



SERVERLESS SERVICES ON AWS

Mikael Puittinen, Chief Technology Officer

13.1.2017

SC5 BRIEFLY



CLOUD
SOLUTIONS



BUSINESS
APPLICATIONS



INTELLIGENT
SOLUTIONS



DIGITAL
DESIGN

10
YEARS

100+
CUSTOMERS

400+
PROJECTS

85
HACKERS
DESIGNERS

HEL
JKL

~7
MEUR
2016



+ OTHERS

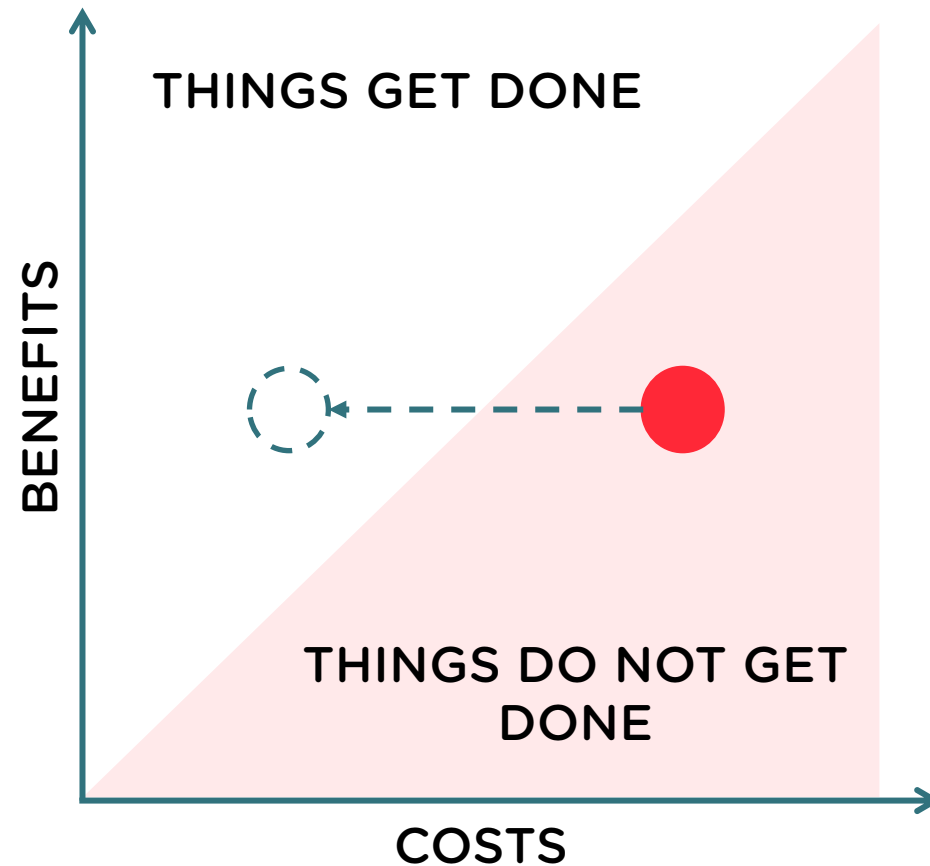


VISIT OUR WEB SITE FOR MORE INFO: [HTTPS://SC5.IO](https://sc5.io)

WHY SERVERLESS?

- Focus on core functionality rather than scaffolding servers (hw/sw)
 - Faster time to value
 - It is more fun
- More cost efficient operations (in some cases)

