Deep Dive on Serverless Application Development

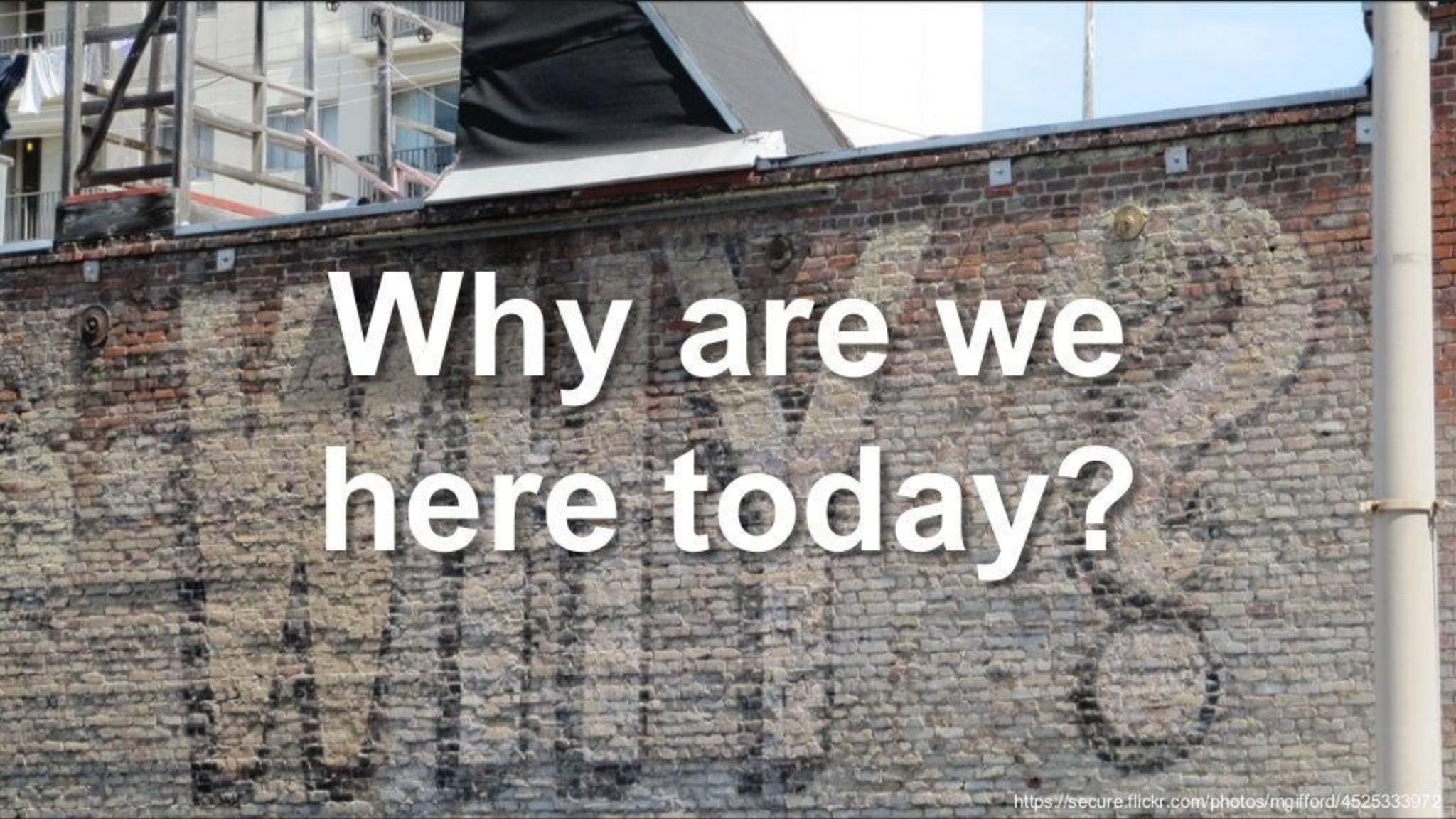
Hal Stiles – Senior Consultant - Serverless

About me:

Hal Stiles - stilesh@amazon.com, @halstiles

- Senior Consultant – AWS Professional Services
- Thoughtworks
- Previously:
  - SVP, Piksel
  - VP, Warner Bros
  - Disney NGE
- 25+ years technology
- 5+ years AWS
- Early Adopter / Specialist – Serverless and Containers
- Developer

