Serverless

My experience deploying a serverless application on AWS

Brendan Dagys (2023-07-28)

What we'll cover

0 - *What is serverless?*

1 - Pros and cons

2 - AWS services review: CloudFormation, Lambda, API Gateway, DynamoDB

3 - Building a serverless project with SAM: Background and motivation

4 - Building a serverless project with SAM: Architectural components and approach

5 - Building a serverless project with SAM: Reflection

6 - Demo

7 - Case study

What is serverless?

- A cloud computing execution model where the provider allocates machine resources on demand and manages the servers for customers
- Takes care of maintenance, fault tolerance, and scaling
- Ideally allows developers to focus on writing code
- Includes databases and workflow orchestration
- **Common use cases**: asynchronous tasks, APIs, CI/CD operations, file conversions, data analytics

Advantages

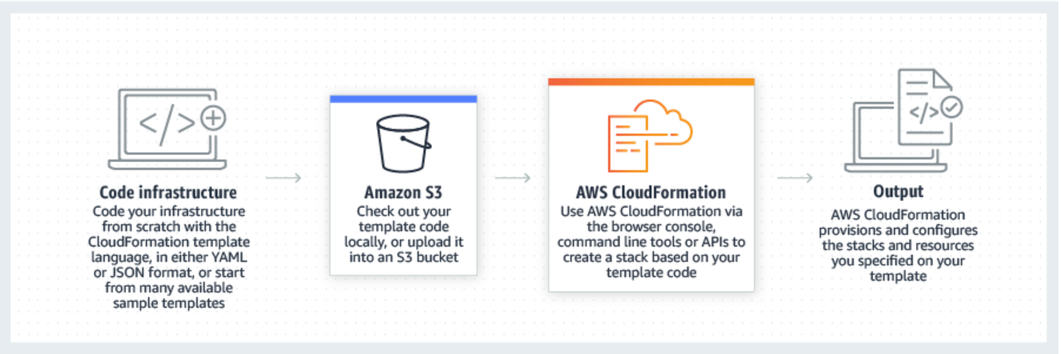
- Low cost
- Availability and fault tolerance
- Scalability
- Productivity

Disadvantages

- Loss of control
- Performance
- Vendor lock-in
- Security and privacy

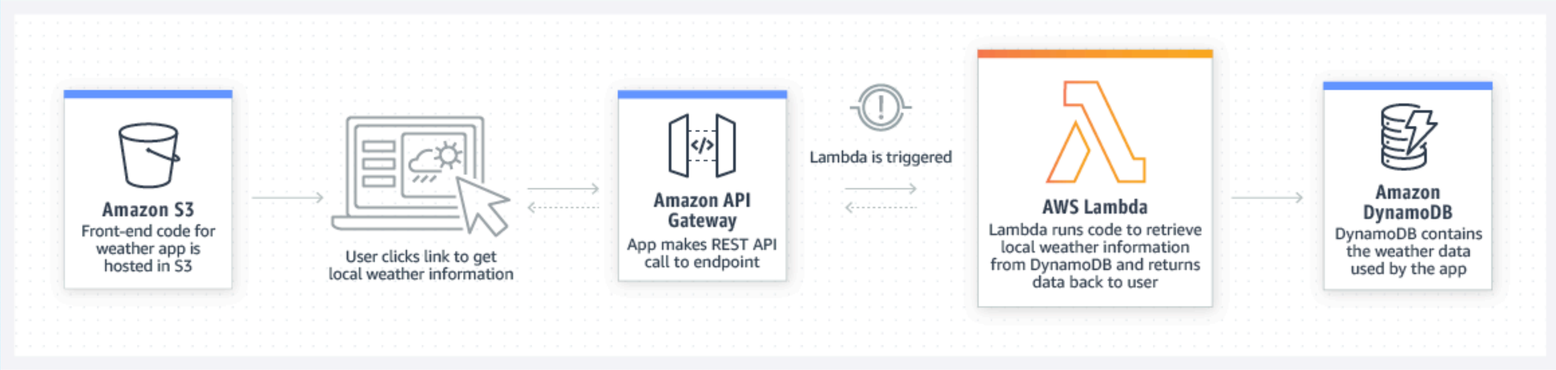
AWS CloudFormation

- Declare your AWS infrastructure as code (YAML or JSON) in templates
  - Deploy stacks in multiple accounts and/or regions
- **Benefits**: Version control, cost savings, automation, productivity
- Reference resources/outputs from other stacks with nested stacks and cross-stack references



https://aws.amazon.com/cloudformation/

AWS Lambda



https://aws.amazon.com/lambda/

- **Compute** service for running code without provisioning or managing servers
  - Scaling, logging, monitoring are all handled for you
- Configure your memory, and the CPU power scales proportionally
- Pay only for the compute time that you consume