DEVELOPMENT EFFICIENCY AS AN ENABLER



CORE AWS SERVERLESS SERVICES

Building blocks for digital services

SOME SERVERLESS AWS SERVICES

STORAGE

 S3 (Simple Storage Service) 2006

DATABASE

 DynamoDB 2012

COMPUTE

 Lambda 2014

INTERNET OF THINGS

 IoT 2015

MOBILE SERVICES

 Simple Notification Service 2010

 Cognito 2014

APPLICATION SERVICES

 API Gateway 2015

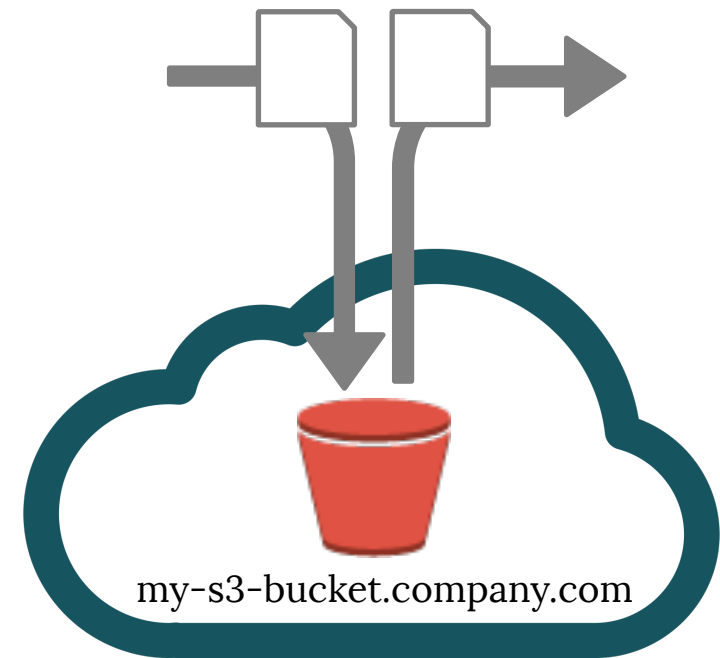
 SQS (Simple Queue Service) 2006

ANALYTICS

 Machine Learning 2015

SIMPLE STORAGE SERVICE (S3)

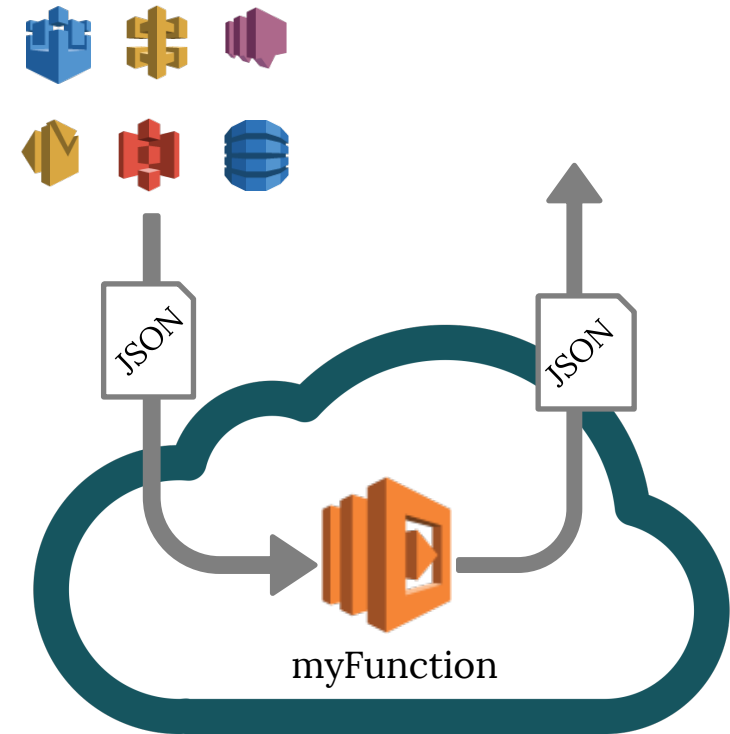
- (Unlimited) file storage service
- Application internal files (user file uploads / downloads)
- Static web content (e.g. application HTML / CSS / JS / image assets)
- Can be complemented with CloudFront CDN to optimize costs and performance