aws


Why are we here today?
https://secure.flickr.com/photos/mgifford/4525333972

Serverless means…

No servers to provision or manage

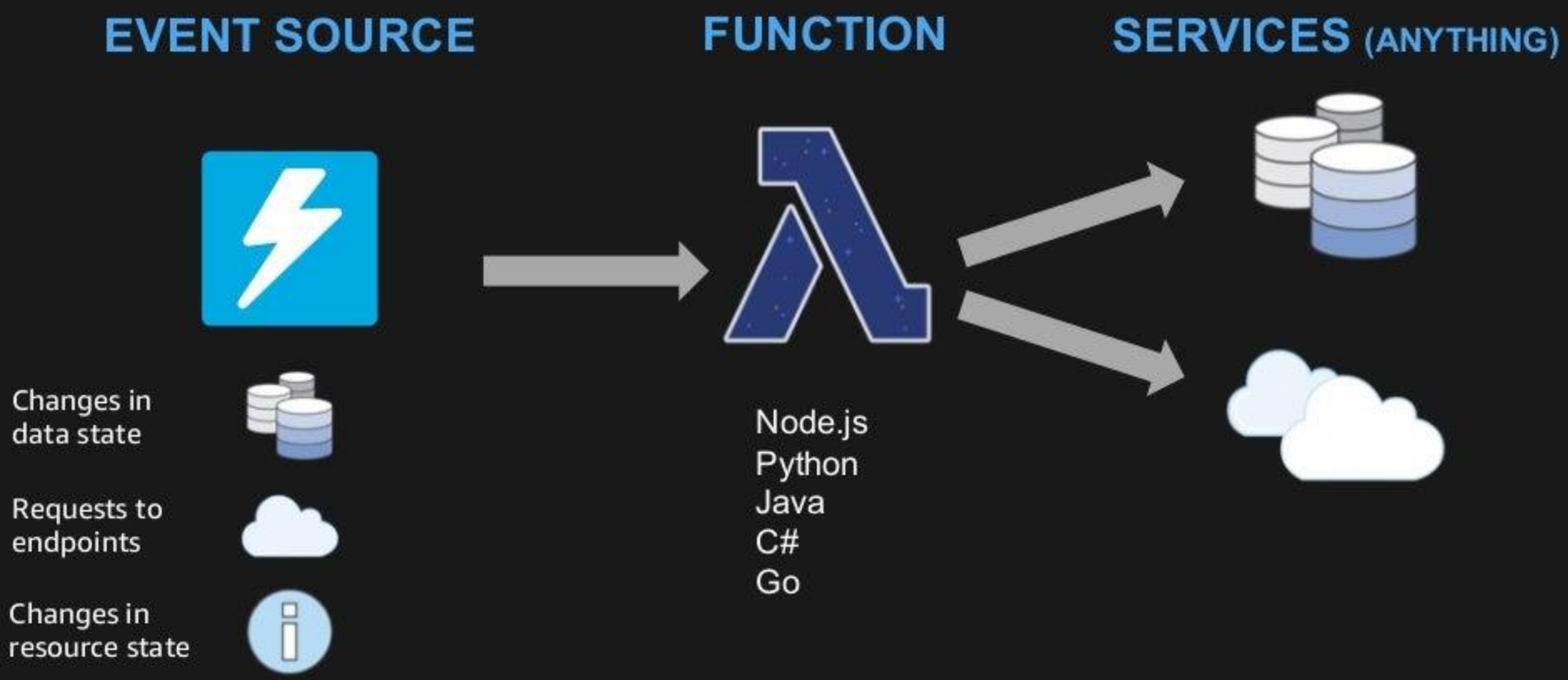
Scales with usage

Never pay for idle

Availability and fault tolerance built in

aws

Serverless applications



Common Lambda use cases

Web Applications

- Static websites
- Complex web apps
- Packages for Flask and Express

Backends

- Apps & services
- Mobile
- IoT

Data Processing

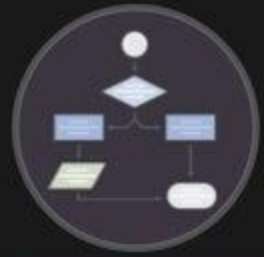
- Real time
- MapReduce
- Batch

Chatbots

- Powering chatbot logic

Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit



IT Automation

- Policy engines
- Extending AWS services
- Infrastructure management

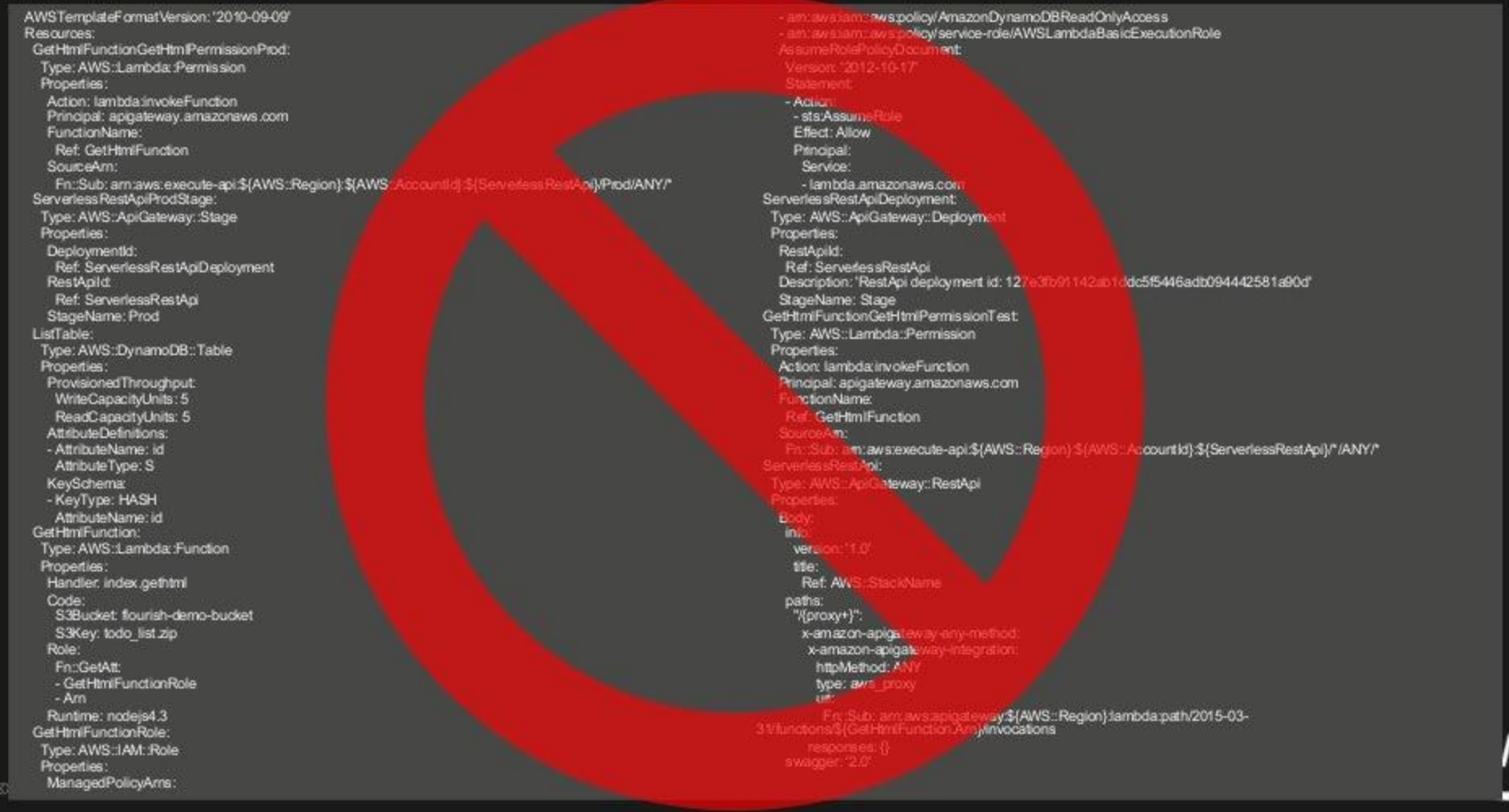
aws

https://secure.flickr.com/photos/stevendepolo/574192025/

CloudFormation template

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  GetHtmlFunctionGetHtmlPermissionProd:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:invokeFunction
      Principal: apigateway.amazonaws.com
      FunctionName:
        Ref: GetHtmlFunction
      SourceArn:
        Fn::Sub: arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${ServerlessRestApi}/Prod/ANY/*
  ServerlessRestApiProdStage:
    Type: AWS::ApiGateway::Stage
    Properties:
      DeploymentId:
        Ref: ServerlessRestApiDeployment
      RestApiId:
        Ref: ServerlessRestApi
      StageName: Prod
  ListTable:
    Type: AWS::DynamoDB::Table
    Properties:
      ProvisionedThroughput:
        WriteCapacityUnits: 5
        ReadCapacityUnits: 5
      AttributeDefinitions:
      - AttributeName: id
        AttributeType: S
      KeySchema:
      - KeyType: HASH
        AttributeName: id
  GetHtmlFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.gethtml
      Code:
        S3Bucket: flourish-demo-bucket
        S3Key: todo_list.zip
      Role:
        Fn::GetAtt:
        - GetHtmlFunctionRole
        - Arn
      Runtime: nodejs4.3
  GetHtmlFunctionRole:
    Type: AWS::IAM::Role
    Properties:
      ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonDynamoDBReadOnlyAccess
      - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Action:
          - sts:AssumeRole
          Effect: Allow
          Principal:
            Service:
            - lambda.amazonaws.com
  ServerlessRestApiDeployment:
    Type: AWS::ApiGateway::Deployment
    Properties:
      RestApiId:
        Ref: ServerlessRestApi
      Description: 'RestApi deployment id: 127e3fb91142ab1ddc5f546adb094442581a90d'
      StageName: Stage
  GetHtmlFunctionGetHtmlPermissionTest:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:invokeFunction
      Principal: apigateway.amazonaws.com
      FunctionName:
        Ref: GetHtmlFunction
      SourceArn:
        Fn::Sub: arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${ServerlessRestApi}/*/ANY/*
  ServerlessRestApi:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Body:
        info:
          version: '1.0'
          title:
            Ref: AWS::StackName
        paths:
          "/{proxy+}":
            x-amazon-apigateway-any-method:
              x-amazon-apigateway-integration:
                httpMethod: ANY
                type: aws_proxy
                uri:
                  Fn::Sub: arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-
31/functions/${GetHtmlFunction.Arn}/invocations
                responses: {}
        swagger: '2.0'
```

CloudFormation template



Meet SAM!

AWS Serverless Application Model (SAM)

CloudFormation extension optimized for serverless

New serverless resource types: functions, APIs, and tables

Supports anything CloudFormation supports

Open specification (Apache 2.0)

https://github.com/awslabs/serverless-application-model

SAM template

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetHtmlFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/todo_list.zip
      Handler: index.gethtml
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetHtml:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: ANY

  ListTable:
    Type: AWS::Serverless::SimpleTable
```

SAM template

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetHtmlFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/todo_list.zip
      Handler: index.gethtml
      Runtime: nodejs4.3
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetHtml:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: ANY

  ListTable:
    Type: AWS::Serverless::SimpleTable
```

Tells CloudFormation this is a SAM template it needs to “transform”

Creates a Lambda function with the referenced managed IAM policy, runtime, code at the referenced zip location, and handler as defined. Also creates an API Gateway and takes care of all mapping/permissions necessary

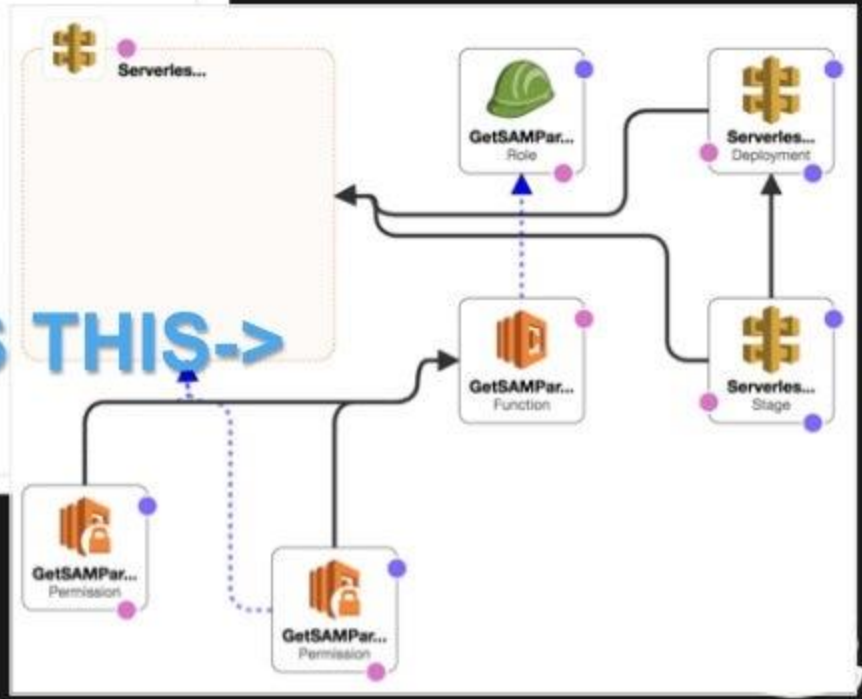
Creates a DynamoDB table with 5 Read & Write units

SAM template

26 lines (25 sloc) | 827 Bytes — Raw | Blame | History

```yaml
Transform: 'AWS::Serverless-2016-10-31'
Parameters:
    SamMultipler:
        Description: "SAM multiplier. Make this really big to have a party :)"
        Type: "String"
    OriginUrl:
        Description: "The origin url to allow CORS requests from. This will be the base URL of your static SAM website."
        Type: "String"
Resources:
    GetSAMPartyCount:
        Type: AWS::Serverless::Function
        Properties:
            Handler: index.handler
            Runtime: nodejs4.3
            CodeUri: ./
            Environment:
                Variables:
                    SAM_MULTIPLIER: !Ref SamMultipler
                    ORIGIN_URL: !Ref OriginUrl
            Events:
              GetResource:
                Type: Api
                Properties:
                  Path: /sam
                  Method: get
```

<-THIS BECOMES THIS->



From: https://github.com/awslabs/aws-serverless-samfarm/blob/master/api/saml.yaml

SAM Template Properties

AWS::Serverless::Function

AWS::Serverless::Api

AWS::Serverless::SimpleTable

```
Handler: index.js
Runtime: nodejs4.3
CodeUri: 's3://my-code-bucket/my-function.zip'
Description: Creates thumbnails of uploaded
images
MemorySize: 1024
Timeout: 15
Policies: AmazonS3FullAccess
Environment:
  Variables:
    TABLE_NAME: my-table
Events:
  PhotoUpload:
    Type: S3
    Properties:
       Bucket: my-photo-bucket
Tracing: Active|PassThrough
Tags:
  AppNameTag: ThumbnailApp
  DepartmentNameTag: ThumbnailDepartment
```

From SAM Version 2016-10-31

AWS::Serverless::Function Event source types

S3
SNS
Kinesis | DynamoDB
Api
Schedule
CloudWatchEvent
IoTRule
AlexaSkill

Note: Events are a map of string to Event Source Object

Event Source Objects have the following structure:

```
Type:
Properties:
```

For Example:

```
Events:
  MyEventName:
    Type: S3
    Properties:
      Bucket: my-photo-bucket
```

From SAM Version 2016-10-31

SAM Globals + Safe Deployments

```
Globals:
  Function:
    Runtime: nodejs4.3
    AutoPublishAlias: !Ref ENVIRONMENT

MyLambdaFunction:
  Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      DeploymentPreference:
        Type: Linear10PercentEvery10Minutes
        Alarms:
          # A list of alarms that you want to monitor
          - !Ref AliasErrorMetricGreaterThanZeroAlarm
          - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
        Hooks:
          # Validation Lambda functions that are run before & after traffic shifting
          PreTraffic: !Ref PreTrafficLambdaFunction
          PostTraffic: !Ref PostTrafficLambdaFunction
```

SAM Globals + Safe Deployments

```
Globals:
  Function:
    Runtime: nodejs4.3
    AutoPublishAlias: !Ref ENVIRONMENT

MyLambdaFunction:
  Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      DeploymentPreference:
        Type: Linear10PercentEvery10Minutes
        Alarms:
          # A list of alarms that you want to monitor
          - !Ref AliasErrorMetricGreaterThanZeroAlarm
          - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
        Hooks:
          # Validation Lambda functions that are run before & after traffic shifting
          PreTraffic: !Ref PreTrafficLambdaFunction
          PostTraffic: !Ref PostTrafficLambdaFunction
```

Lambda Alias Traffic Shifting & AWS SAM

In SAM:

AutoPublishAlias

By adding this property and specifying an alias name, AWS SAM will do the following:

- Detect when new code is being deployed based on changes to the Lambda function's Amazon S3 URI.
- Create and publish an updated version of that function with the latest code.
- Create an alias with a name you provide (unless an alias already exists) and points to the updated version of the Lambda function.

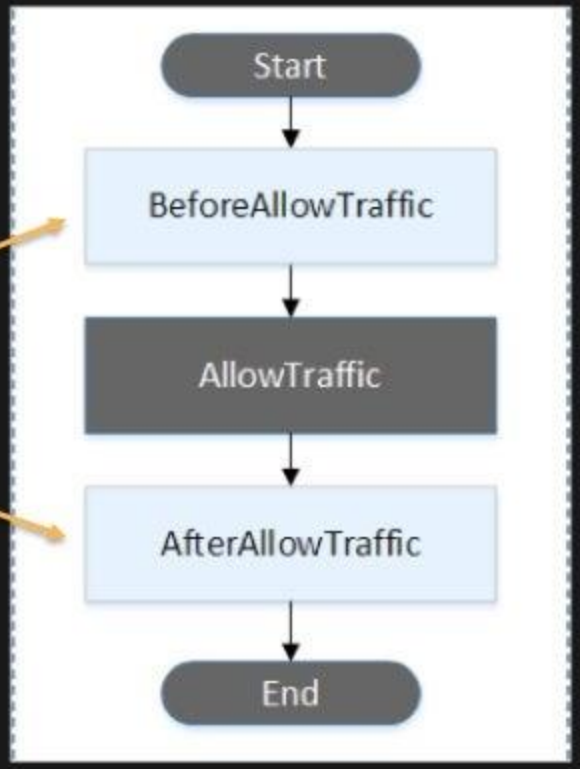
Deployment Preference Type

Canary10Percent30Minutes
Canary10Percent5Minutes
Canary10Percent10Minutes
Canary10Percent15Minutes
Linear10PercentEvery10Minutes
Linear10PercentEvery1Minute
Linear10PercentEvery2Minutes
Linear10PercentEvery3Minutes
AllAtOnce

Lambda Alias Traffic Shifting & AWS SAM

In SAM:

