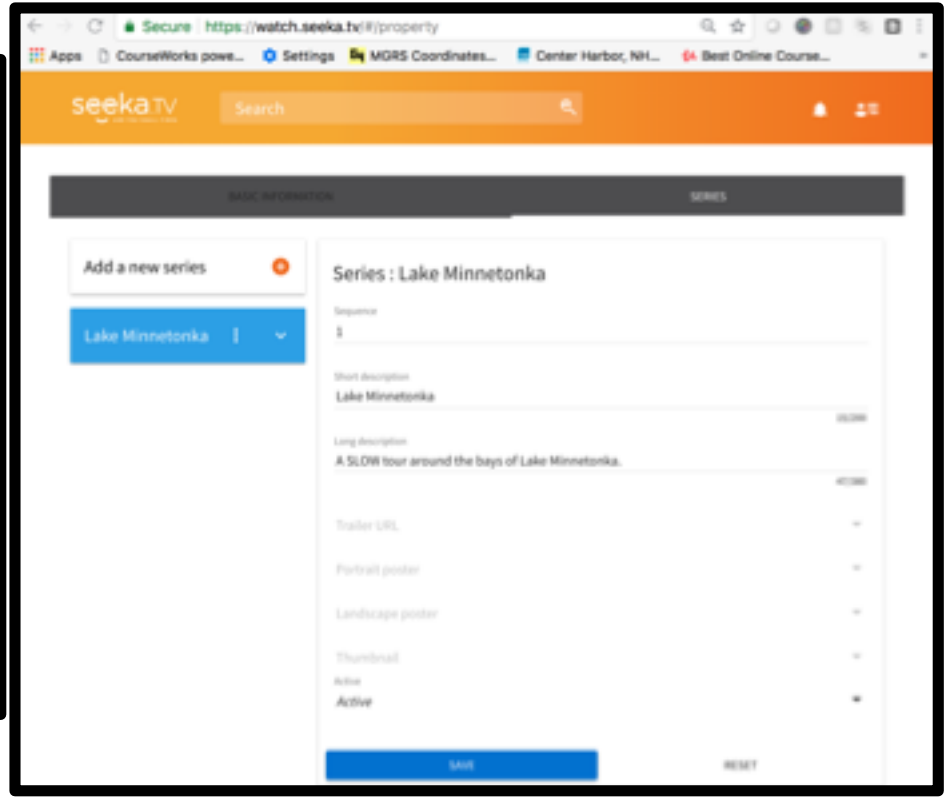


The Solution

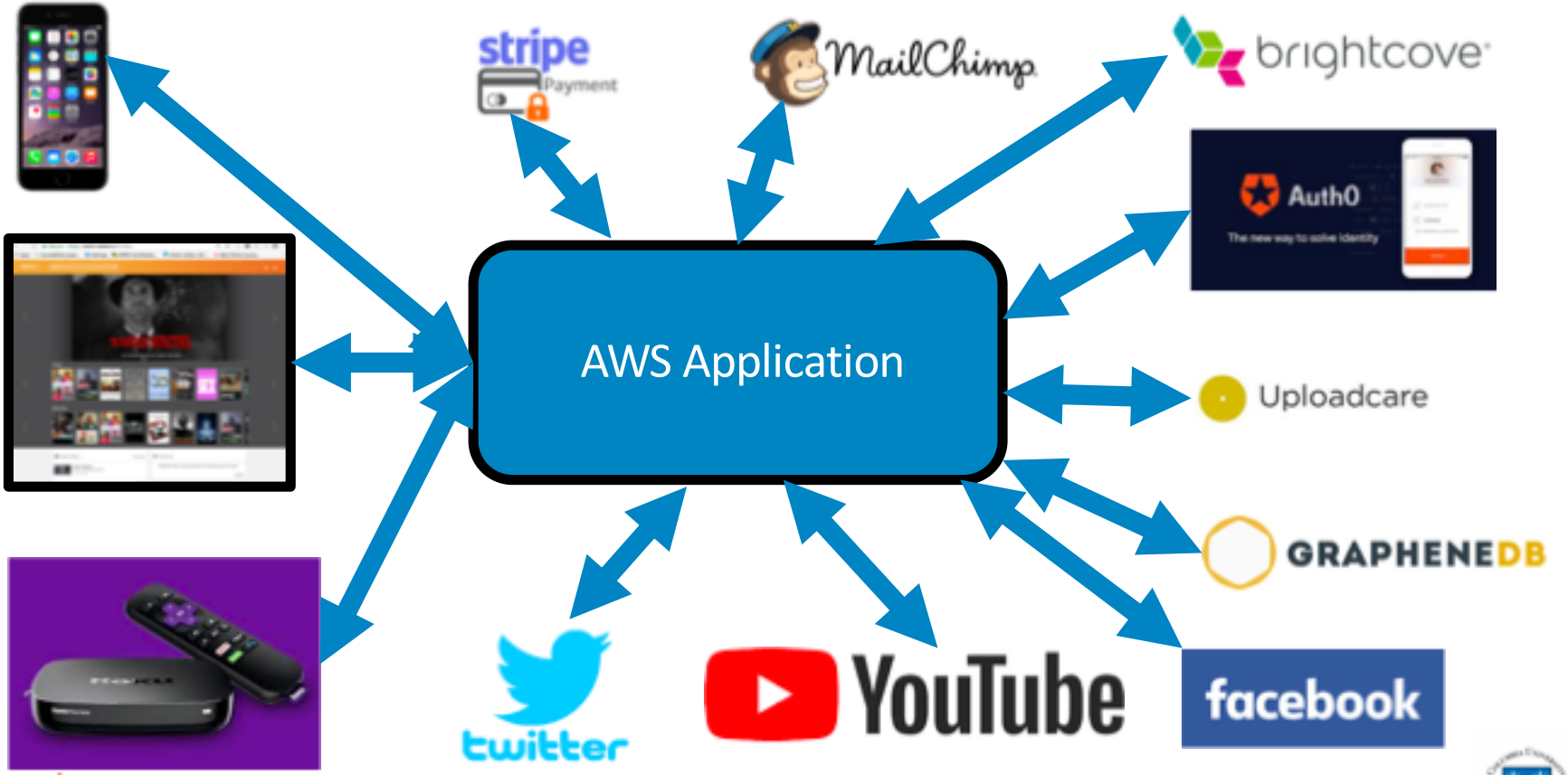


The Solution





Implementation Architecture



AWS Services

Compute
- EC2
- Elastic Container Service
- Lambda
- Batch
- Elastic Beanstalk

Storage
- S3
- EFS
- Glacier
- Storage Gateway

Database
- RDS
- DynamoDB
- ElastiCache
- Neptune
- Amazon Redshift

Migration
- AWS Migration Hub
- Application Discovery Service
- Database Migration Service
- Server Migration Service
- Snowball

Networking & Content Delivery
- VPC
- CloudFront
- Route 53
- API Gateway
- Direct Connect

Analytics
- Athena
- EMR
- CloudSearch
- Elasticsearch Service
- Kinesis
- QuickSight
- Data Pipeline
- AWS Glue

Security, Identity & Compliance
- IAM
- Cognito
- Secrets Manager
- GuardDuty
- Inspector
- Amazon Macie
- AWS Single Sign-On
- Certificate Manager
- CloudHSM
- Directory Service
- WAF & Shield
- Artifact

Mobile Services
- Mobile Hub
- AWS AppSync
- Device Farm
- Mobile Analytics

AR & VR
- Amazon Sumerian

