USING THE SERVERLESS FRAMEWORK

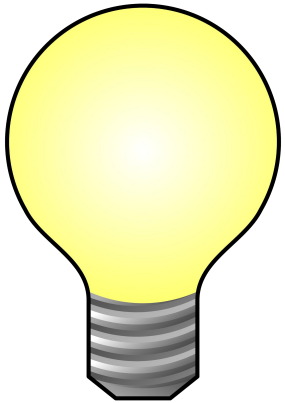
Presented by: Tyler Hendrickson
tyler@shiftgig.com

WHAT IS **“SERVERLESS”?** (The Concept)

- No servers!
- Infrastructure-as-Code
- More time focusing on features that matter to your users

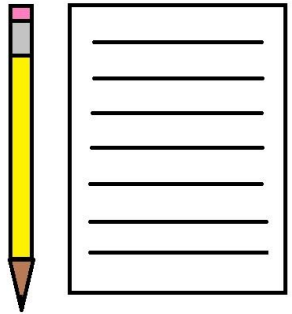
WHY **SERVERLESS?** (The Framework)

- Easy to get started



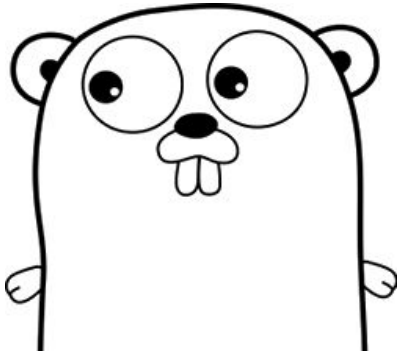
WHY **SERVERLESS?** (The Framework)

- Easy to get started
- Simple templating



WHY **SERVERLESS?** (The Framework)

- Easy to get started
- Simple templating
- Language-agnostic

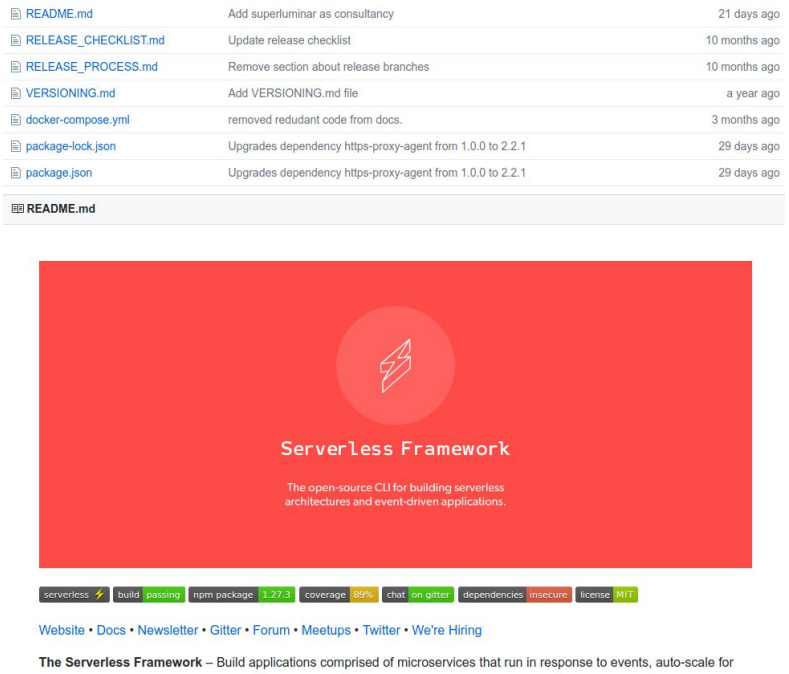


shiftgig™

WHY **SERVERLESS?** (The Framework)

- Easy to get started
- Simple templating
- Language-agnostic

- Active community



shiftgig™

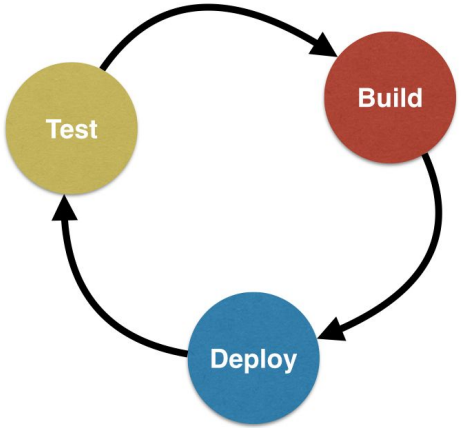
WHY **SERVERLESS?** (The Framework)