$0.0000133334 for every GB-second $0.20 per 1M requests

- **Use cases**: File processing, stream processing, IoT backends, mobile backends
- Function timeout after 1 - 900 seconds
- Supports versioning, environment variables, concurrency controls, and integration with many AWS services

AWS API Gateway

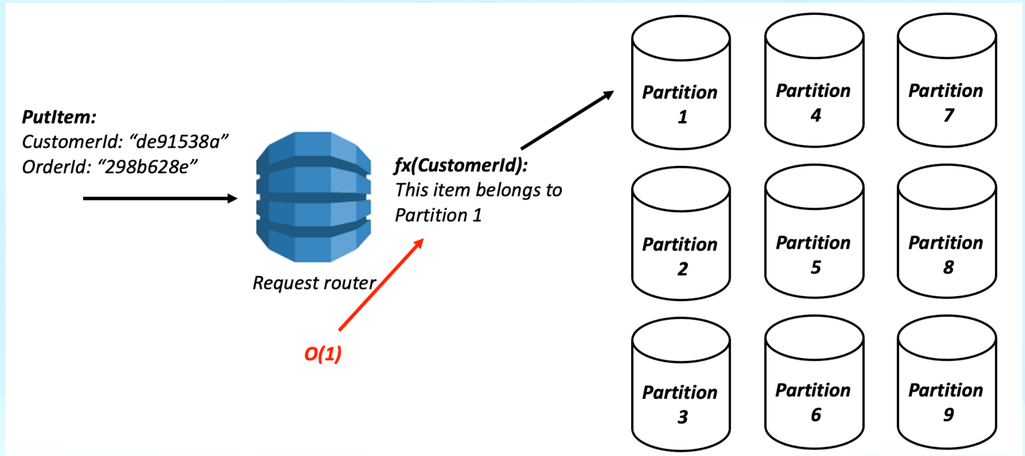
https://aws.amazon.com/api-gateway/

- Fully-managed service for creating, publishing, and maintaining secure APIs at any scale
- REST APIs, WebSocket APIs
- Set up authorization and multiple stages for your different environments
- Can front many AWS services such as Lambda, EC2, or DynamoDB as a proxy

DynamoDB

- Fast, flexible **key-value NoSQL database** service
- Single-digit millisecond performance
- Nearly-unlimited throughput and storage
- Scales up and down to fit your needs (‘on-demand’ mode)
- **Schema-less**: only requires a primary key
- Automatic multi-region replication
- Billed for read capacity units (RCU), write capacity units (WCU), and storage
- **DynamoDB Streams** allow you to react to table modifications with Lambda functions

DynamoDB
Schema Design



https://www.alexdebrie.com/posts/dynamodb-partitions/

- **Tables**… of **items**… composed of **attributes**
- Data types:
  - Scalar: number, string, binary, Boolean, null
  - Document: list, map
  - Set: string set, number set, binary set
- DynamoDB uses primary keys to uniquely identify each item in a table (mandatory: partition key, optional: sort key)
- **Secondary indexes** (global [GSI] and local [LSI]) can be used to allow query flexibility
- **Read operations**: GetItem (+ batch), Query, Scan
- **Write operations**: PutItem (+ batch)

DynamoDB
Schema Design

Entity	PK	SK	Index-PK	Index-SK
Guest	GUEST-*<email>*	GUEST-INFO	GUEST	*<name>*
BookedDate	DATE	*<ymd>*	DATE	*<state>*
Inquiry	INQUIRY	*<created>*	INQUIRY	*<state>*

- **Example**: Cottage rental
- Three entities: **Guest**, **Booked Date**, **Inquiry**
- Access patterns:
  - **Guest**: Get guest by email, name | get all guests
  - **Booked Date**: Get dates by date, state | get all dates
  - **Inquiry**: Get inquiries by date created, state | get all inquiries