Application Integration
- Step Functions
- Amazon MQ
- Simple Notification Service
- Simple Queue Service
- SWF

Developer Tools
- CodeStar
- CodeCommit
- CodeBuild
- CodeDeploy
- CodePipeline
- Cloud9
- X-Ray

Management Tools
- CloudWatch
- AWS Organizations
- CloudFormation
- CloudTrail
- Config
- OpsWorks
- Service Catalog
- Systems Manager
- Trusted Advisor
- Managed Services

Media Services
- Elastic Transcoder
- Kinesis Video Streams
- MediaConvert
- MediaLive
- MediaPackage
- MediaStore
- MediaTailor

Machine Learning
- Amazon SageMaker
- Amazon Comprehend
- AWS DeepLens
- Amazon Lex
- Machine Learning
- Amazon Polly
- Rekognition
- Amazon Transcribe
- Amazon Translate

Customer Engagement
- Amazon Connect
- Pinpoint
- Simple Email Service

Business Productivity
- Alexa for Business
- Amazon Chime
- WorkDocs
- WorkMail

Desktop & App Streaming
- WorkSpaces
- AppStream 2.0

Internet Of Things
- IoT Core
- IoT 1-Click
- IoT Device Management
- IoT Analytics
- Greengrass
- Amazon FreeRTOS