PRICING: Storage volume + amount of requests

AWS LAMBDA

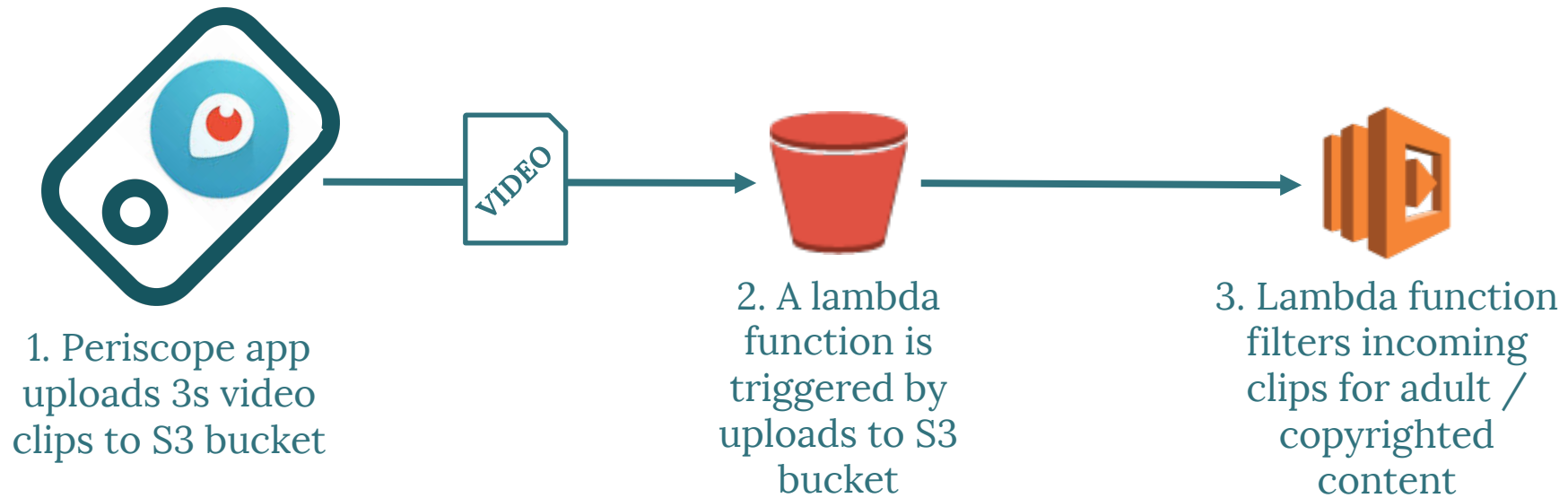
- Compute service for running code (functions) in AWS
- Event driven (API Gateway, SNS, SES, S3, DynamoDB, Schedule, ...)
- Provision memory & max time required by single function run
- Additional "instances" spawned automatically (cold / hot start)



PRICING: Utilized gigabyteseconds (rounded to 100ms)

Example

PERISCOPE CONTENT FILTER



EXERCISE: LAMBDA ECHO

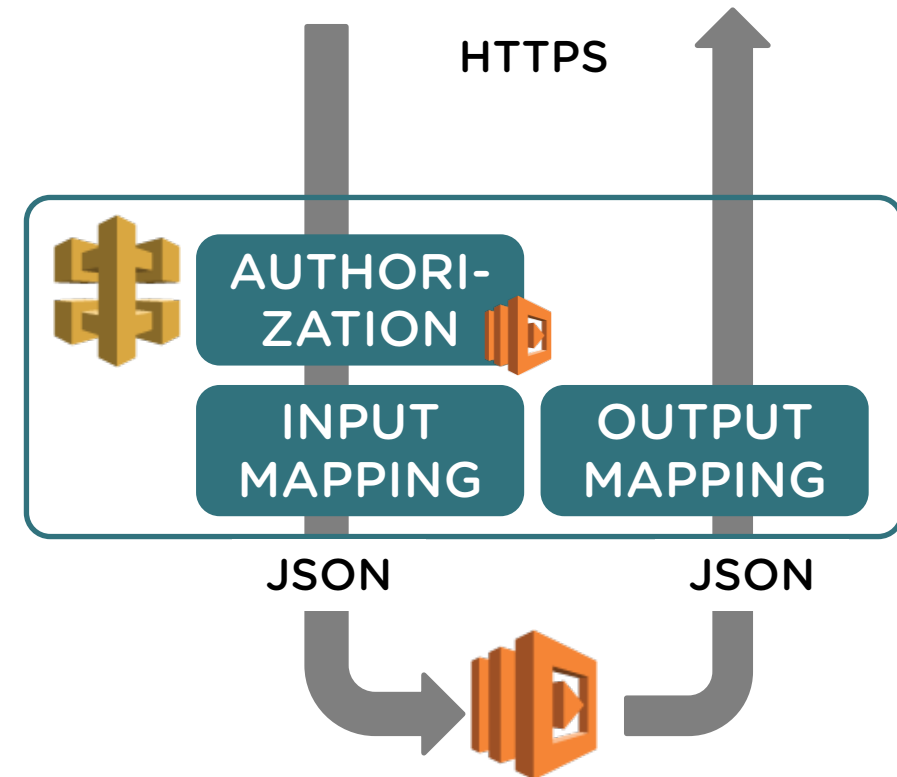
1. Open AWS Console / Lambda
2. Create new function based on the “Hello World” template
3. Name: “Echo”
4. Copy code from the right
5. Role: “*Create new role from template*”
6. Select policy “*Simple Microservice*”
7. Once created, test with some JSON input
8. See logs in Cloudwatch

```
'use strict';

exports.handler = (event,
context, callback) => {
    event.now = new Date();
    console.log('Received
event:', JSON.stringify(event,
null, 2));
    callback(null, event);
};
```