```
Alarms: # A list of alarms that you want to monitor
- !Ref AliasErrorMetricGreaterThanZeroAlarm
- !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
Hooks: # Validation Lambda functions that are run
before & after traffic shifting
PreTraffic: !Ref PreTrafficLambdaFunction
PostTraffic: !Ref PostTrafficLambdaFunction
```



Note: You can specify a maximum of 10 alarms

SAM commands – Package & Deploy

Package

- Creates a deployment package (.zip file)
- Uploads deployment package to an Amazon S3 Bucket
- Adds a CodeUri property with S3 URI

Deploy

- Calls CloudFormation ‘CreateChangeSet’ API
- Calls CloudFormation ‘ExecuteChangeSet’ API

aws

AWS CodeDeploy + Lambda

NEW: Can deploy AWS Lambda!!

Uses AWS SAM to deploy serverless applications

Supports Lambda Alias Traffic Shifting enabling canaries and blue|green deployments

Can rollback based on CloudWatch Metrics/Alarms

Pre/Post-Traffic Triggers can integrate with other services (or even call Lambda functions)

AWS CodeDeploy + Lambda

CodeDeploy comes with a number of added capabilities:

- Custom deployment configurations. Examples:
  - “Canary 5% for 1 hour”
  - “Linear 20% every 1 hour”
- Notification events via SNS on success/failure/rollback
- Console with visibility on deploy status, history, and rollbacks.

SAM Local

CLI tool for local testing of serverless apps

Works with Lambda functions and “proxy-style” APIs

Response object and function logs available on your local machine

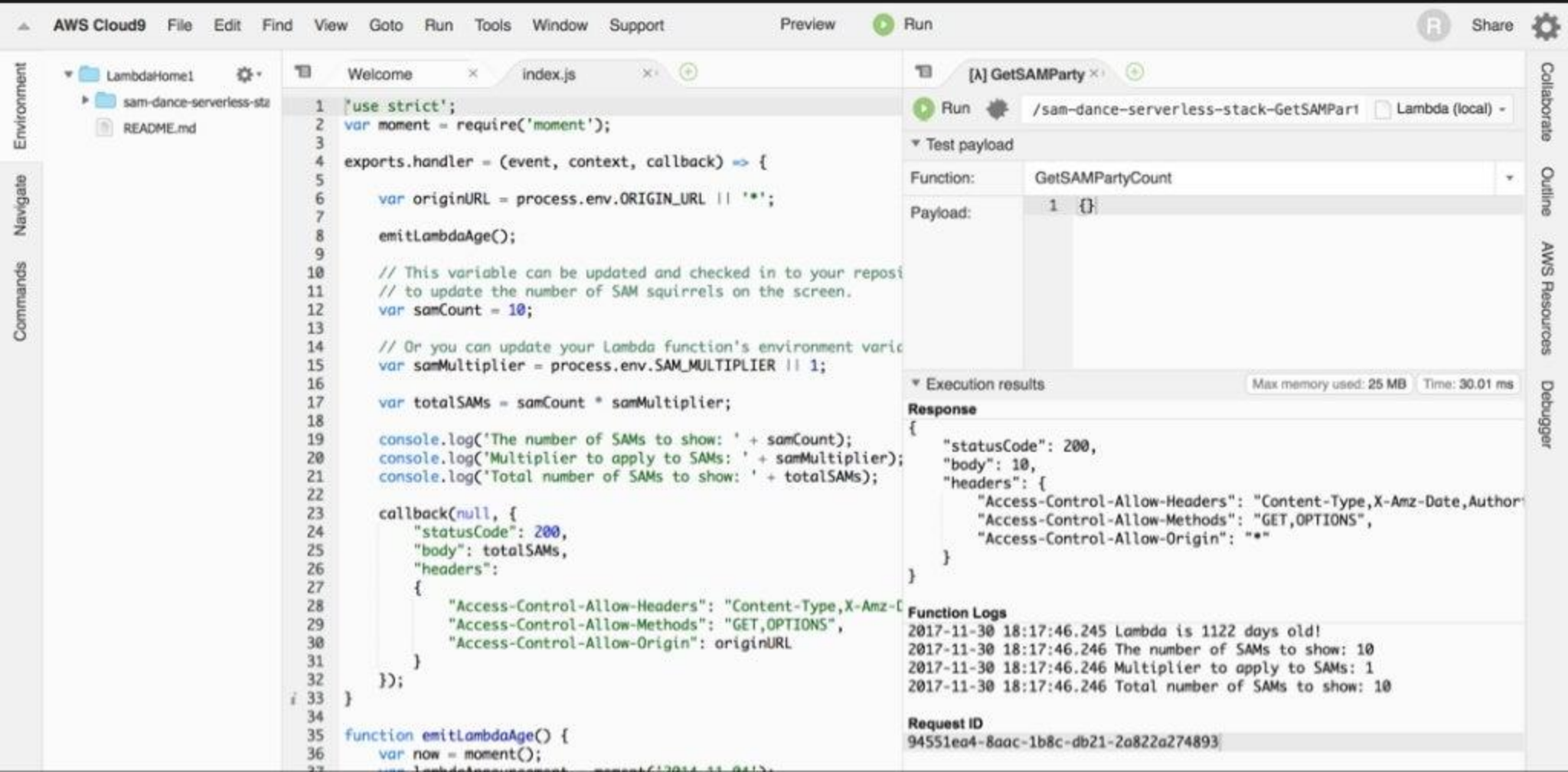
Uses open source docker-lambda images to mimic Lambda’s execution environment:

- Emulates timeout, memory limits, runtimes

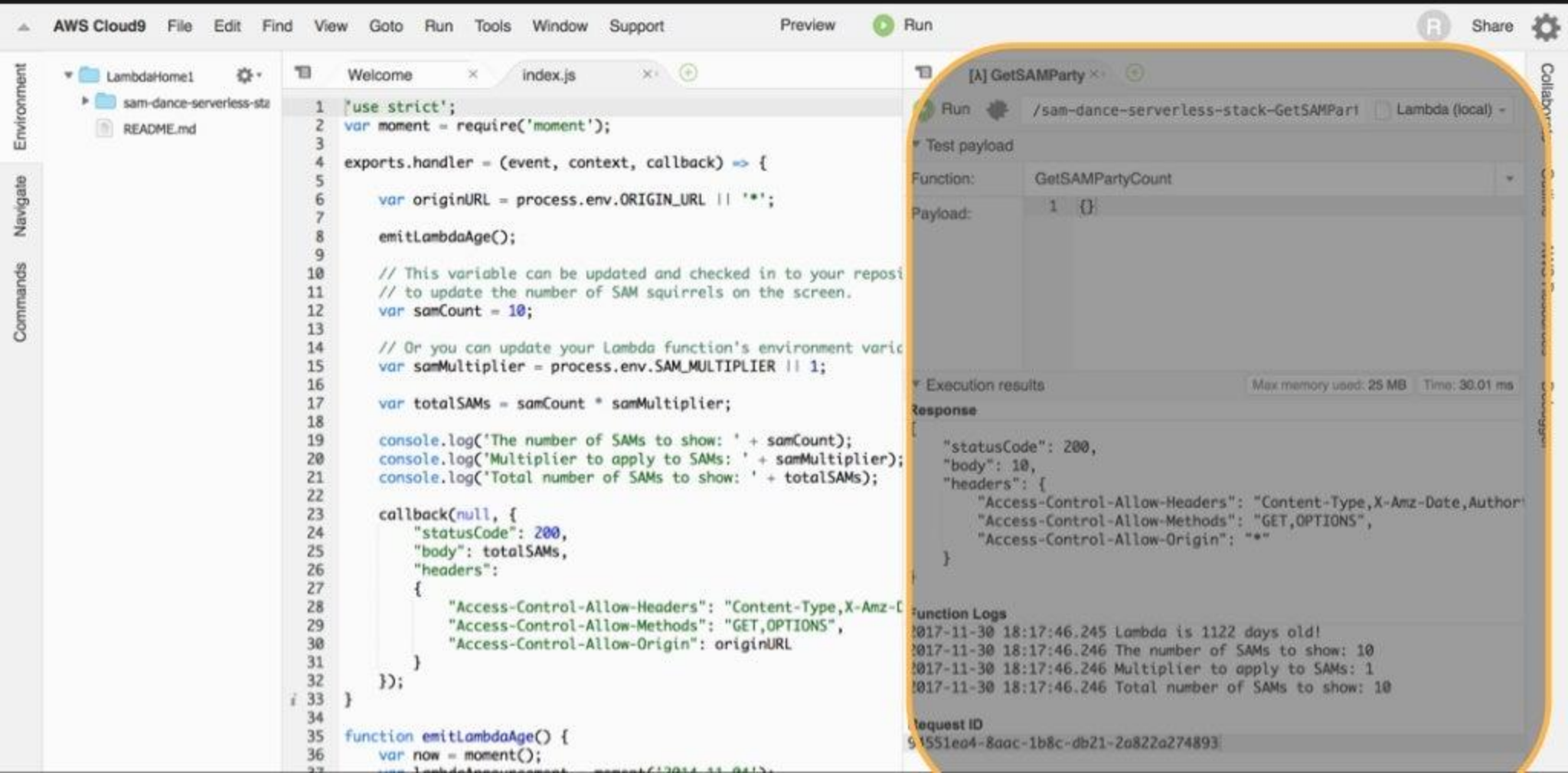
https://github.com/awslabs/aws-sam-local

aws

AWS Cloud9


AWS Cloud9 File Edit Find View Goto Run Tools Window Support Preview Run Share
Environment
Navigate
Commands
LambdaHome1
sam-dance-serverless-sta
README.md
Welcome
index.js
'use strict';
var moment = require('moment');
exports.handler = (event, context, callback) => {
var originURL = process.env.ORIGIN_URL || '*';
emitLambdaAge();
// This variable can be updated and checked in to your reposi
// to update the number of SAM squirrels on the screen.
var samCount = 10;
// Or you can update your Lambda function's environment varia
var samMultiplier = process.env.SAM_MULTIPLIER || 1;
var totalSAMs = samCount * samMultiplier;
console.log('The number of SAMs to show: ' + samCount);
console.log('Multiplier to apply to SAMs: ' + samMultiplier);
console.log('Total number of SAMs to show: ' + totalSAMs);
callback(null, {
"statusCode": 200,
"body": totalSAMs,
"headers":
{
"Access-Control-Allow-Headers": "Content-Type,X-Amz-D
"Access-Control-Allow-Methods": "GET,OPTIONS",
"Access-Control-Allow-Origin": originURL
}
});
}
function emitLambdaAge() {
var now = moment();
[λ] GetSAMParty
Run
/sam-dance-serverless-stack-GetSAMPart
Lambda (local)
Test payload
Function:
GetSAMPartyCount
Payload:
1 {}
Execution results
Max memory used: 25 MB
Time: 30.01 ms
Response
{
"statusCode": 200,
"body": 10,
"headers": {
"Access-Control-Allow-Headers": "Content-Type,X-Amz-Date,Author
"Access-Control-Allow-Methods": "GET,OPTIONS",
"Access-Control-Allow-Origin": "*"
}
}
Function Logs
2017-11-30 18:17:46.245 Lambda is 1122 days old!
2017-11-30 18:17:46.246 The number of SAMs to show: 10
2017-11-30 18:17:46.246 Multiplier to apply to SAMs: 1
2017-11-30 18:17:46.246 Total number of SAMs to show: 10
Request ID
94551ea4-8aac-1b8c-db21-2a822a274893
Collaborate
Outline
AWS Resources
Debugger

AWS Cloud9


AWS Cloud9 File Edit Find View Goto Run Tools Window Support
Preview
Run
Share
Environment
Navigate
Commands
LambdaHome1
sam-dance-serverless-sta
README.md
Welcome
index.js
'use strict';
var moment = require('moment');
exports.handler = (event, context, callback) => {
var originURL = process.env.ORIGIN_URL || '*';
emitLambdaAge();
// This variable can be updated and checked in to your reposi
// to update the number of SAM squirrels on the screen.
var samCount = 10;
// Or you can update your Lambda function's environment varic
var samMultiplier = process.env.SAM_MULTIPLIER || 1;
var totalSAMs = samCount * samMultiplier;
console.log('The number of SAMs to show: ' + samCount);
console.log('Multiplier to apply to SAMs: ' + samMultiplier);
console.log('Total number of SAMs to show: ' + totalSAMs);
callback(null, {
"statusCode": 200,
"body": totalSAMs,
"headers":
{
"Access-Control-Allow-Headers": "Content-Type,X-Amz-D
"Access-Control-Allow-Methods": "GET,OPTIONS",
"Access-Control-Allow-Origin": originURL
}
});
}
function emitLambdaAge() {
var now = moment();
[λ] GetSAMParty
Run
/sam-dance-serverless-stack-GetSAMPart
Lambda (local)
Test payload
Function:
GetSAMPartyCount
Payload:
1 {}
Execution results
Max memory used: 25 MB
Time: 30.01 ms
Response
"statusCode": 200,
"body": 10,
"headers": {
"Access-Control-Allow-Headers": "Content-Type,X-Amz-Date,Author
"Access-Control-Allow-Methods": "GET,OPTIONS",
"Access-Control-Allow-Origin": "*"
}
Function Logs
2017-11-30 18:17:46.245 Lambda is 1122 days old!
2017-11-30 18:17:46.246 The number of SAMs to show: 10
2017-11-30 18:17:46.246 Multiplier to apply to SAMs: 1
2017-11-30 18:17:46.246 Total number of SAMs to show: 10
Request ID
551ea4-8aac-1b8c-db21-2a822a274893
Collaborate

https://secure.flickr.com/photos/jasoneppink/499531891

Metrics and logging are a universal right!

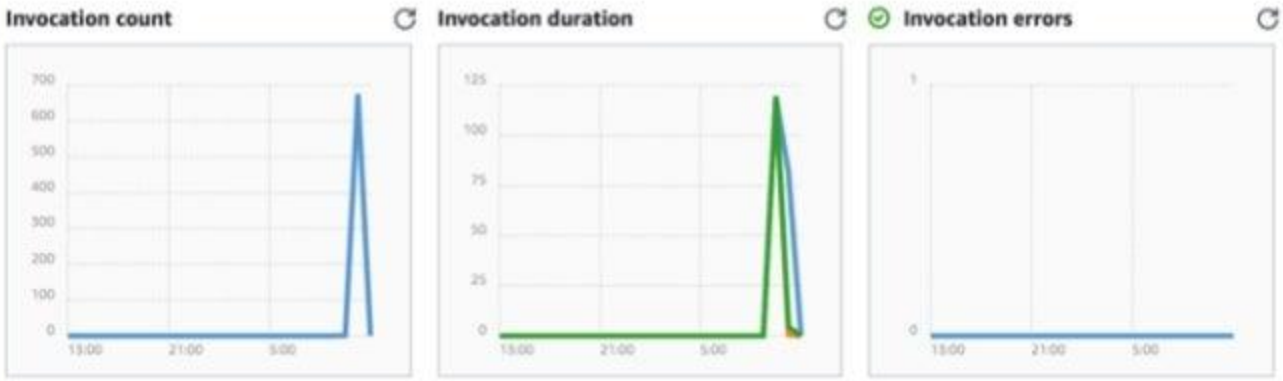
CloudWatch Metrics:

- 7 Built in metrics for Lambda
  - Invocation Count, Invocation duration, Invocation errors, Throttled Invocation, Iterator Age, DLQ Errors, Concurrency
  - Can call "put-metric-data" from your function code for custom metrics
- 7 Built in metrics for API-Gateway
  - API Calls Count, Latency, 4XXs, 5XXs, Integration Latency, Cache Hit Count, Cache Miss Count
  - Error and Cache metrics support *averages and percentiles*

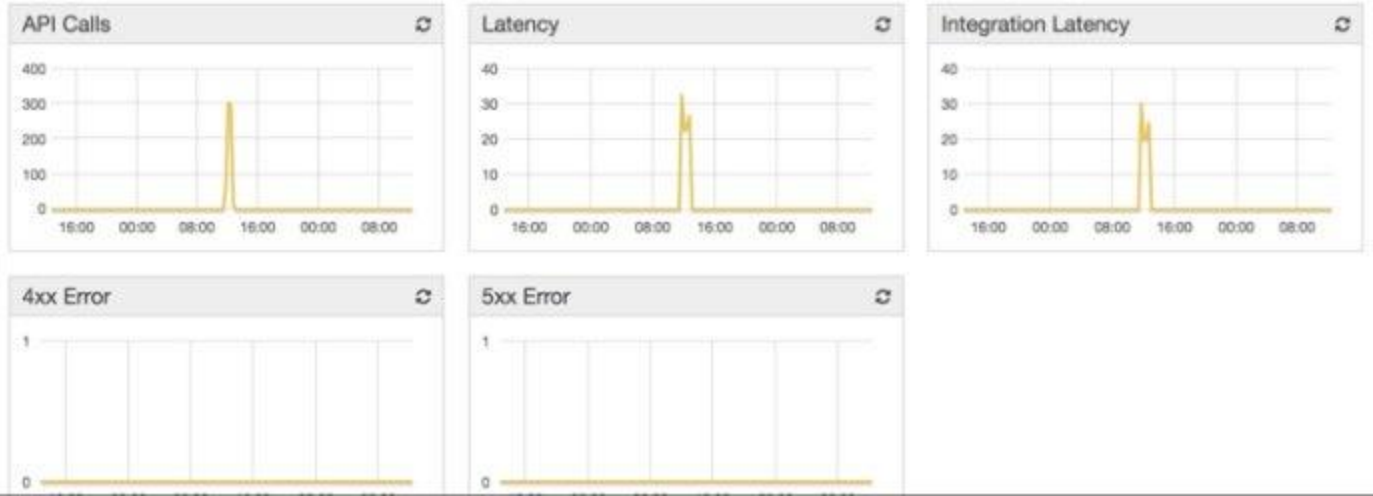
View traces in X-Ray

CloudWatch metrics at a glance (last 24 hours)

View logs in CloudWatch



Stage Prod From 11/23/17 To 11/24/17



Metrics and logging are a universal right!

CloudWatch Logs:

- API Gateway Logging
  - 2 Levels of logging, ERROR and INFO
  - Optionally log method request/body content
  - Set globally in stage, or override per method
- Lambda Logging
  - Logging directly from your code with your language's equivalent of console.log()
  - Basic request information included
- Log Pivots
  - Build metrics based on log filters
  - Jump to logs that generated metrics
- Export logs to AWS ElastiCache or S3
  - Explore with Kibana or Athena/QuickSight

Metrics and logging are a universal right!

CloudWatch Logs:

- API Gateway Logging
  - 2 Levels of logging, ERROR and INFO
  - Optionally log method request/body content
  - Set globally in stage, or override per method
- Lambda Logging
  - Logging directly from your code with your language's equivalent of console.log()
  - Basic request information included
- Log Pivots
  - Build metrics based on log filters
  - Jump to logs that generated metrics

Export logs to AWS ElastiCache or S3

- Explore with Kibana or Athena/QuickSight

Amazon API Gateway Custom Access Logging

Settings | Logs | Stage Variables | SDK Generation | Export | Deployment History | Documentation History

Configure the metering and caching settings for the stage.

CloudWatch Settings

Enable CloudWatch Logs

Enable Detailed CloudWatch Metrics

Custom Access Logging

Enable Access Logging

CloudWatch Group arn:aws:logs:us-east-1:account_no:log-group:API-Ga

Log Format