- Easy to get started
- Simple templating
- Language-agnostic
- Active community
- Good support for common event sources

shiftgig™

WHY **SERVERLESS?** (The Framework)

- Easy to get started
- Simple templating
- Language-agnostic
- Active community
- Good support for common event sources
- Supports various aspects of the development lifecycle



WHAT ARE MY **ALTERNATIVES?**

SAM

(Serverless Application Model)

?

Chalice

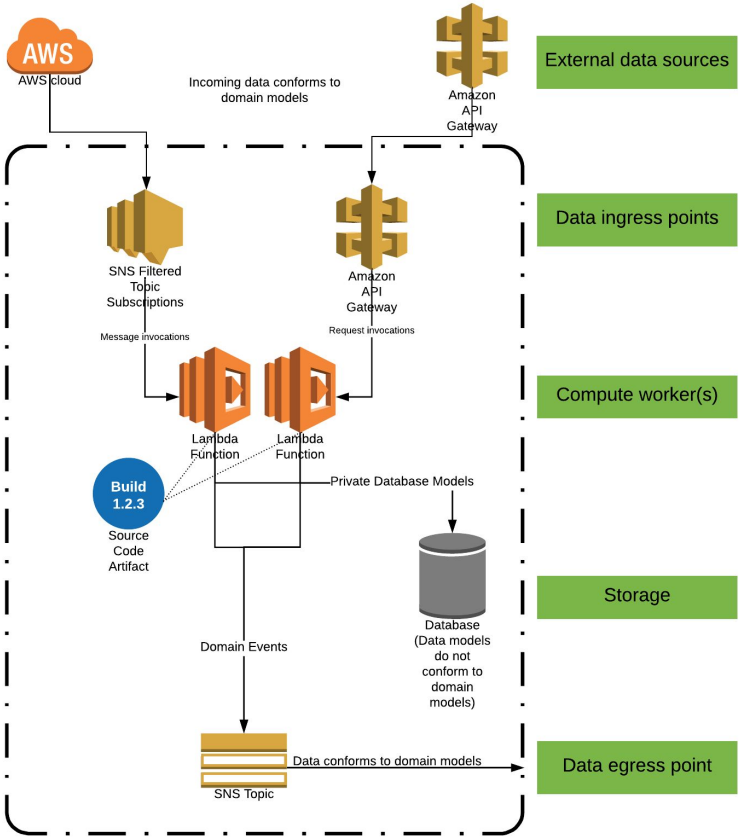
AWS
CloudFormation

shiftgig™

HOW DO WE USE **SERVERLESS AT SHIFTGIG?**

- A few notes about Shiftgig architecture:
  - Microservices!
  - Microservice = Codebase = CloudFormation stack = Serverless application
  - Lambda functions as the main compute resource of choice
  - Avoid HTTP(S)-based communication between microservices and prefer asynchronous messaging instead
  - Microservice stacks are self-contained with little to no overlapping resources
  - Domain models

HOW DO WE USE **SERVERLESS AT SHIFTGIG?**



shiftgig™

WAIT…
ISN’T THIS SUPPOSED TO BE A WORKSHOP?

GETTING STARTED: **PREREQUISITES**

- Development requirements:
  - + Serverless:
    - Node JS
    - NPM
    - `npm install -g serverless`
  - + Development:
    - Python 3.6 (not strictly necessary, but the examples will conform to this)
      - Also: Pipenv or pip + virtualenv
    - Project directory/workspace

GETTING STARTED: **PREREQUISITES**

- AWS authentication
  - + Serverless will authenticate with AWS automatically using your environment-sourced AWS credentials.
  - + Check for `~/.aws/config` and `~/.aws/credentials` files
  - + If you use an AWS config file, make sure you set this environment variable to instruct the Node JS AWS SDK (used internally by Serverless) to load the config file:

```
$ export AWS_SDK_LOAD_CONFIG=1
```

GETTING STARTED: **PREREQUISITES**

- Deployment requirements:
  - AWS account (full permissions are ideal)
  - AWS services we will use
    - Lambda
    - API Gateway
    - DynamoDB
    - CloudWatch
    - CloudFormation
    - S3

STEP 0: **GOALS**

- We will create a simple cloud-native service in AWS as a CloudFormation stack
- Our service will…
  - **Execute custom logic** defined as code run within Lambda functions
  - **Store data** in a DynamoDB table
  - **Provide a web API** with API Gateway
  - **Log execution data** to CloudWatch

- But *what* will my application do?