API GATEWAY

- AWS Service to implement REST (and other) APIs
- Security via API Keys, custom authorizers (Lambda)
- Connect to e.g. Lambda to publish your functions as REST interfaces
- Input / Output mapping (e.g. URL parameters -> JSON)
- No need for provisioning



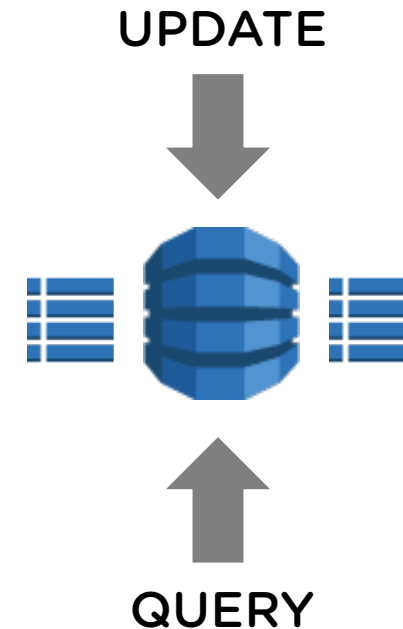
PRICING: # of requests + data transfer + cache size

EXERCISE: API GATEWAY

1. Launch API Gateway from AWS Console
2. Create API "Echo"
3. Create resource "echo" (from Actions)
4. Create "POST" method for resource "echo"
5. Integration type: Lambda Function
6. Deploy API to stage "v1"
7. Copy URL displayed for resource
8. Test API with e.g. Postman / curl
9. `curl --data '{"foo":"bar2"}' https://iydf5xghs8.execute-api.us-east-1.amazonaws.com/dev/echo`

DYNAMODB

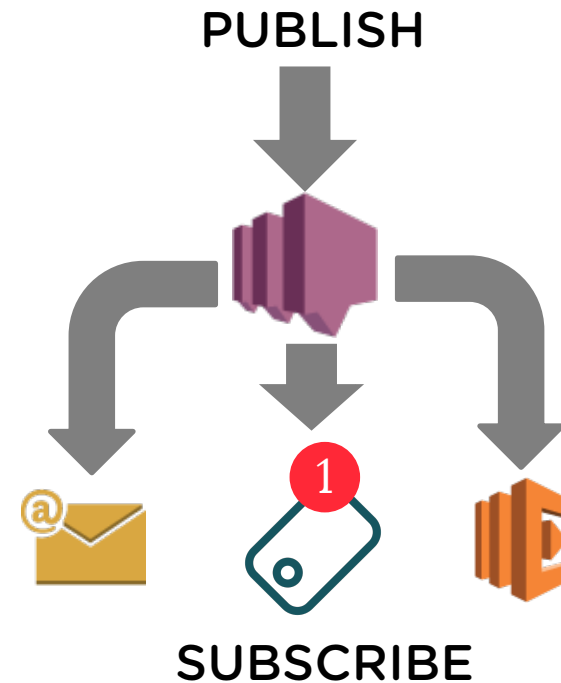
- noSQL database provided by AWS
- noSQL: scalable non-relational database with focus on speed
- Work with tables and indices, no server instances to manage
- Need to provision read / write capacity per table / index



PRICING: Provisioned read / write capacity and storage (over 25Gb)

SIMPLE NOTIFICATION SERVICE (SNS)

- Push notification service
- Mainly targeted for mobile notifications
- Can also be used for triggering e.g. Lambda functions, mobile, email notifications