```
{ "requestId":"$context.requestId", "ip": "$context.identity.sourceIp", "caller":"$context.identity.caller",
"user":"$context.identity.user","requestTime":"$context.requestTime",
"httpMethod":"$context.httpMethod","resourcePath":"$context.resourcePath",
"status":"$context.status","protocol":"$context.protocol", "responseLength":"$context.responseLength" }
```

Insert Example: CLF | JSON | XML | CSV

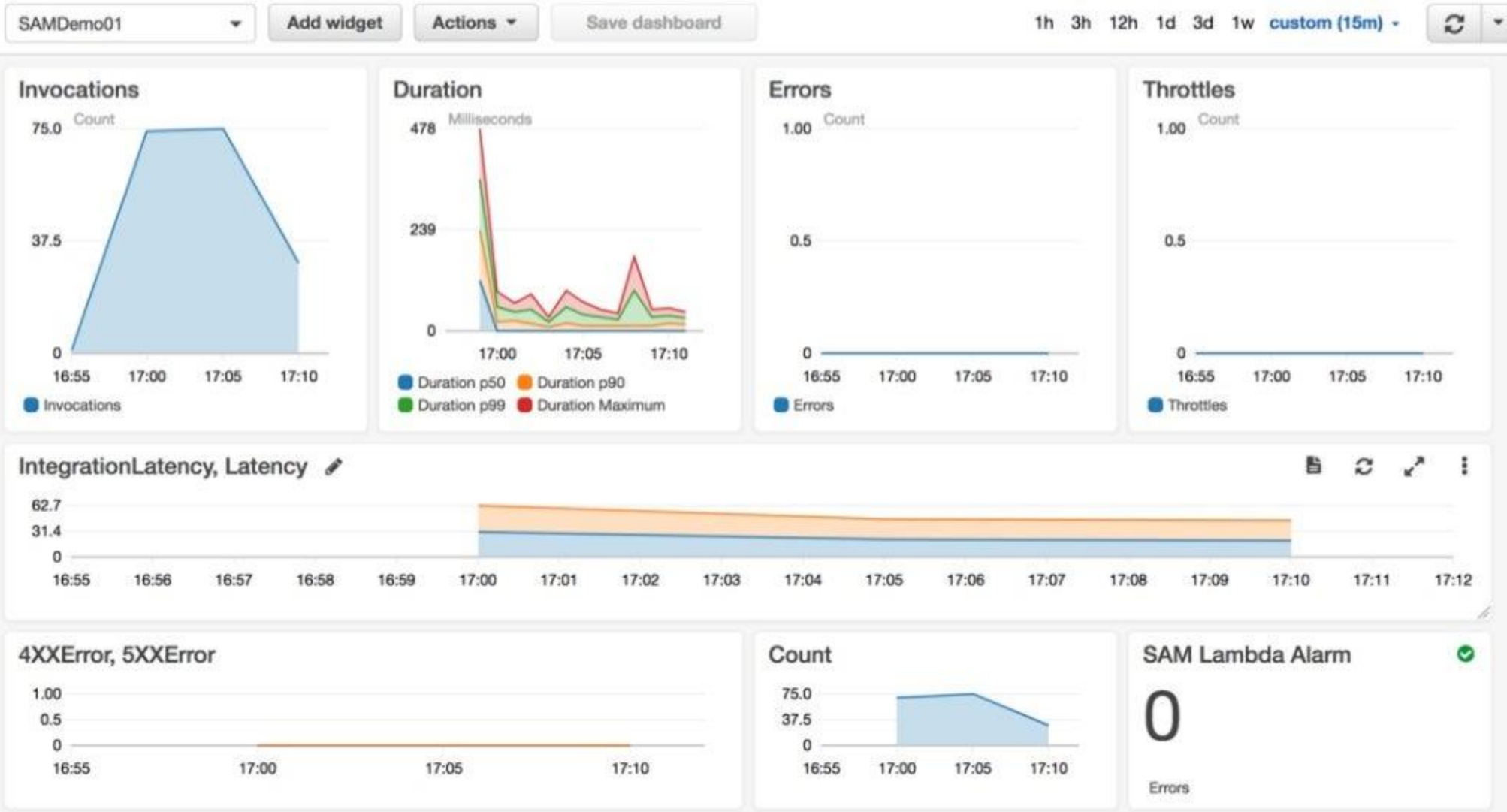
List of Log Variables

Save Changes

23:59:42 { "requestId":"da82692e-ce4e-11e7-ada2-6d3cd4a95886",