My serverless project

Background and motivation

- Previous projects relied on AWS Elastic Container Service (ECS) to deploy the backend
- This worked well, but was costly for a continuously-running application
- While learning more about AWS, I became privy to their serverless offerings
- I decided to try their Serverless Application Model (SAM) to build and deploy a full-stack application
- Expectations were to learn new technologies, form an opinion about serverless, and save costs

- https://github.com/brendandagys/AWS-Architecture-for-React-TypeScript-Deployment-with-S3-Fargate

Cost of my previous approach

Edit AWS Fargate Info

Number of tasks or pods
Enter the number of tasks or pods running for your application
1 | per month

Average duration
Enter the time period for which your tasks or pods are running. Pricing is per second with a 1-minute minimum (Linux) and a 15-minute minimum (Windows). Duration is calculated from the time you start to download your container image (docker pull) until the Task or Pod terminates, rounded up to the nearest second.
30 | days

Amount of vCPU allocated
Enter the amount between 0.25 vCPU and 16 vCPU
0.25

vCPU selected supports memory values 0.5 GB, 1 GB, or 2 GB

Amount of memory allocated.
0.5 GB

Amount of ephemeral storage allocated for Amazon ECS
Enter the amount between 20 GB and 200 GB. The first 20 GB are at no additional charge, you only pay for any additional storage that you configure for the Task.
20 | GB

▶ Show calculations

Total Upfront cost: 0.00 USD
Total Monthly cost: 8.89 USD

Show Details ▼

Cancel | Update

My serverless project

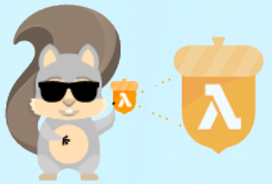
Background and motivation

- Previous projects relied on AWS Elastic Container Service (ECS) to deploy the backend
- This worked well, but was costly for a continuously-running application
- While learning more about AWS, I became privy to their serverless offerings
- I decided to try their Serverless Application Model (SAM) to build and deploy a full-stack application
- Expectations were to learn new technologies, form an opinion about serverless, and save costs

- https://github.com/brendandagys/AWS-Architecture-for-React-TypeScript-Deployment-with-S3-Fargate

My serverless project

Architectural components and approach

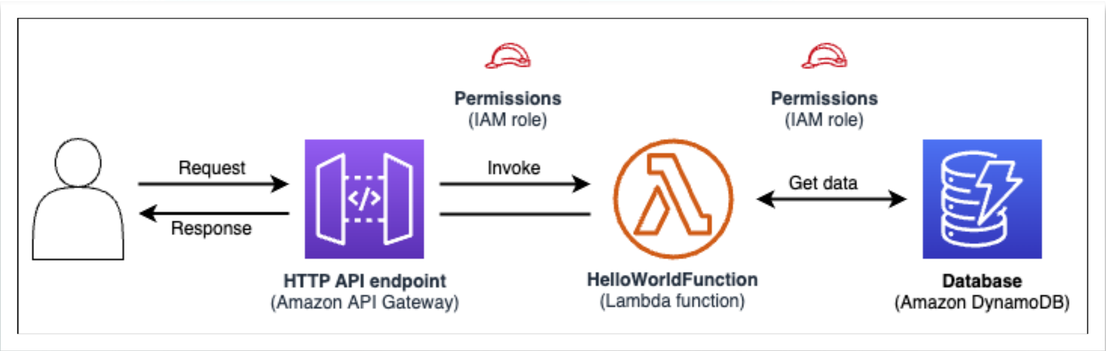


"SAM the Squirrel"

- **AWS SAM**: an open-source framework for building serverless applications
- **Resources**: API Gateway, DynamoDB, Lambda functions, DNS records, S3 bucket
- CloudFormation
- Rust 
- Learning how to design a DynamoDB database

My serverless project
AWS SAM

- Define your serverless application in YAML, and specify and other resources
- SAM transforms/expands into CloudFormation!
- Interact with it using the SAM CLI



https://docs.aws.amazon.com/serverless-application-model/latest/developerguide

```yaml
AWSTemplateFormatVersion: 2010-09-09
Transform: AWS::Serverless-2016-10-31
Resources:
  getAllItemsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: src/get-all-items.getAllItemsHandler
      Runtime: nodejs12.x
      Events:
        Api:
          Type: HttpApi
          Properties:
            Path: /
            Method: GET
    Connectors:
      MyConn:
        Properties:
        Destination:
          Id: SampleTable
          Permissions:
            - Read
  SampleTable:
    Type: AWS::Serverless::SimpleTable
```

