PATTERNS IN SERVERLESS PROGRAMMING
WHAT IS A ‘SERVERLESS’?

• Highly available in seconds or milliseconds
• Automatically scales to demand
• Shuts off when it’s not being used
• Platforms
  • AWS Lambda Functions
  • Google Cloud Functions
  • Azure Functions
  • Various on-premises platforms (OpenFaaS, Firecracker, Knative)
WHAT IS A ‘SERVERLESS’?

• More than just functions
• Different approach toward software architecture
• Componentization in the Large for decomposition of a problem.
• Evolutionary/Agile Architecture
• Unit of Scale is the Job or Task
• Polyglot
WHAT IS A ‘SERVERLESS’?

• Highly simplified internal structure
• Code focused on business logic, not plumbing or request routing
• Generally less code, more reliance on external services
• Should I worry about Vendor Lock-In?
WHAT IS A ‘SERVERLESS’?

• Should I worry about Vendor Lock-In?
  • Equivalent services at most cloud providers
  • Cost of abstractions to avoid it
  • Speed of development vs Speed of organizational administration
  • Lock-in to consumed services is a bigger problem
• TL;DR
  • Probably not
Components are deployed.
Features are released.
Applications are built by routing.
-- Neal Ford
WHAT TO DO WITH THEM

• Event-driven Data Processing
• Serverless Web Apps
• Serverless Application Ecosystems
• Decomposing Monoliths
EVENT-DRIVEN DATA PROCESSING

• Requirements
  • Accept an uploaded file
  • Read the file, and for each line
    • Do something, such as
      • Insert a record into a database
      • Make a REST API call
      • Send an email
EVENT-DRIVEN DATA PROCESSING

- Trigger
  - Upload file to S3

- Lambda Function
  - Reads file
  - Does something with the data
    - Inserts it into a database
    - Uses it to make REST calls to another API
EVENT-DRIVEN DATA PROCESSING

POST request from On Premises System (e.g. curl, script, or application) triggers an S3 Bucket. The S3 Bucket sends a request to Lambda, which interacts with MySQL and sends an email.
SERVERLESS WEB APPS

- **POST**
  - `{ "name":"An Item"} -> POST /items -> { "id":1, "name":"An Item"}

- **GET**
  - GET /items -> [{ "id":1, "name":"An Item"}]
  - GET /items/1 -> { "id":1, "name":"An Item"}

- **PUT**
  - `{ "id":1, "name":"Modified Item"} -> PUT /items/1 -> { "id":1, "name":"Modified Item"}

- **DELETE**
  - DELETE /items/1
SERVERLESS WEB APPS:

**The Internet**
- Cloudfront
  - /items
- S3
  - /items
  - (default)

**API Gateway**
- GET /items
- GET /items/{itemId}
- POST /items
- PUT /items/{itemId}
- DELETE /items/{itemId}

**Lambda**
- $ for Executions
- $ for Traffic
- $ for Traffic & Storage
- Dynamo DB
  - Items Table

**S3**
- $ for Storage

$ for Storage

$ for Traffic
SERVERLESS WEB APPS

Day to Day Traffic on Course Search & Enroll platform
SERVERLESS WEB APPS

Day to Day Traffic on Course Search & Enroll platform
SERVERLESS WEB APPS

Day to Day Traffic on Course Search & Enroll platform
SERVERLESS WEB APPS

Day to Day Traffic on Course Search & Enroll platform
SERVERLESS WEB APPS

The Internet

Cloudfront

S3

$ for Traffic

$ for Storage

API Gateway

GET /items
GET /items/{itemId}

POST /items

PUT /items/{itemId}

DELETE /items/{itemId}

Lambda

Dynamo DB
Items Table

$ for Traffic

$ for Executions

$ for Traffic & Storage

$ for Executions

$ for Executions

$ for Traffic

$ for Traffic
The diagram depicts a serverless application ecosystem with interconnected components:

- **The Internet** as the entry point.
- **API Gateway** connected to **Lambda**.
- **Lambda** interacts with **Simple Notification Service** and **Simple Queue Service**.
- **Simple Notification Service** routes messages to **Topic A**, **Topic B**, and **Topic C**.
- **Simple Queue Service** manages queues **Queue A**, **Queue B1**, and **Queue B2**.
- Messages from **Topic C** are sent to **Queue B2**.
- **Queue B2** triggers **Send Email** event.
- **Lambda A**, **Lambda B1**, and **Lambda B2** are connected to their respective queues and topics, allowing for dynamic and scalable processing.