STEP 1: **CONFIGURE SERVERLESS**

- Navigate to your project directory and set up your development workspace
  - + `$ cd ~/your/projects`

    `$ git clone git@github.com:shiftgig/aws-serverless-workshop.git`
  - + Using Python with Pipenv?

    `$ pipenv shell --python 3.6` to set up your virtual environment
  - + Initialize Node JS in the current directory:

    `$ npm init -f` to create a `package.json` file

    `$ npm install` to install plugin dependencies
  - + Open in your editor: `serverless.yml`

STEP 1: **CONFIGURE SERVERLESS**

```
service: candystore

plugins:
  - serverless-python-requirements  # Python only

custom:
  foo: bar
```

Name your service. This is used to name your AWS resources.

Define array of Serverless plugins

STEP 1: **CONFIGURE SERVERLESS**

```
custom:
  foo: bar
  pythonRequirements: # Python only
    usePipenv: true
    dockerizePip: true
    dockerImage: lambci/lambda:build-python3.6
    dockerSsh: true
```

The `custom` section lets you define arbitrary variables that you can reference in other parts of your Serverless config file.

This section is also often used to configure your installed Serverless plugins.

These `docker*` values tell Serverless to use a Docker image that matches the Lambda Python 3.6 runtime to install dependencies.

STEP 1: **CONFIGURE SERVERLESS**

```
provider:
  name: aws
  stage: ${opt:stage}
  region: ${opt:region}
  logRetentionInDays: 7
  stackTags:
    SERVICE: ${self:service}
  iamRoleStatements:
  environment:
    FOO_VALUE: ${self:custom.foo}
```

The `provider` section configures values specific to your vendor.

STEP 1: **CONFIGURE SERVERLESS**

```
provider:
  name: aws
  stage: ${opt:stage}
  region: ${opt:region}
  logRetentionInDays: 7
  stackTags:
    SERVICE: ${self:service}
  iamRoleStatements:
  environment:
    FOO_VALUE: ${self:custom.foo}
```

Our vendor is AWS.

STEP 1: **CONFIGURE SERVERLESS**

```
provider:
  name: aws
  stage: ${opt:stage}
  region: ${opt:region}
  logRetentionInDays: 7
  stackTags:
    SERVICE: ${self:service}
  iamRoleStatements:
  environment:
    FOO_VALUE: ${self:custom.foo}
```

The `stage` value is used (along with the service name) to name resources.

STEP 1: **CONFIGURE SERVERLESS**

```
provider:
  name: aws
  stage: ${opt:stage}
  region: ${opt:region}
  logRetentionInDays: 7
  stackTags:
    SERVICE: ${self:service}
  iamRoleStatements:
  environment:
    FOO_VALUE: ${self:custom.foo}
```

These are variables: `${source:variable}`

There are several variable sources. The `opt:` source retrieves variables from command line flags (e.g. `--region us-east-1`).

The `self:` source references variables from within this YAML file.

STEP 1: **CONFIGURE SERVERLESS**

```
provider:
  name: aws
  stage: ${opt:stage}
  region: ${opt:region}
  logRetentionInDays: 7
  stackTags:
    SERVICE: ${self:service}
  iamRoleStatements:
  environment:
    FOO_VALUE: ${self:custom.foo}
```

Tagging resources comes in handy when trying to control and manage costs in AWS.

STEP 1: **CONFIGURE SERVERLESS**

```
provider:
  name: aws
  stage: ${opt:stage}
  region: ${opt:region}
  logRetentionInDays: 7
  stackTags:
    SERVICE: ${self:service}
  iamRoleStatements:
  environment:
    FOO_VALUE: ${self:custom.foo}
```

`iamRoleStatements` is an array of IAM permissions used to create your Lambda functions' execution role. For now, we'll leave it blank.