PRICING: Amount of messages

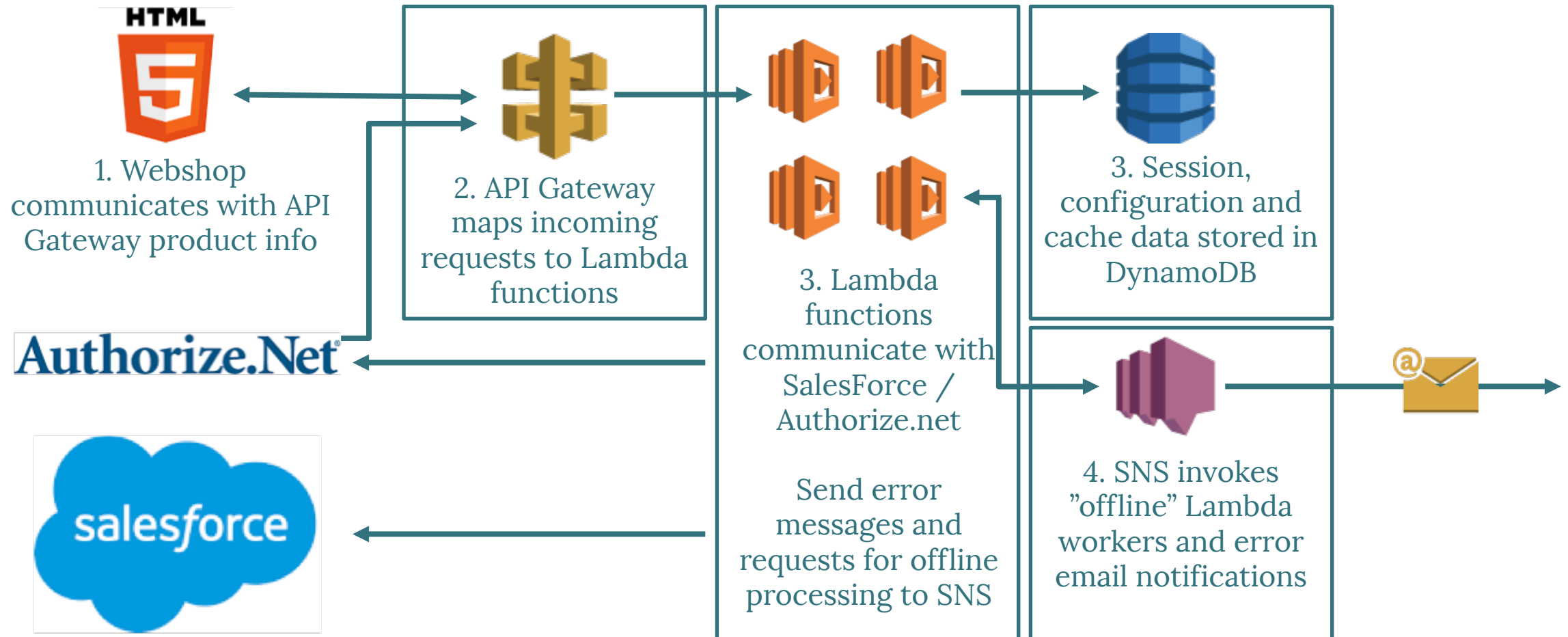
EXERCISE: SNS

1. In AWS Console, go to Services -> SNS
2. Create new topic "SNSTestTopic"
3. Create a subscription for the Lambda function created earlier
4. Publish something to the topic
5. Go to Services -> CloudWatch
6. Open Logs for the Lambda function

Logs show the SNS message sent above (and the messages from earlier tests)

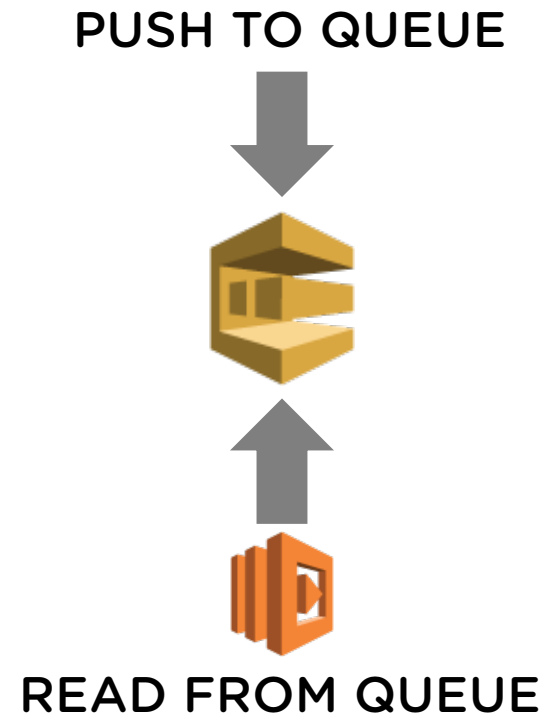
Example

HAPPY OR NOT WEBSHOP



SIMPLE QUEUE SERVICE (SQS)