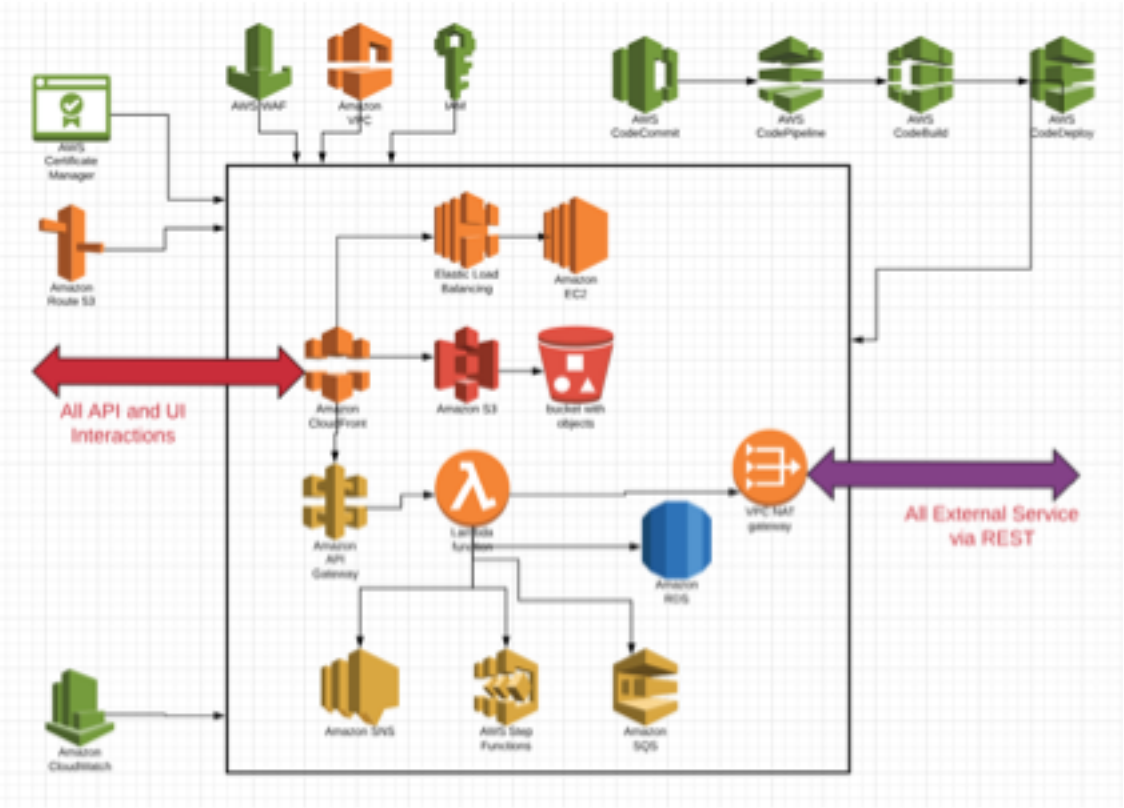
Game Development
- Amazon GameLift

AWS Services

Comments:

- I probably forgot a couple.
- There is a bunch we should/will use:
  - Cognito, SSO
  - CloudSearch, QuickSight, …
  - Media services
  - AI (e.g. for recommendation)
- And a lot of the 3rd parties we use are themeselves built on top of AWS.