```
{
    "requestId": "da82692e-ce4e-11e7-ada2-6d3cd4a95886",
    "ip": "54.240.196.186",
    "caller": "-",
    "user": "-",
    "requestTime": "20/Nov/2017:23:59:32 +0000",
    "httpMethod": "GET",
    "resourcePath": "/pets",
    "status": "200",
    "protocol": "HTTP/1.1",
    "responseLength": "3"
}
```

Log Example


SAMDemo01
Add widget
Actions
Save dashboard
1h 3h 12h 1d 3d 1w custom (15m)
Invocations
Count
75.0
37.5
0
16:55 17:00 17:05 17:10
Invocations
Duration
Milliseconds
478
239
0
17:00 17:05 17:10
Duration p50 Duration p90
Duration p99 Duration Maximum
Errors
Count
1.00
0.5
0
16:55 17:00 17:05 17:10
Errors
Throttles
Count
1.00
0.5
0
16:55 17:00 17:05 17:10
Throttles
IntegrationLatency, Latency
62.7
31.4
0
16:55 16:56 16:57 16:58 16:59 17:00 17:01 17:02 17:03 17:04 17:05 17:06 17:07 17:08 17:09 17:10 17:11 17:12
4XXError, 5XXError
1.00
0.5
0
16:55 17:00 17:05 17:10
Count
75.0
37.5
0
16:55 17:00 17:05 17:10
SAM Lambda Alarm
0
Errors

AWS X-Ray

AWS X-Ray traces requests made to your application.

X-Ray collects data about the request from each of the underlying applications services it passes through.

X-Ray combines the data gathered from each service into singular units called traces.

View the service map to see trace data such as latencies, HTTP statuses, and metadata for each service.

Drill into the service showing unusual behavior to identify the root issue.

aws

Application instrumentation (Node.js)

```
1  var AWSXRay = require('aws-xray-sdk-core');
2  var AWS = AWSXRay.captureAWS(require('aws-sdk'));
3  s3 = new AWS.S3({signatureVersion: 'v4'});
4
5  exports.handler = (event, context, callback) => {
6
7      var params = {Bucket: 'tim-example-bucket', Key: 'MyKey', Body: 'Hello!'};
8
9      s3.putObject(params, function(err, data) {});
10 };
```


aws

X-Ray Service Map

X-Ray Trace Timeline

How do I figure out what's wrong?

These tools are here, so use them!

1. Turn on X-Ray now
    1. look at wrapping your own calls with it via the X-Ray SDKs
2. Don't underestimate the power of logging in Lambda
    1. Simple "debug: in functionX" statements work great and are easy to find in CloudWatch Logs
3. The most valuable metrics are the ones closest to your customer/use-case
    1. How many gizmos did this function call/create/process/etc

Lambda Dead Letter Queues

"By default, a failed Lambda function invoked asynchronously is retried twice, and then the event is discarded. Using Dead Letter Queues (DLQ), you can indicate to Lambda that unprocessed events should be sent to an Amazon SQS queue or Amazon SNS topic instead, where you can take further action." – https://docs.aws.amazon.com/lambda/latest/dg/dlq.html

- Turn this on! (for async use-cases)
- Monitor it via an SQS Queue length metric/alarm
- If you use SNS, send the messages to something durable and/or a trusted endpoint for processing
    - Can send to Lambda functions in other regions
- If and when things go "boom" DLQ can save your invocation event information

aws


Build & deploy your application
https://secure.flickr.com/photos/spenceyc/7481166880

Building a deployment package

Node.js & Python

- .zip file consisting of your code and any dependencies
- Use npm/pip to install libraries
- All dependencies must be at root level

Java

- Either .zip file with all code/dependencies, or standalone .jar
- Use Maven / Eclipse IDE plugins
- Compiled class & resource files at root level, required jars in /lib directory

C# (.NET Core)

- Either .zip file with all code / dependencies, or a standalone .dll
- Use NuGet / VisualStudio plugins
- All assemblies (.dll) at root level

Go

- .zip file consisting of your Go binary and any dependencies
- Use “go get” to install dependencies

aws

AWS CodeBuild

Fully managed build service that can compile source code, runs tests, and produces software packages

Scales continuously and processes multiple builds concurrently

Can consume environment variables from AWS SSM Parameter Store

NEW: Can run in your VPC

NEW: Supports dependency caching

buildspec.yml Example

```
version: 0.1

environment_variables:
  plaintext:
      "INPUT_FILE": "saml.yaml”
      "S3_BUCKET": ""

phases:
  install:
    commands:
      - npm install
pre_build:
    commands:
      - eslint *.js
build:
    commands:
      - npm test
post_build:
    commands:
      - aws cloudformation package --template $INPUT_FILE --s3-
bucket $S3_BUCKET --output-template post-saml.yaml
artifacts:
  type: zip
  files:
      - post-saml.yaml
      - beta.json
```

buildspec.yml Example

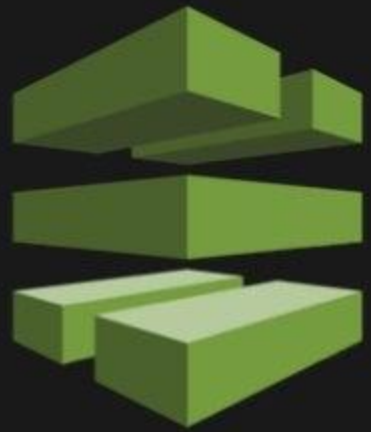
```
version: 0.1

environment_variables:
  plaintext:
      "INPUT_FILE": "saml.yaml”
      "S3_BUCKET": ""

phases:
  install:
    commands:
      - npm install
pre_build:
    commands:
      - eslint *.js
build:
    commands:
      - npm test
post_build:
    commands:
      - aws cloudformation package --template $INPUT_FILE --s3-
bucket $S3_BUCKET --output-template post-saml.yaml
artifacts:
  type: zip
  files:
      - post-saml.yaml
      - beta.json
```

- Variables to be used by phases of build
- Examples for what you can do in the phases of a build:
  - You can install packages or run commands to prepare your environment in ”install”.
  - Run syntax checking, commands in “pre_build”.
  - Execute your build tool/command in “build”
  - Test your app further or ship a container image to a repository in post_build
- Create and store an artifact in S3

AWS CodePipeline



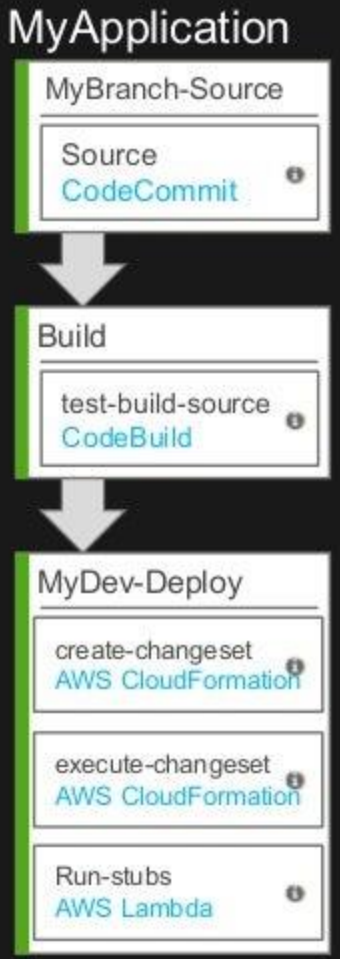
Continuous delivery service for fast and reliable application updates

Model and visualize your software release process

Builds, tests, and deploys your code every time there is a code change

Integrates with third-party tools and AWS

An example minimal Developer's pipeline:



This pipeline:

- Three Stages
- Builds code artifact
- One Development environment
- Uses SAM/CloudFormation to deploy artifact and other AWS resources
- Has Lambda custom actions for running my own testing functions

Lambda Environment Variables

- Key-value pairs that you can dynamically pass to your function
- Available via standard environment variable APIs such as process.env for Node.js or os.environ for Python
- Can optionally be encrypted via AWS Key Management Service (KMS)
  - Allows you to specify in IAM what roles have access to the keys to decrypt the information
- Useful for creating environments per stage (i.e. dev, testing, production)

API Gateway Stage Variables

- Stage variables act like environment variables
- Use stage variables to store configuration values
- Stage variables are available in the $context object
- Values are accessible from most fields in API Gateway
  - Lambda function ARN
  - HTTP endpoint
  - Custom authorizer function name
  - Parameter mappings

aws

AWS Systems Manager – Parameter Store

Centralized store to manage your configuration data

- supports hierarchies
- plain-text or encrypted with KMS
- Can send notifications of changes to Amazon SNS/ AWS Lambda
- Can be secured with IAM
- Calls recorded in CloudTrail
- Can be tagged
- Available via API/SDK

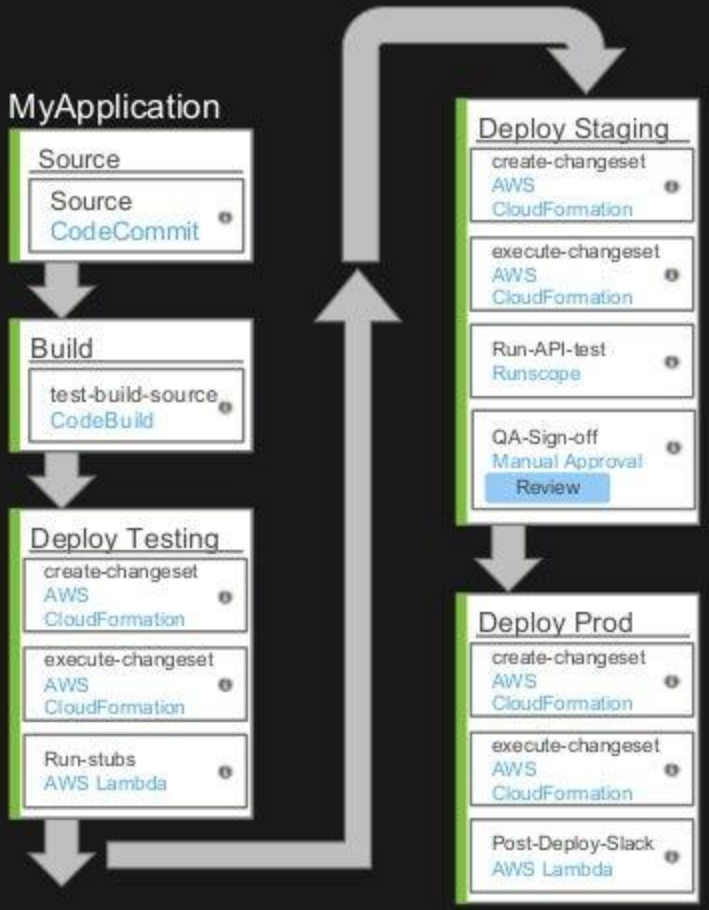
Useful for: centralized environment variables, secrets control, feature flags