This ecosystem showcases a flexible, event-driven architecture where incoming requests are processed by functions (Lambdas) in a cost-effective and scalable manner.
Contrived Use Case Requirements

- Unauthenticated users allowed to upload CSV files
- Data in files must be validated
- ‘Real time’ aggregation chart shown on website
- Notifications sent to Slack or Teams with usage graphs
- Must be able to handle all current and prospective students concurrently (upwards of 100,000 users)
- 24/7/365 uptime required due to global outreach
- Deadline is in three months
- Stakeholders planning to send out mass emails with the site URL

- Good Luck!
1. Upload a file directly to S3 from an S3-hosted webpage
   1. Upload form has cryptographically signed value for authentication
   2. User can be sent to a specified URL upon success and/or error
   3. Uses S3 multi-part upload (for huge files)
   4. Bucket triggers SQS notification when file uploaded

2. Lambda (processor)
   1. Triggered by SQS message
   2. Processes uploaded file into consistent validated format
   3. Stores in a new bucket

3. Lambda (reporter)
   1. Listens S3 additions to processed file bucket
   2. Queries Amazon Athena defined on S3 bucket
   3. Writes results to S3

4. Fargate Docker Task (chartTask)
   1. Runs on a scheduled Cloudwatch Event
   2. Sends notification to Slack (or Teams) with chart
1. Upload a file directly to S3 from an S3-hosted webpage
   1. Upload form has cryptographically signed value for authentication
   2. User can be sent to a specified URL upon success and/or error
   3. Uses S3 multi-part upload (for huge files)
   4. Bucket triggers SQS notification when file uploaded

2. Lambda (processor)
   1. Triggered by SQS message
   2. Processes uploaded file into consistent validated format
   3. Stores in a new bucket

3. Lambda (reporter)
   1. Listens S3 additions to processed file bucket
   2. Queries Amazon Athena defined on S3 bucket
   3. Writes results to S3

4. Fargate Docker Task (chartTask)
   1. Runs on a scheduled Cloudwatch Event
   2. Sends notification to Slack (or Teams) with chart
SERVERLESS APPLICATION ECOSYSTEMS

• Upload a file directly to S3 from an S3-hosted webpage
  • Upload form has cryptographically signed value for authentication
  • User can be sent to a specified URL upon success and/or error
  • Uses S3 multi-part upload (for huge files)
  • Bucket triggers SQS notification when file uploaded
SERVERLESS APPLICATION ECOSYSTEMS

<form action="https://psp-uploads.s3.amazonaws.com" method="post" enctype="multipart/form-data">
  <input type="hidden" name="Content-Type" value="text/csv">
  <input type="hidden" name="AWSAccessKeyId" value="AKIAFFDSF4FEDD44FB">
  <input type="hidden" name="acl" value="private">
  <input type="hidden" name="success_action_redirect" value="http://psp-website.s3.amazonaws.com/index.html">
  <input type="hidden" name="policy" value="eyJXQ1oiJwc3Atv8sb2FkcyJadspfawecHjpdmF0ZSj9Hlsic3VjVvZc1hY3Rpbav34tr3awetgsdfgNwLx1YnNpdeGuczMuYXZhem99uYXzLmNvbS9pbnRleC5odGlsIn0sWyJzdGFydhMTd2IIsiRDb250Zn8LRsGUiLCJ0ZXhhbDlJLSy0v29uGvudC15w35w4twF4cmFuZ2UlDAcMTA0ODU3N11dfQ==">
  <input type="hidden" name="signature" value="vhj8iAn5wqm2TPvLqwy5GySZM="">
  <br>
  File to upload to S3:
  <input name="file" type="file">
  <br>
  <input type="submit" value="Upload File to S3" />
</form>
Lambda (processor, 65 lines of Javascript)