STEP 1: **CONFIGURE SERVERLESS**

```
provider:
  name: aws
  stage: ${opt:stage}
  region: ${opt:region}
  logRetentionInDays: 7
  stackTags:
    SERVICE: ${self:service}
  iamRoleStatements:
  environment:
    FOO_VALUE: ${self:custom.foo}
```

The `environment` section is for defining environment variables available to your Lambda functions at runtime. We'll add more useful items to this section later.

shiftgig™

STEP 2: **ADD FUNCTIONS**

```
functions:
  PutProduct:
    description: Creates and updates products
    memorySize: 128
    runtime: python3.6
    handler: handlers.put_product__http
    events:
      -
        http:
          method: put
          path: "/product/{name}"
          request:
            parameters:
              paths:
                name: true
```

The `functions` section of your Serverless configuration defines the various Lambda functions for your service.

STEP 2: **ADD FUNCTIONS**

```
functions:
  PutProduct:
    description: Creates and updates products
    memorySize: 128
    runtime: python3.6
    handler: handlers.put_product__http
    events:
      -
        http:
          method: put
          path: "/product/{name}"
          request:
            parameters:
              paths:
                name: true
```

Your Lambda function's memory allocation, in megabytes.

STEP 2: **ADD FUNCTIONS**

```
functions:
  PutProduct:
    description: Creates and updates products
    memorySize: 128
    runtime: python3.6
    handler: handlers.put_product__http
    events:
      -
        http:
          method: put
          path: "/product/{name}"
          request:
            parameters:
              paths:
                name: true
```

Any supported Lambda runtime

STEP 2: **ADD FUNCTIONS**

```
functions:
  PutProduct:
    description: Creates and updates products
    memorySize: 128
    runtime: python3.6
    handler: handlers.put_product__http
    events:
      -
        http:
          method: put
          path: "/product/{name}"
          request:
            parameters:
              paths:
                name: true
```

The `handler` string defines the function in your source code that should be called by your Lambda function when it is invoked.

STEP 2: **ADD FUNCTIONS**

```
functions:
  PutProduct:
    description: Creates and updates products
    memorySize: 128
    runtime: python3.6
    handler: handlers.put_product__http
    events:
      -
        http:
          method: put
          path: "/product/{name}"
          request:
            parameters:
              paths:
                name: true
```

The `events` array defines various triggers for your Lambda function. Here, we are defining API Gateway triggers to expose this functionality within a web API.

This marks the `name` segment of the URL as required.

STEP 3: **WRITE HANDLER CODE**

```
# handlers.py
import json
from urllib.parse import unquote

def put_product__http(event, context):
    payload = json.loads(event['body'])
    product = {
        'name': unquote(event['pathParameters']['name']),
        'price': payload['price'],
        'description': payload['description']
    }
    print('Saving product {}'.format(product['name']))
    return {
        'statusCode': '200',
        'body': json.dumps(product),
        'headers': {
            'Content-Type': 'application/json'
        }
    }
```

Standard `(event, context)` signature for our handler function. The `event` argument is a dict containing information about the request.

STEP 3: **WRITE HANDLER CODE**

```
# handlers.py
import json
from urllib.parse import unquote

def put_product__http(event, context):
    payload = json.loads(event['body'])
    product = {
        'name': unquote(event['pathParameters']['name']),
        'price': payload['price'],
        'description': payload['description']
    }
    print('Saving product {}'.format(product['name']))
    return {
        'statusCode': '200',
        'body': json.dumps(product),
        'headers': {
            'Content-Type': 'application/json'
        }
    }
```

We'll add database interactions here, but just log something for now.

API Gateway expects response data in this format.

STEP 4: **DEPLOY!**

```
$ export SLS_DEBUG=*


$ serverless deploy \
  --region us-east-1 \
  --aws-profile myprofile \
  --stage dev \
  --verbose
```

Makes our output nice and verbose!

Specify the AWS region your deployment should target

If you use IAM role-assumption to authenticate with multiple AWS accounts, specify your profile name here

Sensible and consistent `stage` values help keep things organized. It is used to name your CloudFormation stack and its resources.

Even more verbose!