SAM can:

- Easily start a new project
- Build the application for deployment
- Debug and test
- Deploy with a single command
- Configure CI/CD pipelines
- Monitor your application
- Sync local changes to the cloud as you develop

My serverless project
SAM CLI

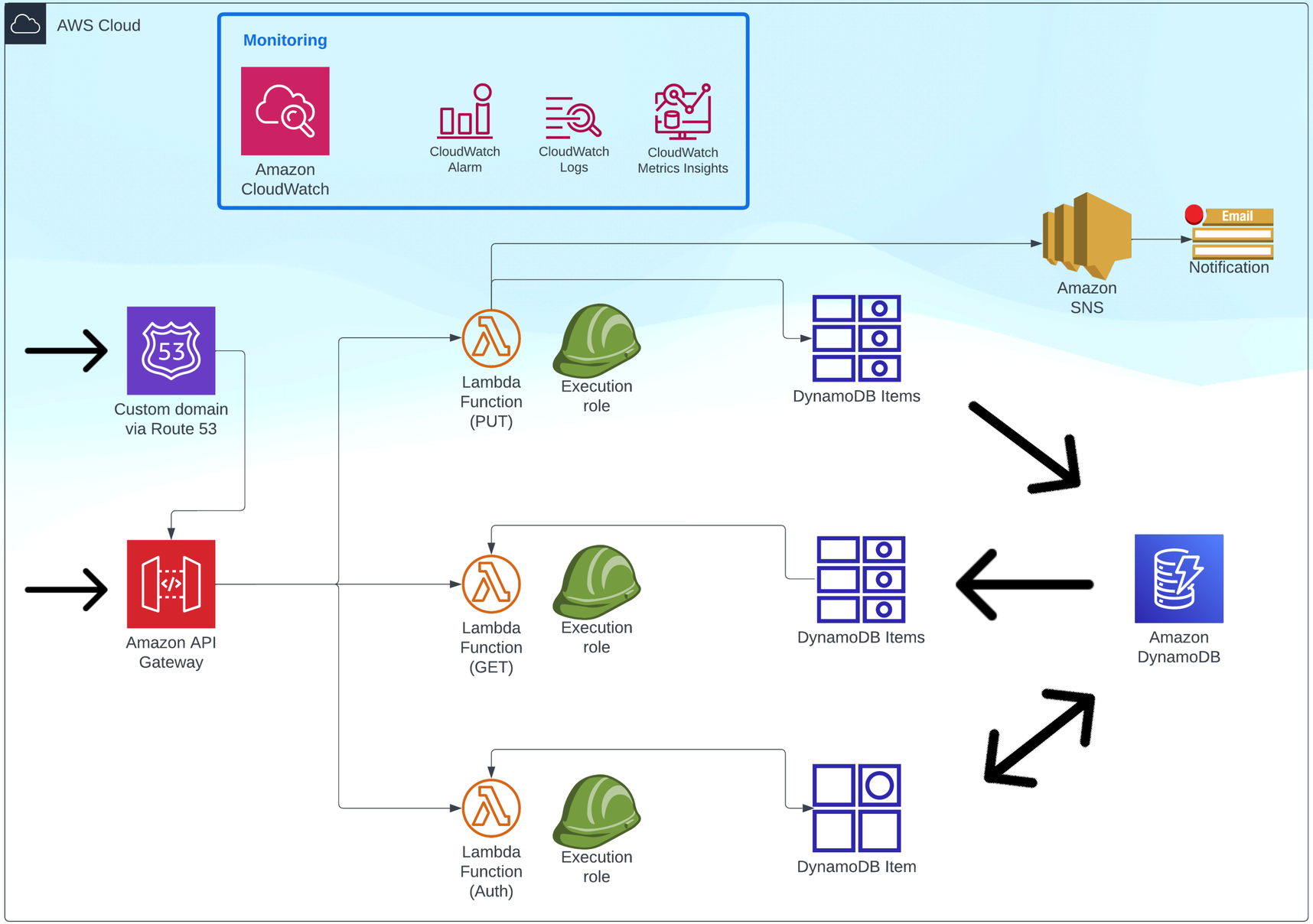
```
Successfully created/updated stack - SQLShackDemoStack2 in None


C:\temp\my-serverless-app>sam deploy --template-file deploy.yaml --stack-name SQLShackDemoStack --capabilities CAPABILITY_IAM

        Deploying with following values
        ===============================
        Stack name                 : SQLShackDemoStack
        Region                     : None
        Confirm changeset          : False
        Deployment s3 bucket       : None
        Capabilities               : ["CAPABILITY_IAM"]
        Parameter overrides        : {}

Initiating deployment
=====================
GetS3BucketsFunction may not have authorization defined.

Waiting for changeset to be created..

CloudFormation stack changeset
---------------------------------------------------------------------------------------------------------------------------------------------------------------
Operation                                                LogicalResourceId                                        ResourceType
---------------------------------------------------------------------------------------------------------------------------------------------------------------
+ Add                                                    GetS3BucketsFunctionGetS3BucketsAPIPermissionProd        AWS::Lambda::Permission
+ Add                                                    GetS3BucketsFunction                                     AWS::Lambda::Function
+ Add                                                    ServerlessRestApiDeployment7753d8019e                    AWS::ApiGateway::Deployment
+ Add                                                    ServerlessRestApiProdStage                               AWS::ApiGateway::Stage
+ Add                                                    ServerlessRestApi                                        AWS::ApiGateway::RestApi
---------------------------------------------------------------------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:ap-south-1:890277245818:changeSet/samcli-deploy1596304966/21788a46-77ce-4ae9-a4f6-de511675ebde


2020-08-01 23:32:52 - Waiting for stack create/update to complete
```