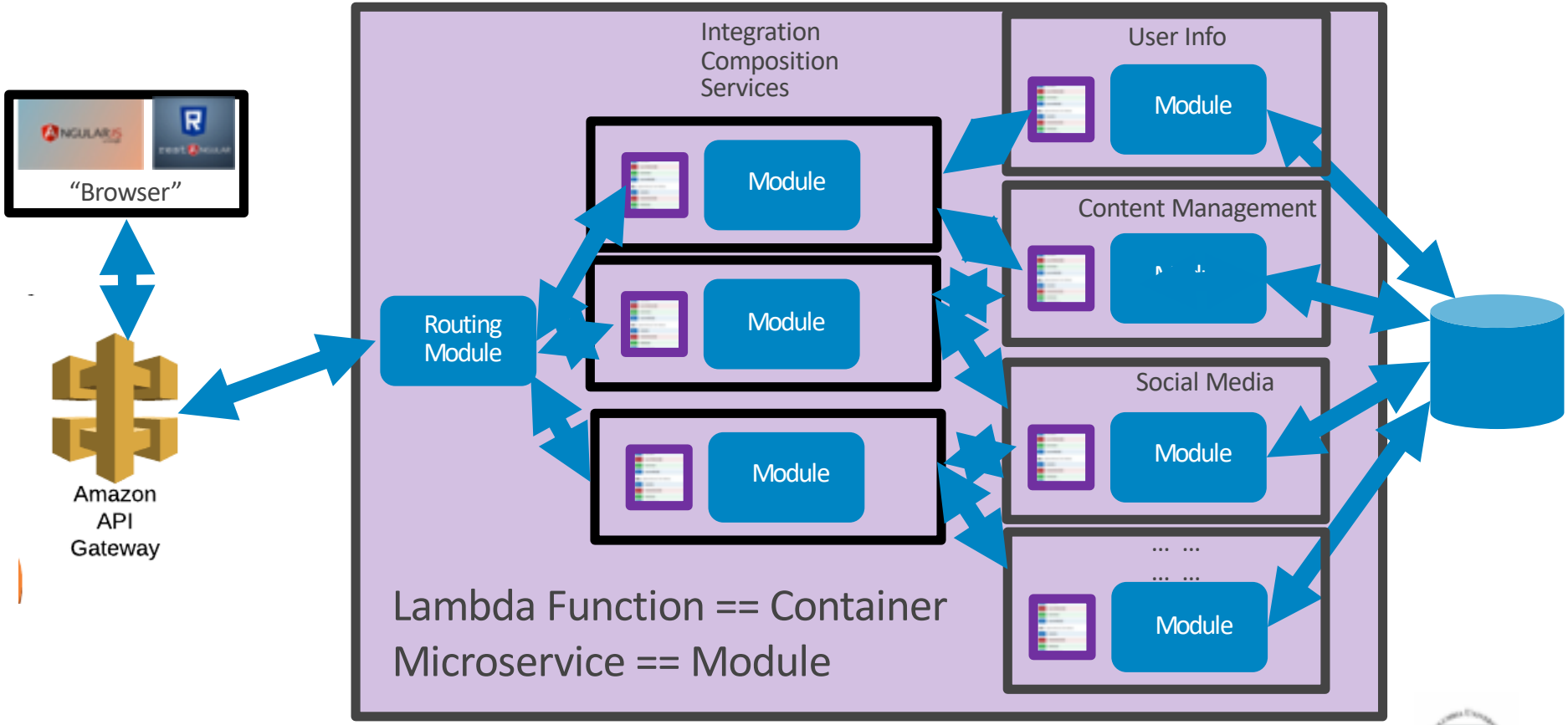
An Attempt to Draw an Architecture Diagram



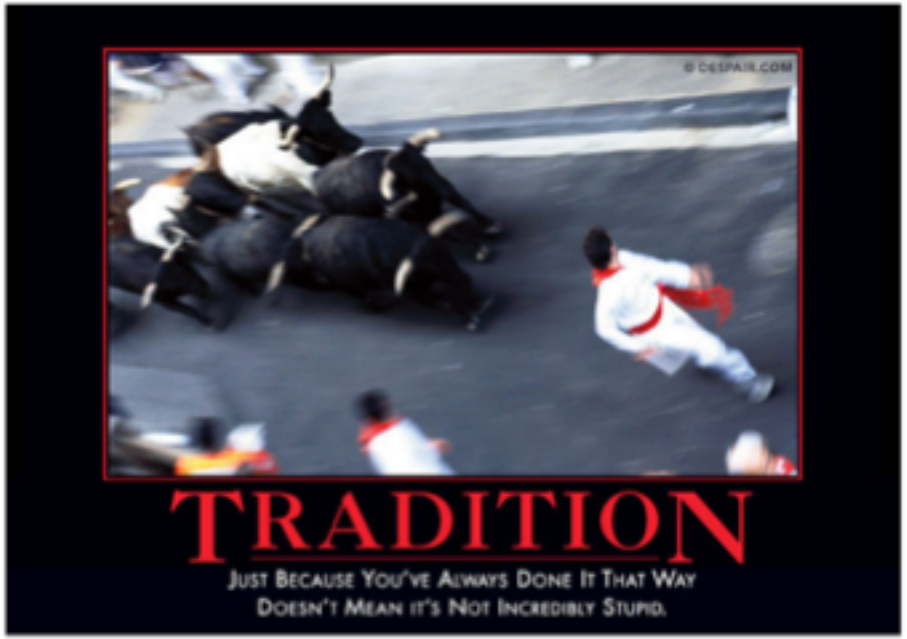
Some numbers:

- 2 EC2 instances.
- 21 RDS instances.
- 152 Lambda Functions.
- 82 S3 buckets.
- 1 Neo4J DB.
- 17 CloudFront Distributions
- 52 API GW APIs.