STEP 4: **DEPLOY!**

```
CloudFormation - UPDATE_IN_PROGRESS - AWS::CloudFormation::Stack - candystore-dev
CloudFormation - CREATE_IN_PROGRESS - AWS::IAM::Role - IamRoleLambdaExecution
CloudFormation - CREATE_IN_PROGRESS - AWS::Logs::LogGroup - PutProductLogGroup
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::RestApi - ApiGatewayRestApi
CloudFormation - CREATE_IN_PROGRESS - AWS::IAM::Role - IamRoleLambdaExecution
CloudFormation - CREATE_IN_PROGRESS - AWS::Logs::LogGroup - PutProductLogGroup
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::RestApi - ApiGatewayRestApi
CloudFormation - CREATE_COMPLETE - AWS::ApiGateway::RestApi - ApiGatewayRestApi
CloudFormation - CREATE_COMPLETE - AWS::Logs::LogGroup - PutProductLogGroup
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Resource - ApiGatewayResourceProduct
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Resource - ApiGatewayResourceProduct
CloudFormation - CREATE_COMPLETE - AWS::ApiGateway::Resource - ApiGatewayResourceProduct
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Resource -
ApiGatewayResourceProductNameVar
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Resource -
ApiGatewayResourceProductNameVar
CloudFormation - CREATE_COMPLETE - AWS::ApiGateway::Resource - ApiGatewayResourceProductNameVar
CloudFormation - CREATE_COMPLETE - AWS::IAM::Role - IamRoleLambdaExecution
CloudFormation - CREATE_IN_PROGRESS - AWS::Lambda::Function - PutProductLambdaFunction
CloudFormation - CREATE_IN_PROGRESS - AWS::Lambda::Function - PutProductLambdaFunction
CloudFormation - CREATE_COMPLETE - AWS::Lambda::Function - PutProductLambdaFunction
CloudFormation - CREATE_IN_PROGRESS - AWS::ApiGateway::Method -
ApiGatewayMethodProductNameVarPut
CloudFormation - CREATE_IN_PROGRESS - AWS::Lambda::Version -
PutProductLambdaVersionI17kp6O9rcYZSzgFOR8MgEqWaxZpBDZjz94X6WavEcD
```

Look at all that CloudFormation you didn't have to write!

shiftgig™

STEP 4: **DEPLOY!**

```
Serverless: Stack update finished...
Service Information
service: candystore
stage: dev
region: us-east-1
stack: candystore-dev
api keys:
  None
endpoints:
  PUT - https://fv907pkjxg.execute-api.us-east-1.amazonaws.com/dev/product/{name}
functions:
  PutProduct: candystore-dev-PutProduct
Stack Outputs
PutProductLambdaFunctionQualifiedArn:
arn:aws:lambda:us-east-1:123456789876:function:candystore-dev-PutProduct:1
ServiceEndpoint: https://fv907pkjxg.execute-api.us-east-1.amazonaws.com/dev
ServerlessDeploymentBucketName: candystore-dev-serverlessdeploymentbucket-1s7wgv4xc93h6
```

Useful information printed at the end!

Includes our API Gateway endpoints

shiftgig™

STEP 5: **TEST**

```
$ curl -X PUT \
  https://fv907pkjxg.execute-api.us-east-1.amazonaws.com/dev/product/skittles \
  -H 'Content-Type: application/json' \
  -d '{
    "description": "Taste the rainbow",
    "price": "2.55"
}' \
-w '\n'

{"name": "skittles", "price": "2.55", "description": "Taste the rainbow"}
```

STEP 5: **TEST**

```
$ serverless logs \
  --function PutProduct \
  --region us-east-1 \
  --aws-profile poc \
  --stage dev \
  --verbose \
  --tail

START RequestId: ff3ff102-692f-11e8-801f-cde97c6f36e8 Version: $LATEST
Saving product skittles
END RequestId: ff3ff102-692f-11e8-801f-cde97c6f36e8
REPORT RequestId: ff3ff102-692f-11e8-801f-cde97c6f36e8  Duration: 1.72 ms
Billed Duration: 100 ms     Memory Size: 128 MB Max Memory Used: 22 MB
```

STEP 6: **ADD A DYNAMODB TABLE**

```yaml
custom:
  foo: bar
  pythonRequirements: # Python only
    usePipenv: true
    dockerizePip: true
    dockerImage: lambci/lambda:build-python3.6
    dockerSsh: true
  product_table_name: ${self:service}-${self:provider.stage}-product
```

We centrally define the name of our DynamoDB table in `custom` so we can reference it elsewhere.