```
from __future__ import print_function
import json
import boto3
ssm = boto3.client('ssm', 'us-east-1')

def get_parameters():
    response = ssm.get_parameters(
        Names=['LambdaSecureString'],WithDe
cryption=True
    )
    for parameter in
response['Parameters']:
        return parameter['Value']

def lambda_handler(event, context):
    value = get_parameters()
    print("value1 = " + value)
    return value  # Echo back the first key
value
```

An example pipeline that goes to production:



This CodePipeline pipeline:

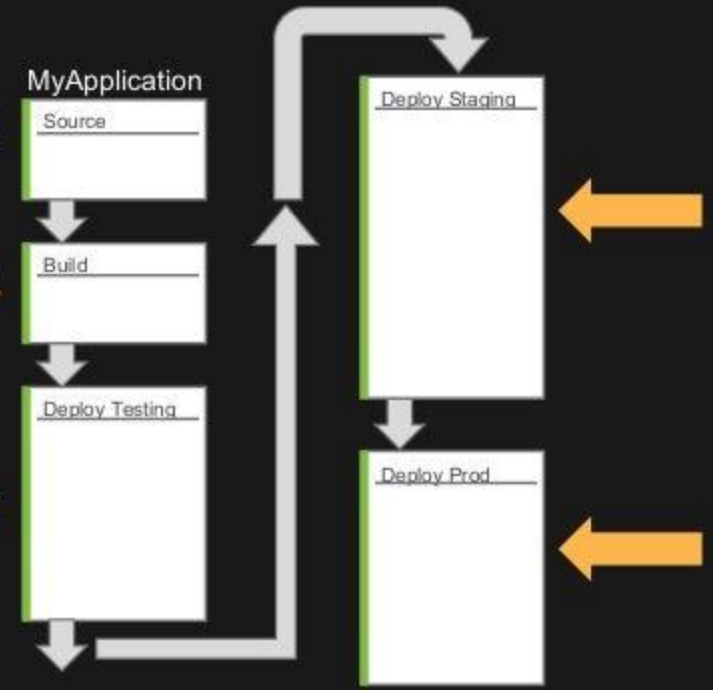
- Five Stages
- Builds code artifact w/ CodeBuild
- Three deployed to “Environments”
- Uses SAM/CloudFormation to deploy artifact and other AWS resources
- Has Lambda custom actions for running my own testing functions
- Integrates with a 3rd party tool/service
- Has a manual approval before deploying to production

Where and what to test?

1.
- Code review via Pull Requests
- (NEW In CodeCommit)

2.
- Lint/syntax check
- Unit tests pass
- Code successfully compiles

3.
- Application deploys successfully
- Mocked/stubbed integration tests



4.
- Application deploys successfully
- Tests against real services (potentially against production dependencies)

5.
- Deploy canaries
- Complete wait period successfully
- Deploy 100%

Environments, Stages, Versioning, & Canaries?

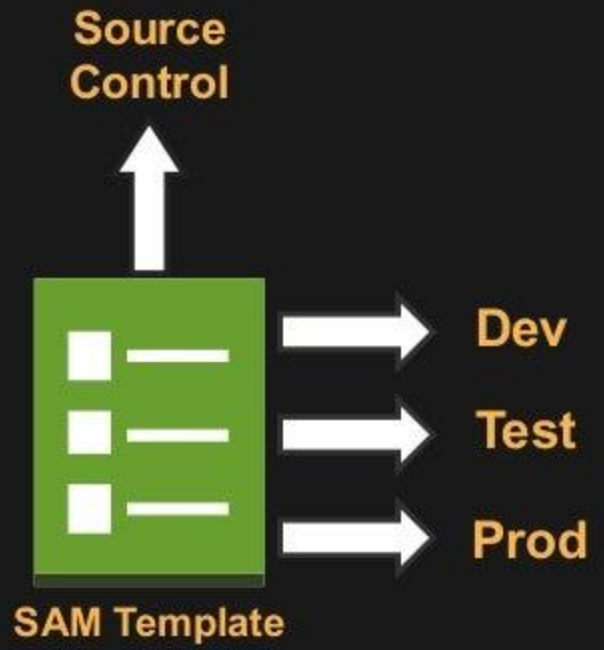
A few best practices:

1. Use blue|green or canaries for production deployments with a rollback as automated as possible
2. In Lambda Versioning is useful if you need to support multiple versions to multiple consumers/invocation points
3. In API-Gateway Stages work similarly and are useful if you need to support multiple API versions
4. Try to always have separate “stacks” for Development, Testing, Staging, Production environments
    1. Do not use Stages or Versioning for this
    2. Think about having different accounts all together for different environments

aws

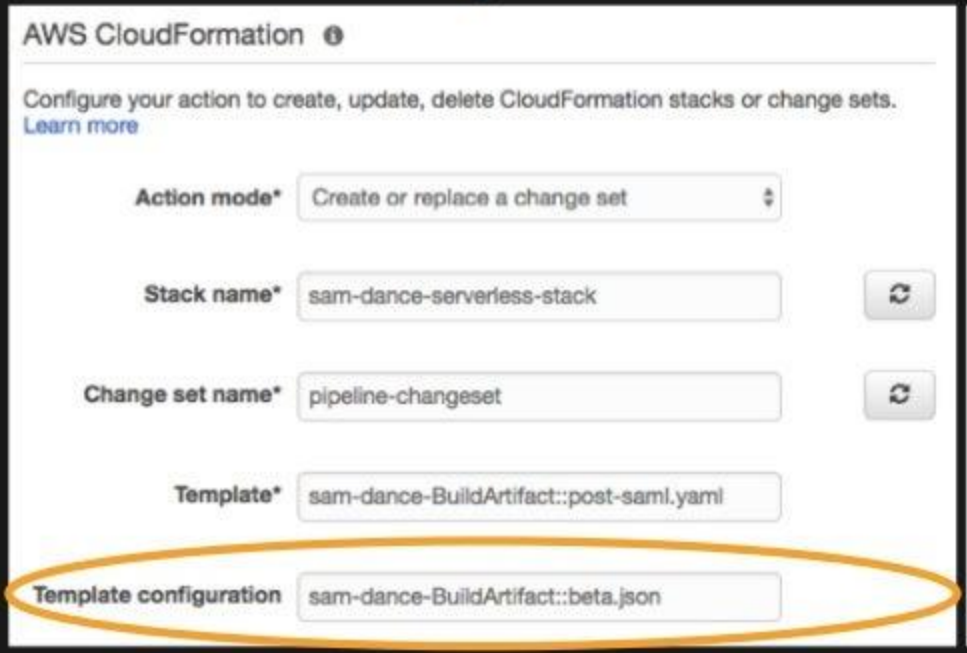
SAM Best Practices

- Use Parameters and Mappings when possible to build dynamic templates based on user inputs and pseudo parameters such as AWS::Region
- Use the Globals section to simplify templates
- Use ExportValue & ImportValue to share resource information across stacks
- Build out multiple environments, such as for Development, Test, Production and even DR using the same template, even across accounts

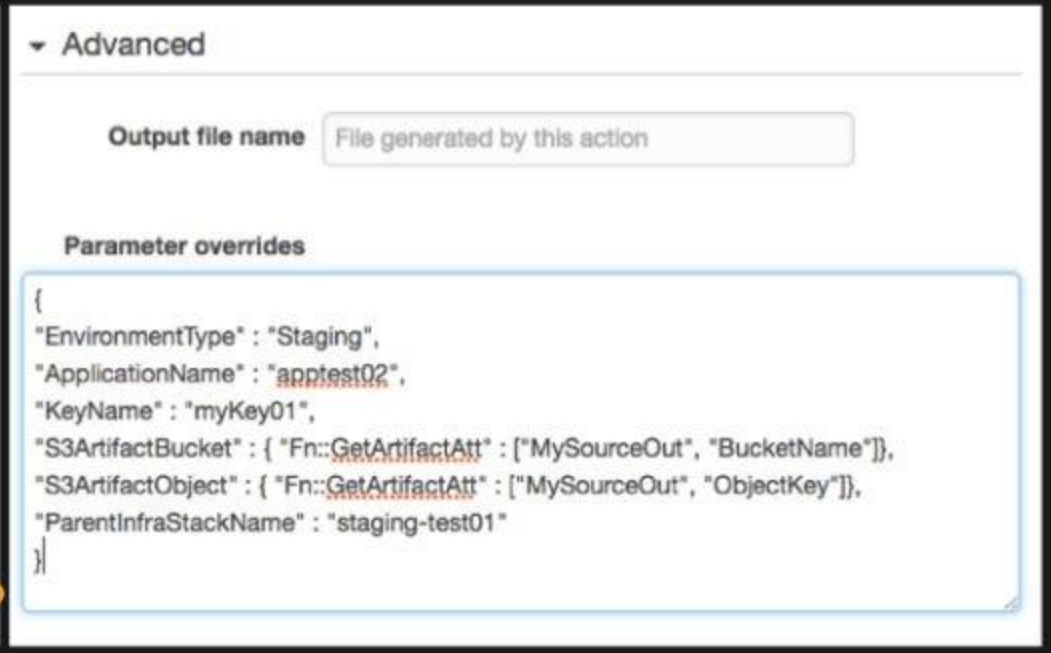


CodePipeline + CloudFormation Parameters

Via referenced parameter file:



Via Parameter Overrides:



Start with AWS CodeStar

Best Practices

aws

Lambda Function Best Practices

- Unless function handlers share code, split them into their own independent Lambda functions files or binaries
    - Another option is to use language specific packages to share common code between functions

- Unless independent Lambda functions share event sources, split them into their own code repositories with their own SAM templates

- Locally validate your YAML or JSON SAM files before committing them. Then do it again in your CI/CD process

aws

Computer power

> Memory > Cores

Lambda exposes only a memory control, this also affects the **% of CPU core** allocated to a function

Is your code CPU, Network or memory-bound, it could be **cheaper** to give more memory

aws

Less memory is not always cheaper

Stats for Lambda function that calculates **1000 times** all prime numbers **<= 1000000**

128mb	11.722965sec	$0.024628
256mb	6.678945sec	$0.028035
512mb	3.194954sec	$0.026830
1024mb	1.465984sec	$0.024638

aws

Cold start: Understand the function lifecycle

Separate business logic from function signature

```
app = Todo()

def lambda_handler(event, context):
    ret = app.dispatch(event)

    return {
        'statusCode': ret["status_code"],
        'headers':    ret["headers"],
        'body':       json.dumps(ret["body"])
    }
```


aws

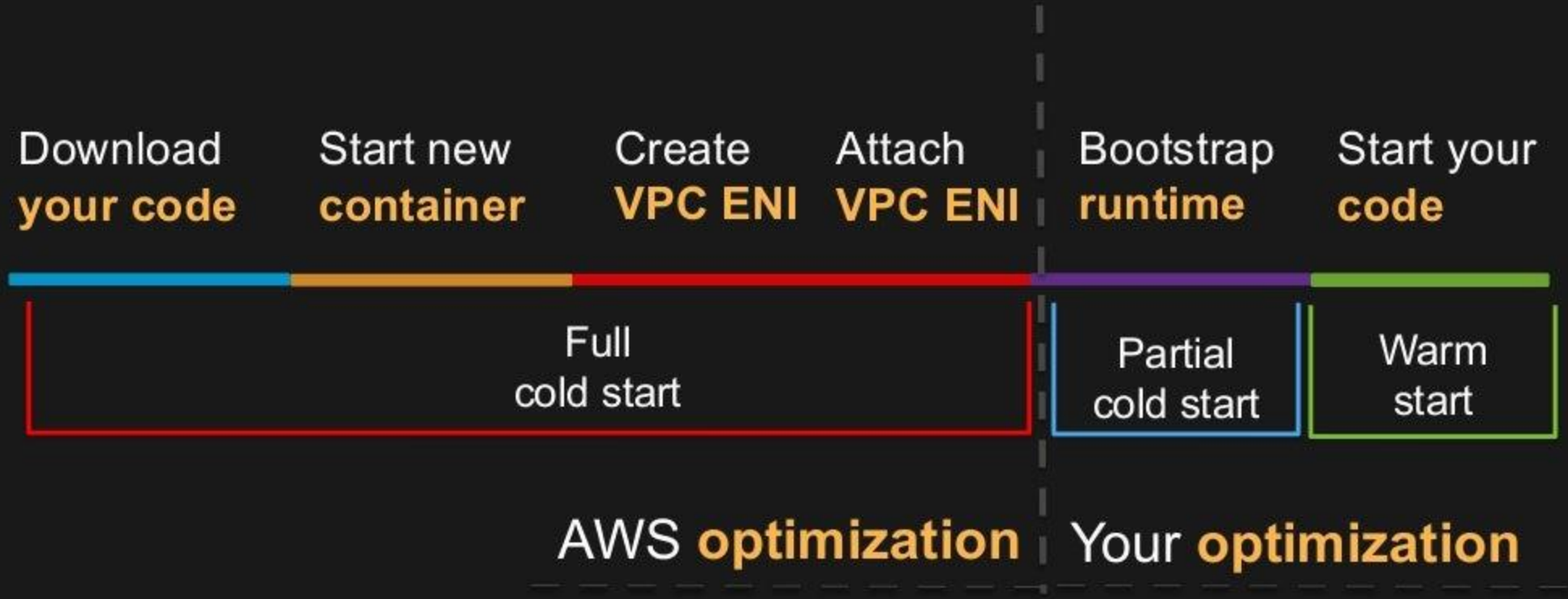
Leverage container reuse

- Lazily load variables in the **global scope** – functions stay warm for several minutes
- Don't load it if you don't need it – **cold starts** are affected

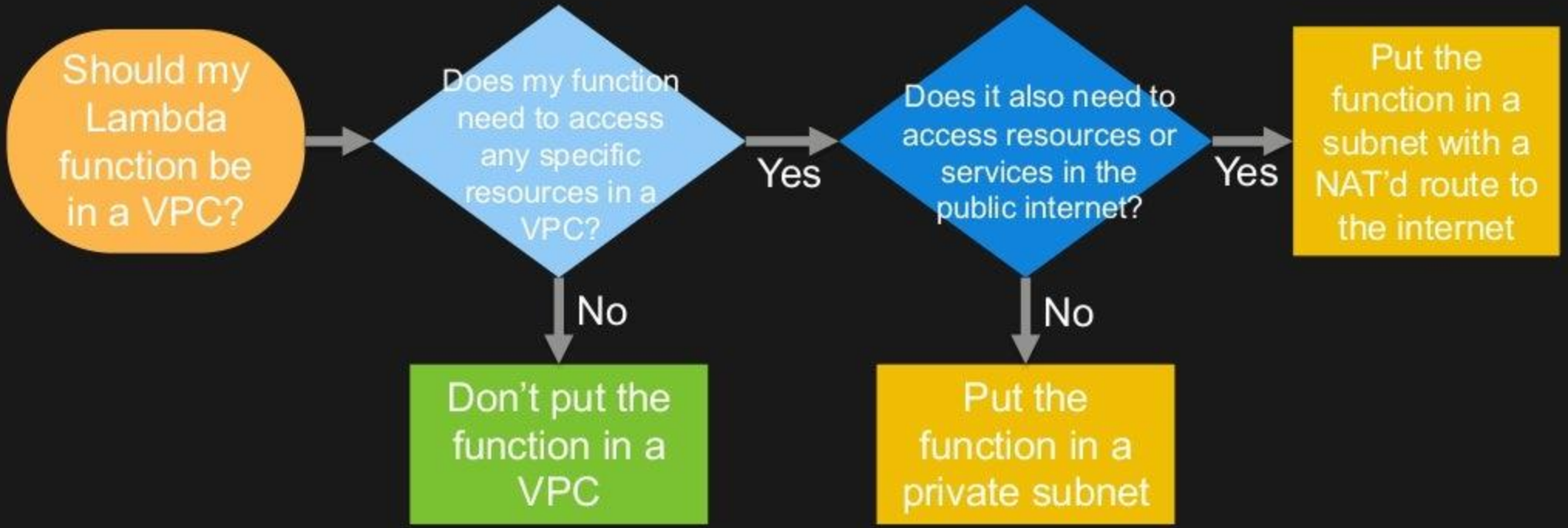
```
s3 = boto3.resource('s3')
db = db.connect()