https://aws.amazon.com/blogs/compute/a-simpler-deployment-experience-with-aws-sam-cli/

- **sam init** - Provides options to initialize a new serverless application
- **sam build** - Prepares your application for local use and deployment
- **sam deploy** - Deploys your application to the cloud
- **sam local** - Commands for local interaction with your application
  - **generate-event** - Creates event payload samples that you can use to test locally
  - **invoke** - Initiates a one-time local invocation of a Lambda function
  - **start-api** - Creates a local API in front of your Lambda functions
  - **start-lambda** - Spins up a Lambda function locally for testing via the AWS CLI or Software Development Kit (SDK)
- **sam delete** - Tears down your CloudFormation stack (resources)

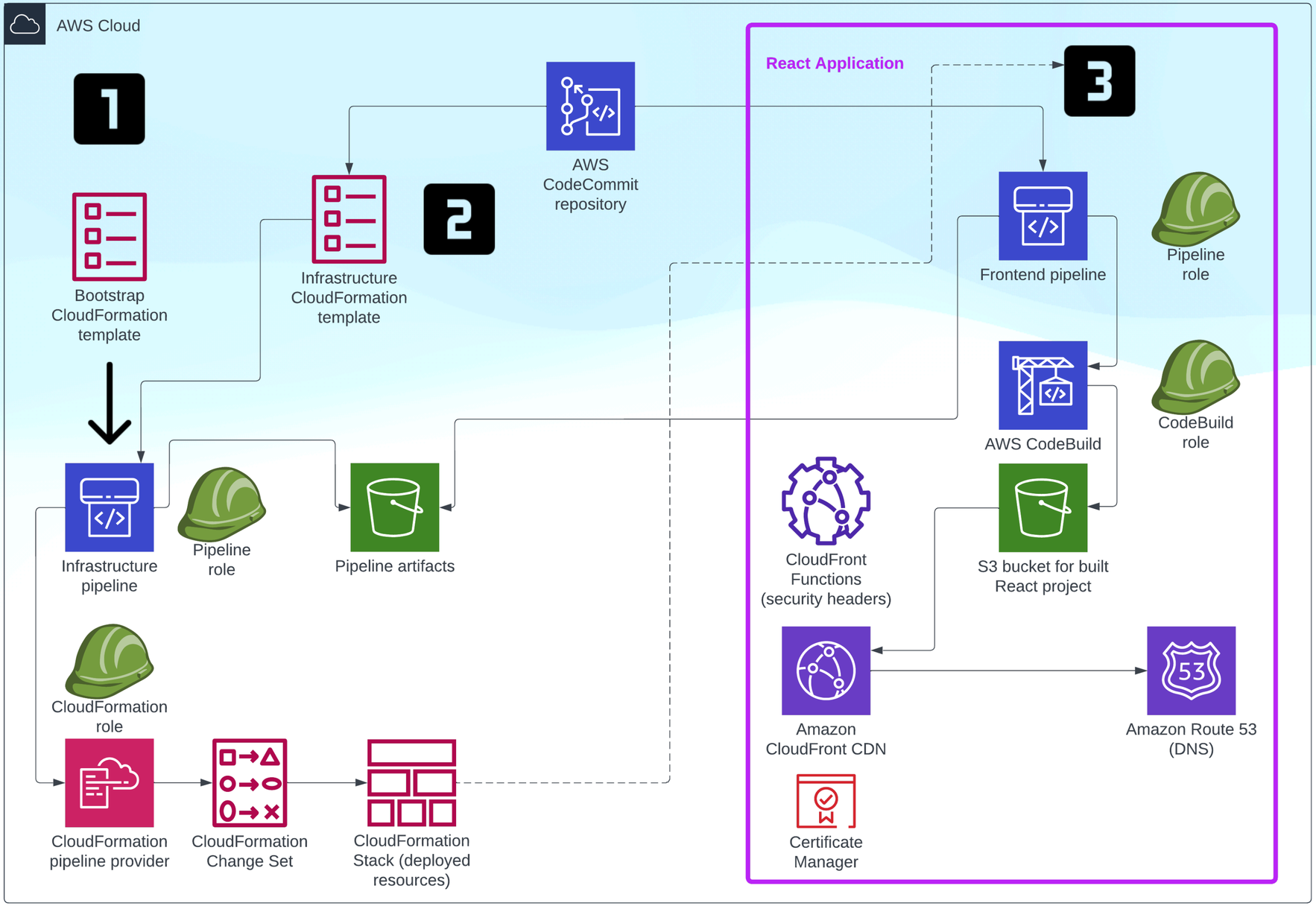
My serverless project

SAM project (back-end)



My serverless project

DevOps and front-end



My serverless project

Reflection

- Function layout takes some getting used to
- Did not notice any cold starts (but could mitigate this)
- Easy development and deployment with the SAM CLI
- Using Rust adds some complexity
- Extremely low cost

My serverless costs

Service	Purpose	Cost (CAD)
CloudWatch	Monitoring	$3.55
Route 53	DNS zone + record	$0.67