- Message queue service (pull)
- Delivery guaranteed
- Each message handled only once
- Cannot currently trigger Lambda (Lambda needs to read queue e.g. with scheduled event)

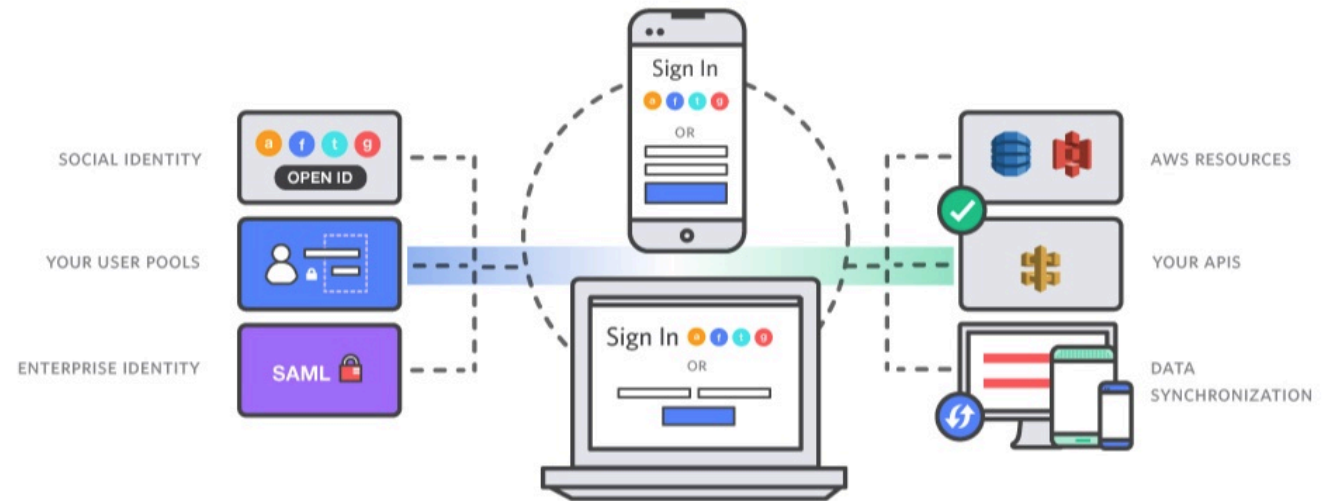


PRICING: Amount of messages

Serverless user management

COGNITO

- User sign-up / sign-in as a service
- Email / Phone verification
- Own user pool
- Federation through social identity providers
- Multifactor authentication
- ...