def lambda_handler(event, context):
    global db
    # verify if still connected
    # otherwise carry on
    if not db:
        db = db.connect()
...
```

Cold start: Understand the function lifecycle (VPC)



Do I need to use VPC?



Amazon Virtual Private Cloud (VPC)

Provision a logically isolated section of the AWS cloud where you can launch resources in a virtual network that you define

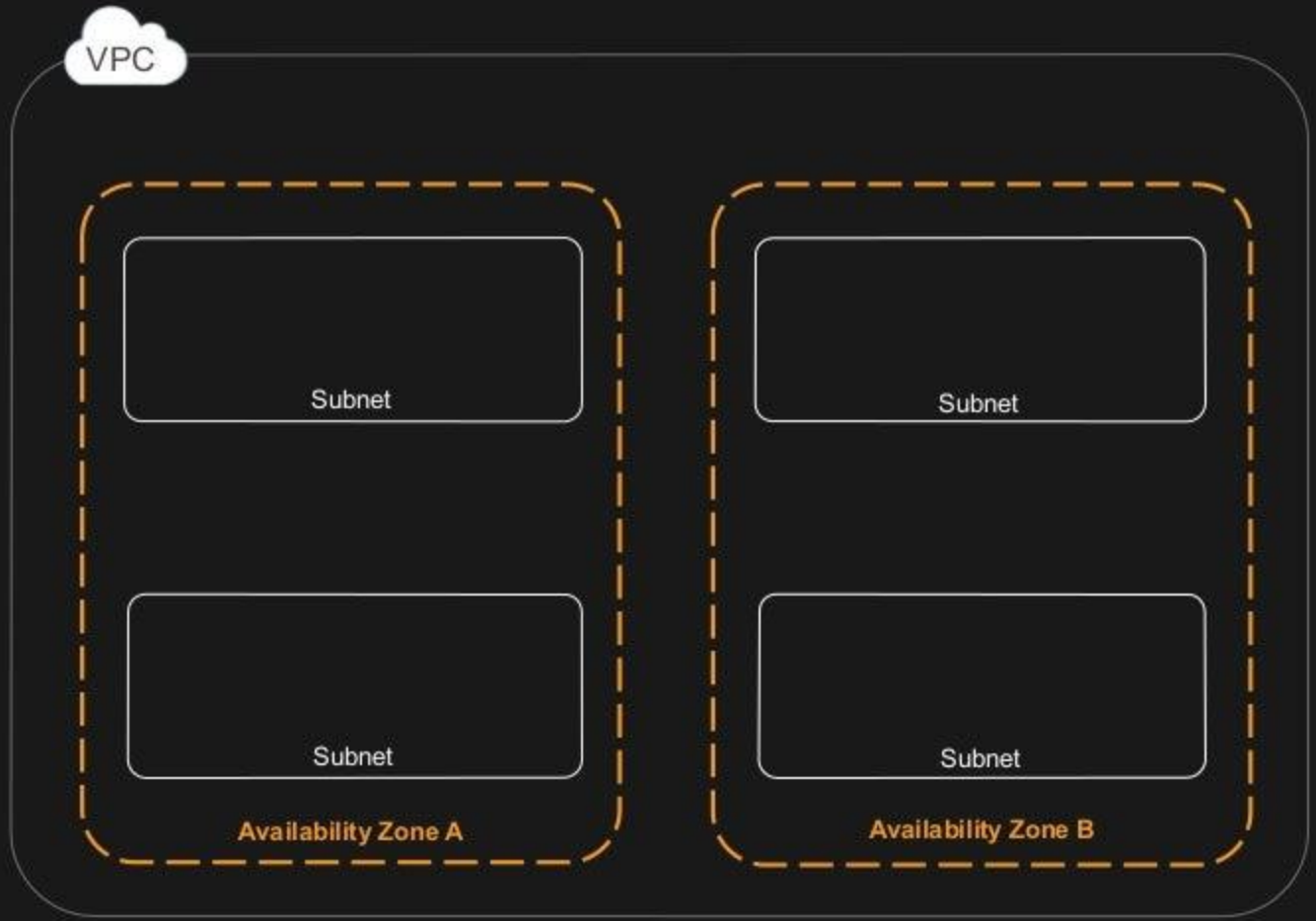
Create public and/or private networks

Can use subnets, network ACLs, route tables to define your network based on your needs

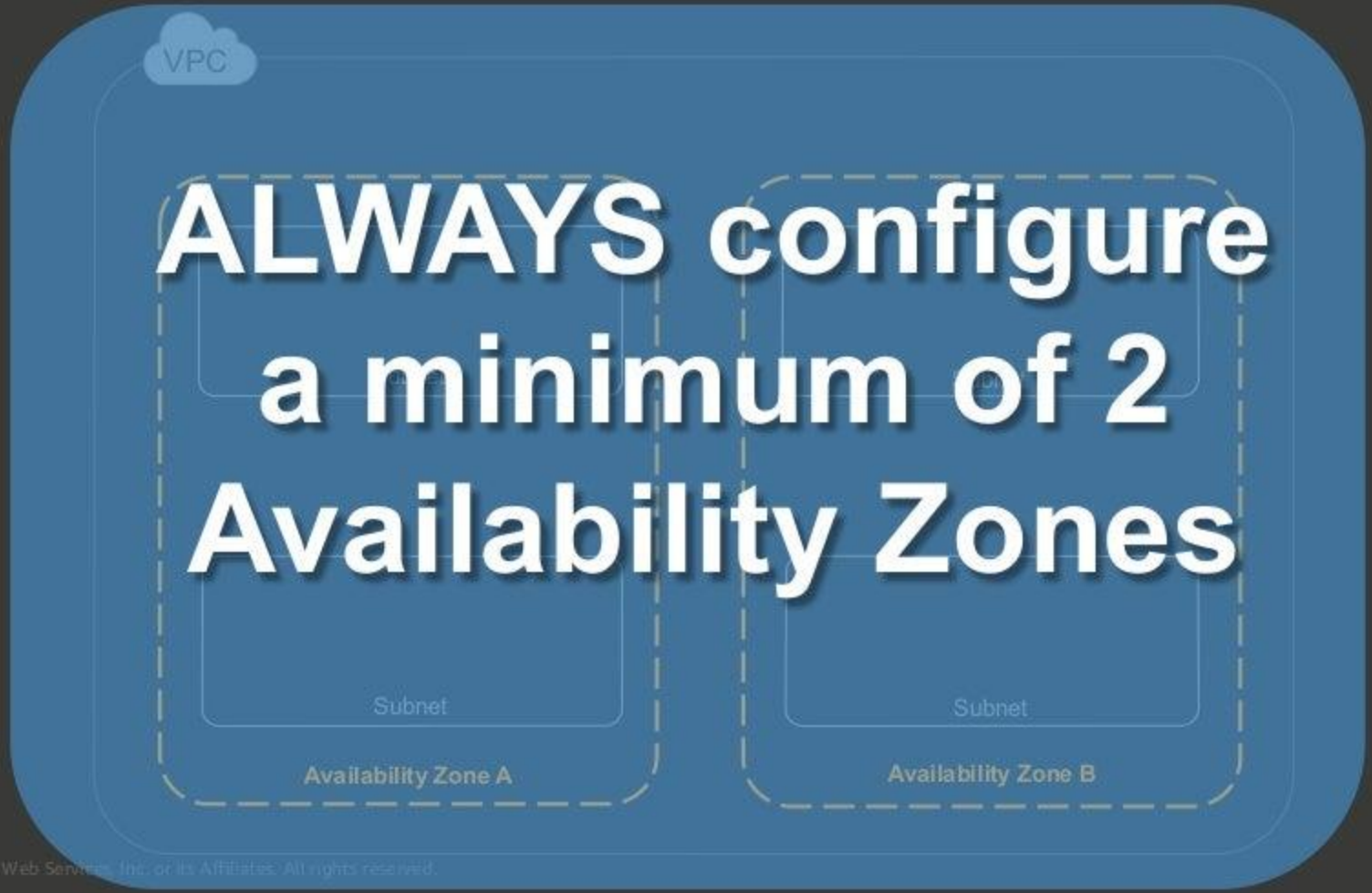
Can connect VPCs to each other with VPC Peering, to on-prem with VPNs or Direct Connect

aws

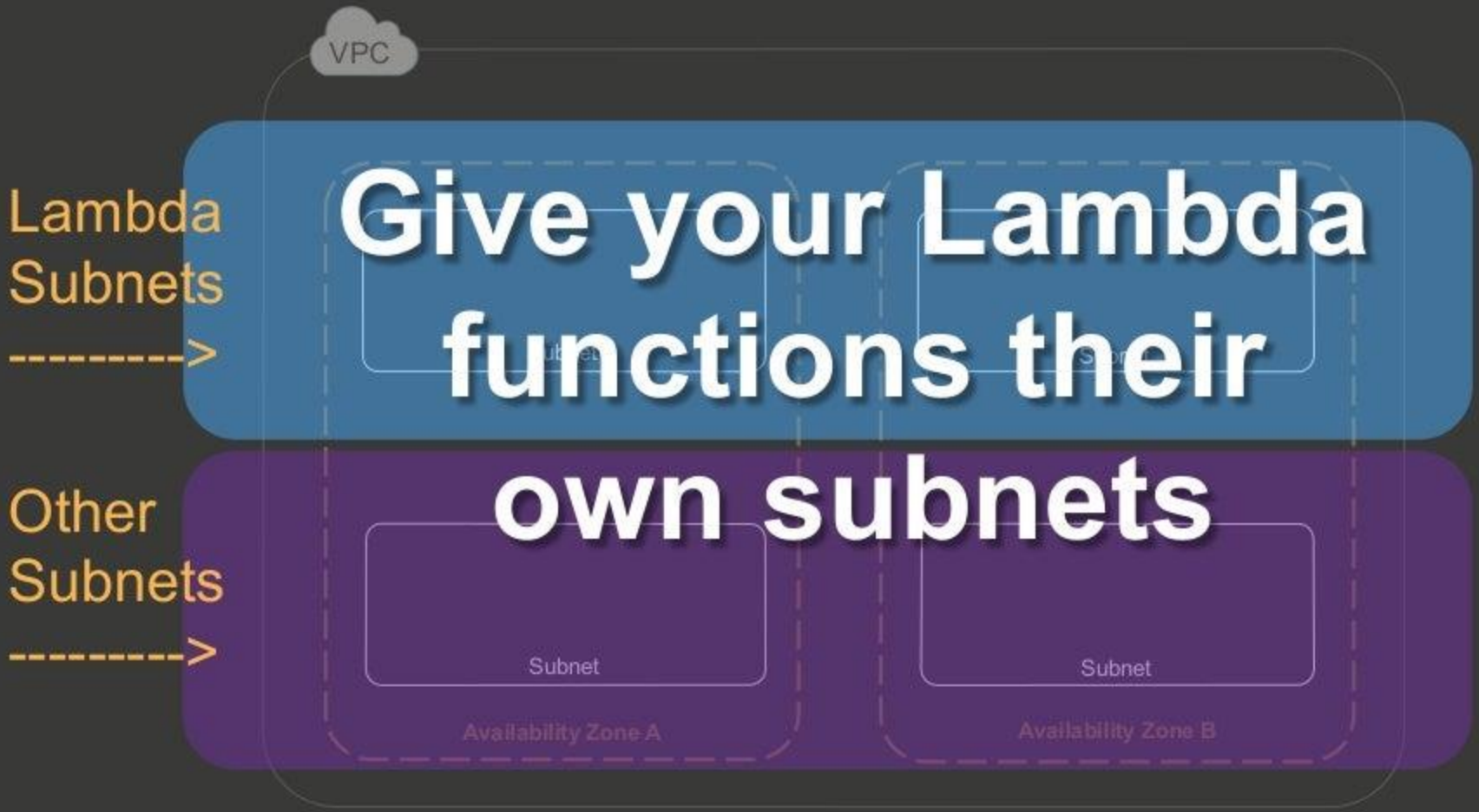
Basic VPC Design



Basic VPC Design

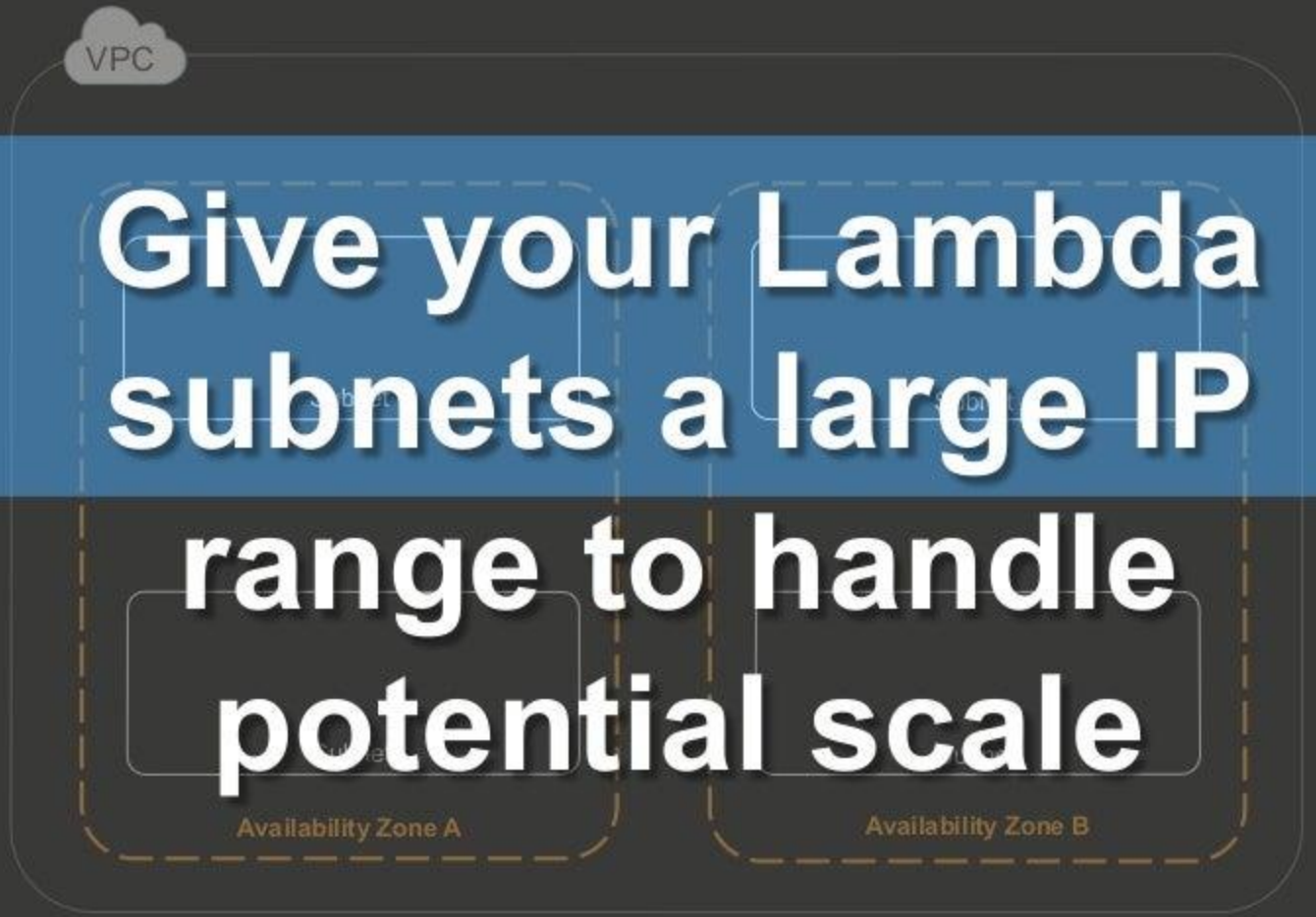


Basic VPC Design



Give your Lambda functions their own subnets

Basic VPC Design



Give your Lambda subnets a large IP range to handle potential scale