- Triggered by SQS message
  - S3ObjectCreated, s3://psp-uploads/person-1342342342242.csv
- Processes uploaded file into consistent validated format
  - Reads and writes from S3 streams to handle large files
    - Never reads entire file into memory
  - Discards and logs rows that fail validation
- Stores in a new bucket
  - s3://psp-processed-files/person-1342342342242.csv.json
- Triggers next Lambda function
SERVERLESS APPLICATION ECOSYSTEMS

• Lambda (reporter, 95 lines of Javascript)
  • Triggered when an object is added to the psp-processed-files S3 bucket
    • ObjectCreated, psp-uploads/person-1342342342422242.csv.json
  • Queries an Amazon Athena table defined over the JSON data in the S3 bucket
    • SELECT count(Id) AS Count, Academic_level
      FROM people
      GROUP BY Academic_level
      ORDER BY Count desc, Academic_level
  • Generates json data for a graph, and uploads it to the psp-website bucket
    • s3://psp-website/chartData.json
“Schema on Read” — Athena uses Hive DDL to define schema

```
CREATE EXTERNAL TABLE IF NOT EXISTS people (  
  Id STRING,  
  First_Name STRING,  
  Last_Name STRING,  
  Email STRING,  
  Address STRING,  
  City STRING,  
  State STRING,  
  Academic_Level STRING,  
  Zip_Code STRING,  
  IP_Address STRING,  
) ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe' STORED AS TEXTFILE  
LOCATION 's3://psp-processed-data/
```

• Web console, JDBC driver available
SERVERLESS APPLICATION ECOSYSTEMS

• Fargate Docker Task (chartTask, 33 lines of Bash code, 4 command line tools, 1 Dockerfile of 23 lines, 22 lines of ECS Task configuration in JSON)
  • Runs on a scheduled Cloudwatch Event
  • Uses a selection of shell tools to create an SVG file for slack
  • Converts webpage with embedded SVG to PDF using a wkhtmltopdf command line tool
  • Sends notification to Slack (or Teams) with chart PDF
DECOMPOSING MONOLITHS

The Internet → Application Load Balancer / (default) → EC2
DECOMPOSING MONOLITHS

The Internet -> Application Load Balancer

EC2

Lambda Functions

/(default)

/new-feature-endpoint
DECOMPOSING MONOLITHS

The Internet

Application
Load Balancer

EC2

Lambda Functions

ECS Cluster

Container

Container

Container
DECOMPOSING MONOLITHS

• AWS Application Load Balancer can route specified requests to AWS Lambda functions
  - Path
  - Port
  - Hostname

• Routing can be configured like this:
  - /newFeatureEndpoint => AWS Lambda or arbitrary Docker Container cluster (ECS)
  - /brokenFeatureReplacement => AWS Lambda or arbitrary Docker Containers cluster (ECS)
  - /*(default) => Monolithic Application cluster : 8080

• Once the monolith has nothing else to do, you can turn them off entirely
DECOMPOSING MONOLITHS

Doing this from Apache

```xml
# mysite.wisc.edu.conf

<VirtualHost :443>
    ServerName mysite.wisc.edu
    ...
    ...
    ProxyPass / http://myoldservlet.doit.wisc.edu:8080
</VirtualHost>

<Location /api/myoldendpoint>
    ProxyPass https://2rkvg3sf0i.execute-api.us-east-1.amazonaws.com/prod/newendpoint
    RequestHeader set x-api-key "gs0m21 dfsdfsa2df0jz0uantw4r3wEF15623VQq4VI"
</Location>
</VirtualHost>
```
NOT A HAIKU

Components are deployed.
Features are released.
Applications are built by routing.
-- Neal Ford
EXAMPLE AND THE SOURCE CODE

- http://www.mypresentationdemo.com

More questions? Future questions?

Brian Hill
brian.hill@wisc.edu
Enterprise Integrations
Application Integration Services
Division of Information Technology
UW Madison