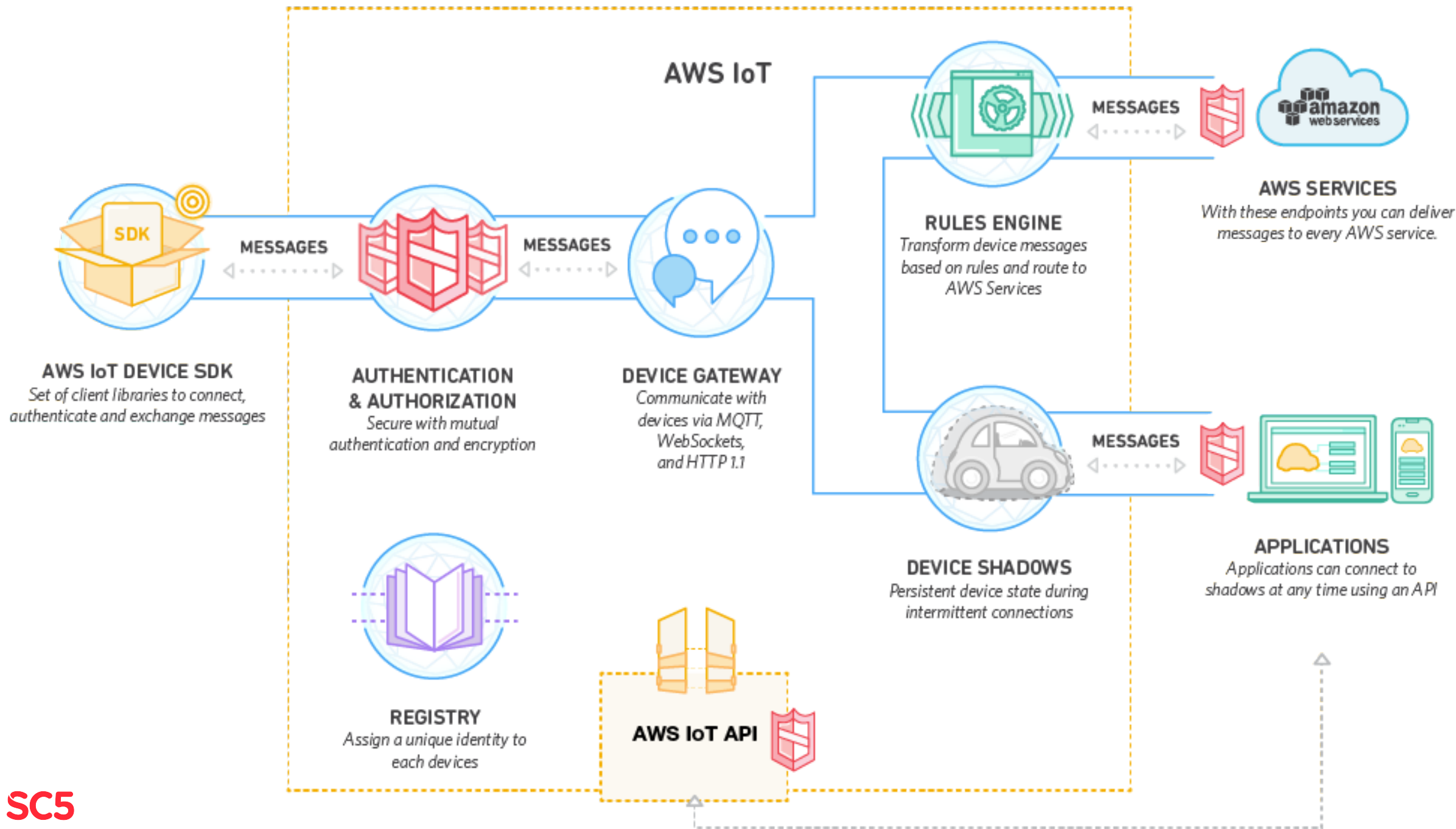
PRICING: # of Monthly Active Users

Push notifications

AWS IOT

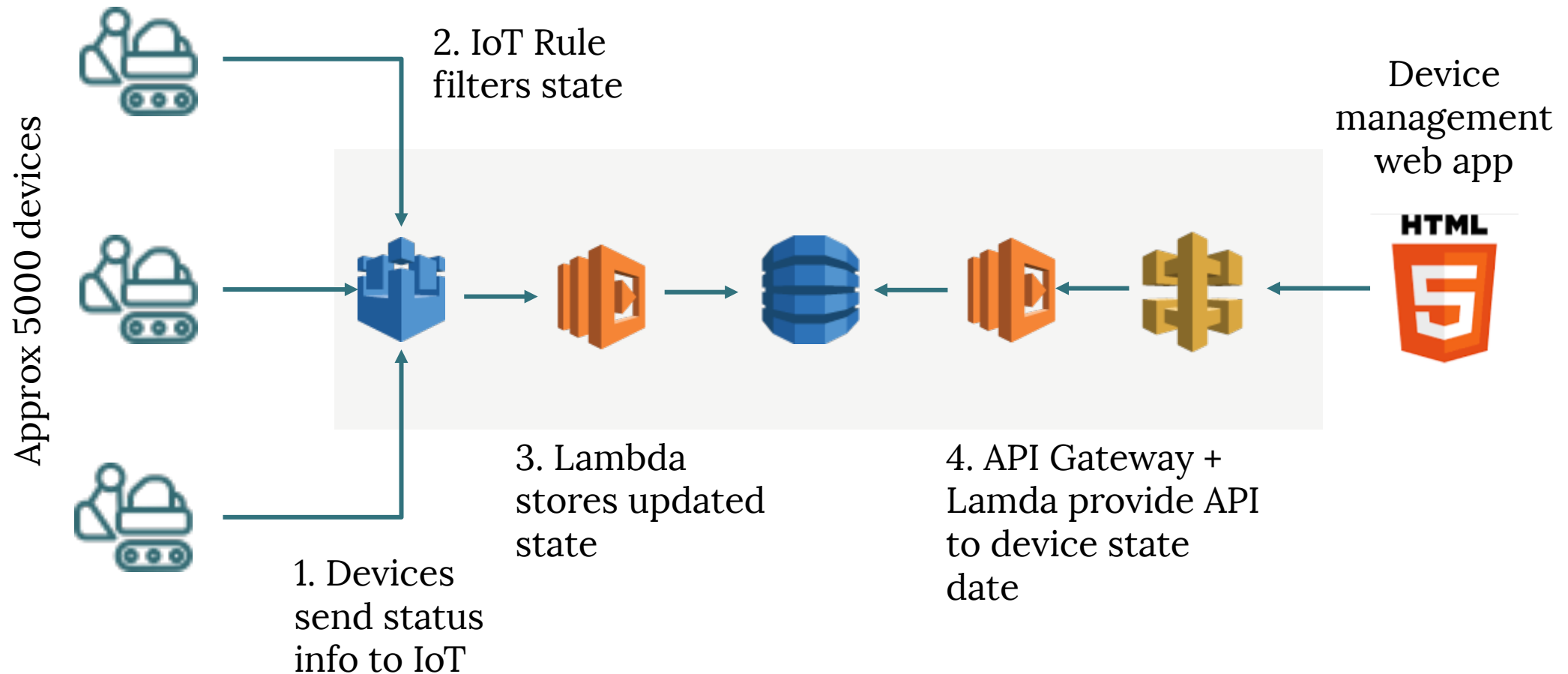
- Device registry + API for communicating with devices + automated actions (Rules)
- Authentication of devices