When you do know know what to do, do what you now how to do.

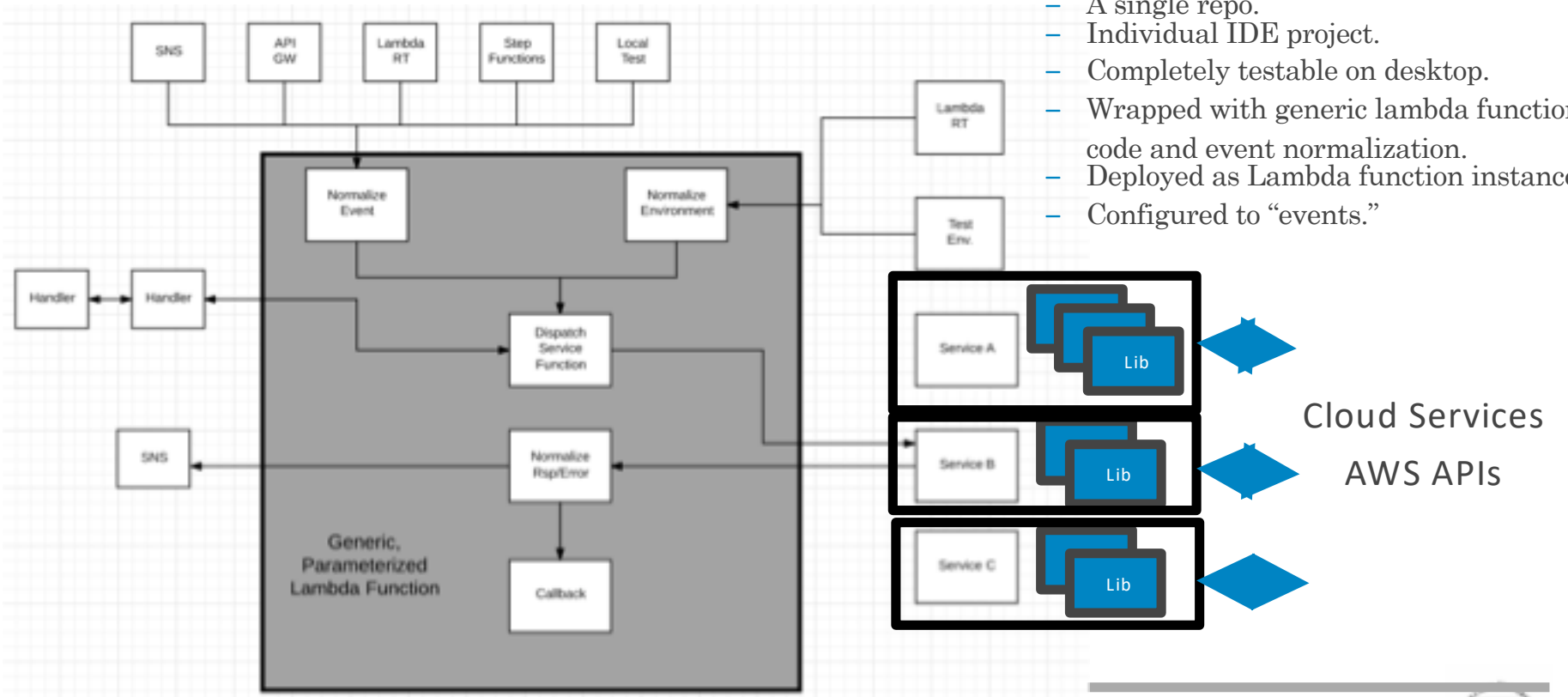


It Worked, but …



Current Approach

- Write code as functions: msg = f(msg)
- Micro-service is
  - A set of functions.
  - A single repo.
  - Individual IDE project.
  - Completely testable on desktop.
  - Wrapped with generic lambda function code and event normalization.
  - Deployed as Lambda function instance.
  - Configured to "events."