CodeBuild	Build pipeline for app	$0.33
S3	Application storage (static)	$0.17
API Gateway	API	$0.01
CloudFront	CDN	$0
CodeCommit	Git repository	$0
DynamoDB	Database	$0
Lambda	Compute	$0
SNS	Email notifications	$0

CloudWatch		**USD 2.66**
Canada (Central)		USD 0.00
US East (N. Virginia)		USD 2.66
Amazon CloudWatch		USD 2.66
$0.00 per alarm metric month - first 10 alarm metrics	10 Alarms	USD 0.00
$0.00 per request - first 1,000,000 requests	5,004 Requests	USD 0.00
$0.01 per 1,000 metrics requested using GetMetricData API - US E	2,282 Metrics	USD 0.02
$0.10 per alarm metric month (standard resolution) - US East (No	26.395 Alarms	USD 2.64

DynamoDB		**USD 0.00**
US East (N. Virginia)		USD 0.00
Amazon DynamoDB PayPerRequestThroughput		USD 0.00
$0.25 per million read request units (N. Virginia)	3,221.5 ReadRequestUnits	USD 0.00
$1.25 per million write request units (N. Virginia)	2,146 WriteRequestUnits	USD 0.00
Amazon DynamoDB TimedStorage-ByteHrs		USD 0.00
$0.00 per GB-Month of storage for first 25 free GB-Months	0 GB-Mo	USD 0.00
Amazon DynamoDB USE1-ExportDataSize-Bytes		USD 0.00
$0.10000 per GB of data exported in US East (N. Virginia)	0 GB	USD 0.00
Amazon DynamoDB USE1-TimedPITRStorage-ByteHrs		USD 0.00
$0.20 per GB-Month of storage used for DynamoDB PITR backup	0 GB-Mo	USD 0.00