STEP 6: **ADD A DYNAMODB TABLE**

```
resources:
  Resources:
    ProductDynamoDBTable:
      Type: "AWS::DynamoDB::Table"
      Properties:
        AttributeDefinitions:
          -
            AttributeName: name
            AttributeType: S
        KeySchema:
          -
            AttributeName: name
            KeyType: HASH
        TableName: ${self:custom.product_table_name}
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
```

The `resources` section is reserved for CloudFormation syntax.

In the `Resources` subsection, you define AWS resources for your stack that aren't implicitly created for you in the `functions` section.

Use the variable defined in `custom` to name the table

shiftgig™

STEP 6: **ADD A DYNAMODB TABLE**

```
iamRoleStatements:
    -
      Effect: Allow
      Action:
        - dynamodb:GetItem
        - dynamodb:PutItem
        - dynamodb:UpdateItem
        - dynamodb:DeleteItem
      Resource:
        -
          Fn::GetAtt: [ "ProductDynamoDBTable", "Arn" ]
```

We add the IAM permissions our Lambda functions will require in order to interact with our DynamoDB table.

This statement will be added to the existing execution role.

We can use CloudFormation intrinsic functions in our `serverless.yml` file too!

STEP 6: **ADD A DYNAMODB TABLE**

```
environment:
    FOO_VALUE: ${self:custom.foo}
    PRODUCT_TABLE_NAME: ${self:custom.product_table_name}
```

We'll also add an environment variable to make our table's name available to our Lambda functions

STEP 6: **ADD A DYNAMODB TABLE**

```
# handlers.py
import json
import os
from urllib.parse import unquote
import boto3
def put_product__http(event, context):
    payload = json.loads(event['body'])
    product = {
        'name': unquote(event['pathParameters']['name']),
        'price': payload['price'],
        'description': payload['description']
    }
    db = boto3.resource('dynamodb')
    table = db.Table(name=os.getenv('PRODUCT_TABLE_NAME'))
    table.put_item(Item=product)
    print('Saved product {}'.format(product['name']))
    return {
        'statusCode': '200',
        'body': json.dumps(product),
        'headers': {
            'Content-Type': 'application/json'
```

Lambda provides the `boto3` library automatically.

Save the product to the DynamoDB table!

STEP 7: **DEPLOY CHANGES AND TEST (AGAIN)**

```
$ serverless deploy \
  --region us-east-1 \
  --aws-profile myprofile \
  --stage dev \
  --verbose
```

STEP 7: **DEPLOY CHANGES AND TEST (AGAIN)**

```
$ curl -X PUT \
  https://fv907pkjxg.execute-api.us-east-1.amazonaws.com/dev/product/snickers \
  -H 'Content-Type: application/json' \
  -d '{
    "description": "Hungry? Grab a Snickers!",
    "price": "2.55"
}' \
-w '\n'

{"name": "snickers", "price": "1.25", "description": "Hungry? Grab a Snickers!"}
```

STEP 7: **DEPLOY CHANGES AND TEST (AGAIN)**

```
$ serverless logs \
  --function PutProduct \
  --region us-east-1 \
  --aws-profile poc \
  --stage dev \
  --verbose \
  --tail \
  --startTime 5m

START RequestId: b7ebe0b5-76b0-4b31-bfe9-334892d57674 Version: $LATEST
Saved product snickers
END RequestId: b7ebe0b5-76b0-4b31-bfe9-334892d57674
REPORT RequestId: b7ebe0b5-76b0-4b31-bfe9-334892d57674  Duration: 464.11 ms
Billed Duration: 500 ms     Memory Size: 128 MB Max Memory Used: 32 MB
```

NEXT STEPS: **MORE FEATURES**

- Respond to GET requests
- Respond to DELETE requests
- **Bonus:** Restrict access to your API with a usage plan in under 5 minutes!

RECAP

FINAL NOTES: **LINKS**

- **Shiftgig is hiring!**
  - https://www.shiftgig.com/careers
- Example code used today:
  - https://github.com/shiftgig/aws-serverless-workshop

shiftgig™

Q & A

shiftgig™