Some Perspective

“Serverless Computing (Serverless) is emerging as a new and compelling paradigm for the deployment of cloud applications, and is enabled by the recent shift of enterprise application architectures to containers and micro services.”

- “Enabled by …” Yes.
  - Increased focus on event-based programming.
  - Small(er), focused pieces of code.
  - Polyglot programming and persistence.
  - Loose coupling and independent change.
- I have come to think of severless being like an old-style pub/sub topic space or programmable wiki/web.
  - The application is a URI space.
  - Events happen on URLs → map to a function.
  - You may choose to have a URL subtree map to a microservice.
  - The runtime is the OS process model.

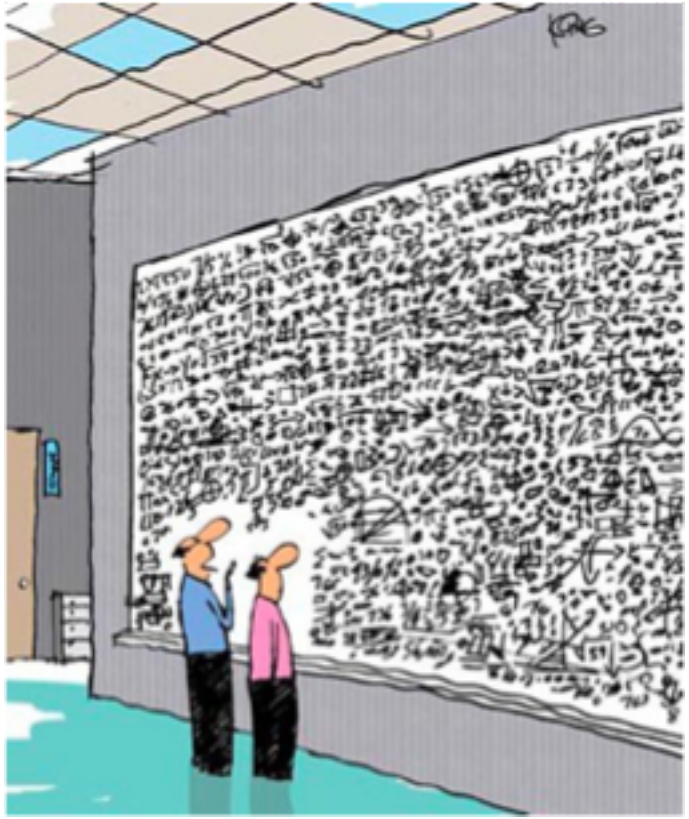
"Serverless architectures offer different tradeoffs in terms of control, cost, and flexibility. For example, this requires developers to more carefully consider the resources used by their code (time to execute, memory used, etc.) when modularizing their applications. This is in contrast to concerns around latency, scalability, and elasticity, which is where significant development effort has traditionally been spent when building cloud services."

- Scalability, elasticity, latency: Serverless
  - Does not change development wrt to scalability, elasticity, latency.
  - Vastly simplifies the environment definition, config, operation, etc.
- "Consider resources … when modularizing application:"
  - Hmm. I almost never consider this. The function runs and isolation and the function consumes what the function needs.
  - API calls effect cost:
    - The multi-programming level is approximately 1, unlike an app sever.
    - You pay for execution duration when the thread is waiting.
  - You think of your application as a set of steps/commands.
    - Each step is one serverless function execution/invocation.
    - You should probably do this anyway for long-running, complex tasks.

"A rich ecosystem of services built into the platform is typically easier to compose and would offer better performance. However, composing external services may be unavoidable, and in such cases, many of the benefits of serverless disappear, including performance and availability guarantees."

- The cloud and its APIs are the platform.
- Even in J2EE or .NET, you should assume that
  - The service API is "local."
  - The service implementation is remote.
- Ecosystem of services:
  - Technical services might be built into the platform/server.
  - Business services are almost always an API call.
- Availability is an issue. A bigger issue is understanding what needs to be undone or redone.

Teaching this Stuff …



Taught serverless/aaS/Microservices 4 semesters to seniors, MS and Ph.D. students.

- Students lack understanding of basic concepts in SW engineering, e.g.
  - Design patterns.
  - Pub/Sub, event-driven-architecture.
  - Queueing, orchestration, workflow.
  - Asynchrony
- There is a huge startup overhead to get a basic "HelloWorld" to work.
  - Lambda.
  - API GW.
  - Correctly setting policies.
  - etc.