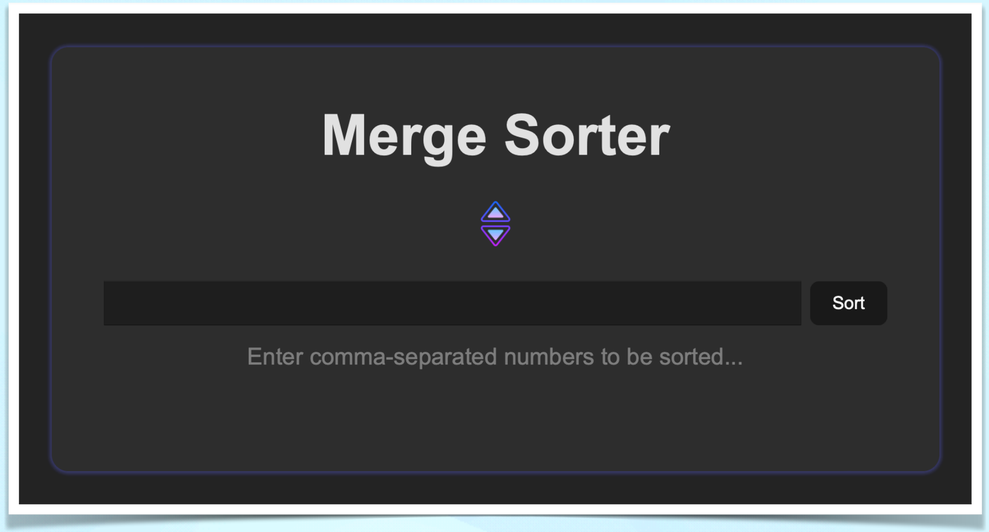
Simple Storage Service		**USD 0.13**
Canada (Central)		USD 0.01
US East (N. Virginia)		USD 0.12
Amazon Simple Storage Service Requests-Tier1		USD 0.02
$0.005 per 1,000 PUT, COPY, POST, or LIST requests	4,171 Requests	USD 0.02
Amazon Simple Storage Service Requests-Tier2		USD 0.01
$0.004 per 10,000 GET and all other requests	34,309 Requests	USD 0.01
Amazon Simple Storage Service TimedStorage-Byte		USD 0.09
$0.023 per GB - first 50 TB / month of storage used	3.791 GB-Mo	USD 0.09

Lambda		**USD 0.00**
US East (N. Virginia)		USD 0.00
AWS Lambda Lambda-GB-Second-ARM		USD 0.00
AWS Lambda - Compute Free Tier for ARM - 400,000 GB-Seconds - US East (N. Virginia)	36.384 Lambda-GB-Second	USD 0.00
AWS Lambda Request-ARM		USD 0.00
AWS Lambda - Requests Free Tier for ARM - 1,000,000 Requests - US East (N. Virginia)	936 Requests	USD 0.00

CodeBuild		**USD 0.25**
US East (N. Virginia)		USD 0.25
CodeBuild USE1-Build-Min:Linux:g1.small		USD 0.25
AWS CodeBuild - Build minutes on build.general1.sm	50 minutes	USD 0.25
AWS CodeBuild Free Tier - 100 build minutes on buil	100 minutes	USD 0.00

API Gateway		**USD 0.01**
US East (N. Virginia)		USD 0.01
Amazon API Gateway ApiGatewayRequest		USD 0.01
$3.50/million requests - first 333 million requests/month	1,841 USE1-AmazonApiGateway-Request	USD 0.01

Demo



- We are going to…
  - Initialize a SAM project from scratch from the terminal
  - Understand what SAM creates for us
    - A directory for our application code
    - Infrastructure template
  - Clone/examine a sorting application
  - Build and deploy the project

Case study

Coca-Cola Freestyle

- Services used: Lambda, API Gateway, CloudFront, WAF
- **Benefits**
  - Launched mobile pour app prototype in 1 week
  - Scaled to 10,000 machines in 150 days
  - Collected no data from customers
  - < 1 second latency to pour a drink
  - Uses dispenser data to ease maintenance and aid user selection
- https://aws.amazon.com/solutions/case-studies/coca-cola-freestyle/

Thank you!

- Slides and code are available at https://github.com/brendandagys
- Feel free to sort some numbers at https://sort.brendandagys.com !
- Questions?