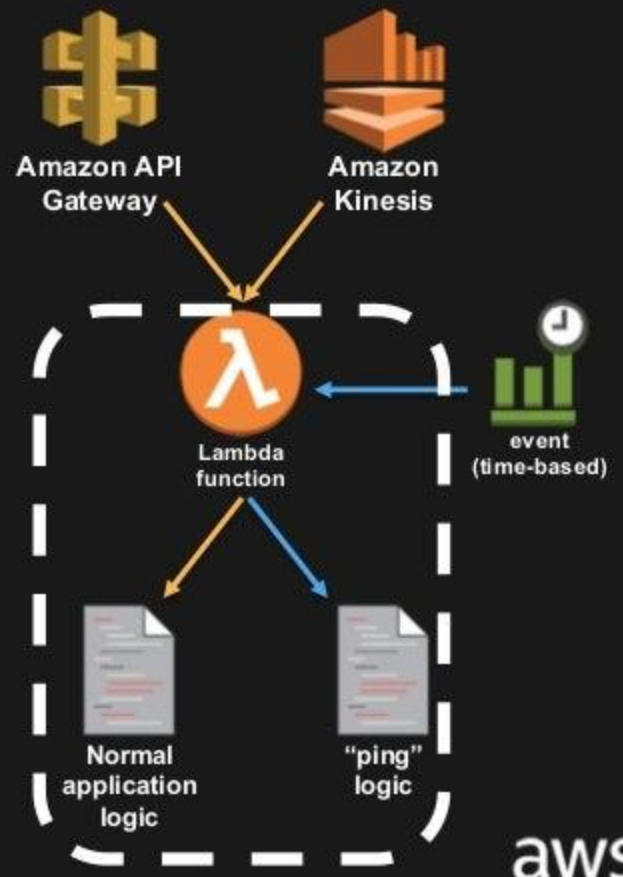
Basic VPC Design

If your functions need to talk to a resource on the internet, you need a NAT!

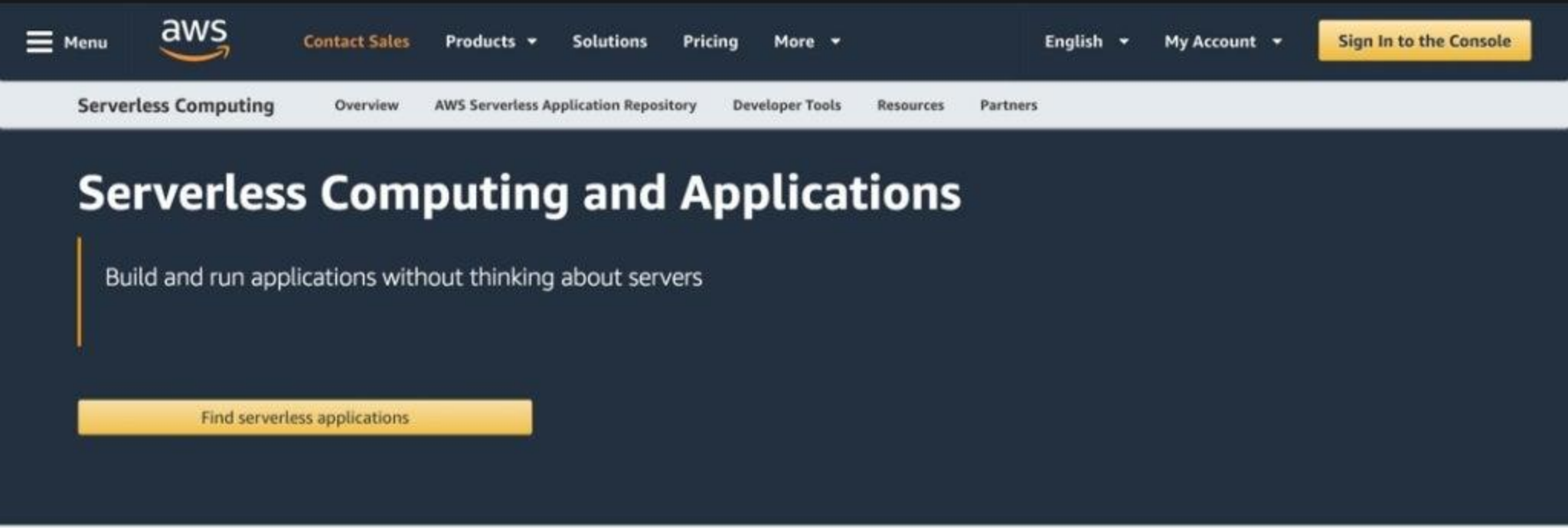
CloudWatch Events ”ping” hack

Largely unnecessary, but if cold starts have a visible impact on your overall performance:

- Use CloudWatch Events’ scheduled events to invoke (“ping”) a Lambda function via API call to the Lambda service
    - DO NOT add an unnecessary API Gateway, for example
- Pass in a payload that you can test for as not a real payload
- Have a function in your code that handles and replies accordingly



aws.amazon.com/serverless



Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for nearly any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.

Building serverless applications means that your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

Hal Stiles
stilesh@amazon.com
@halstiles

https://www.flickr.com/photos/theredproject/3302110152/

https://secure.flickr.com/photos/dullhunk/202872717/