- Devices can send and retrieve (desired) state over MQTT
- Can perform actions based on rules (e.g. Temperature reading from a specific sensor is out of bounds)

PRICING: # of messages



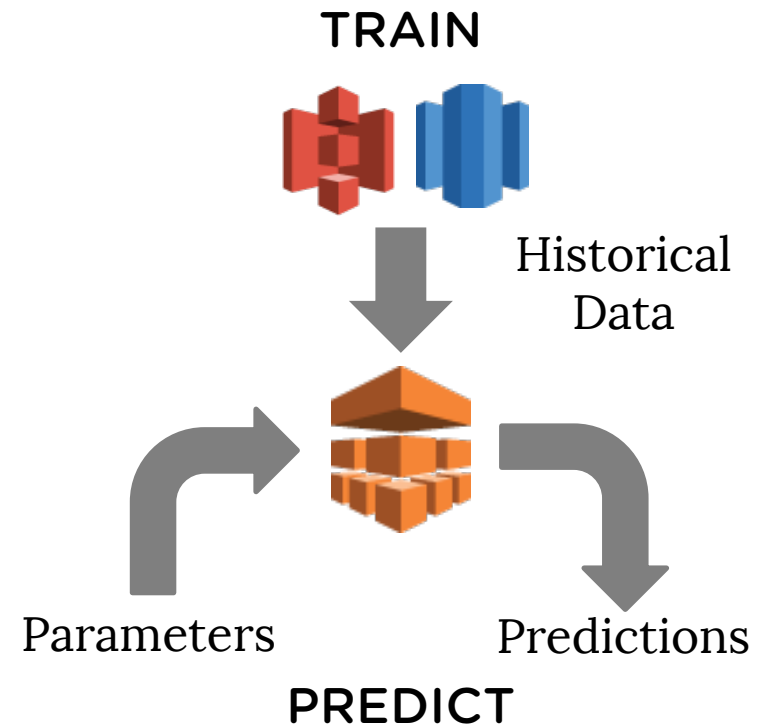
Example: IoT

DEVICE MANAGEMENT CASE



MACHINE LEARNING

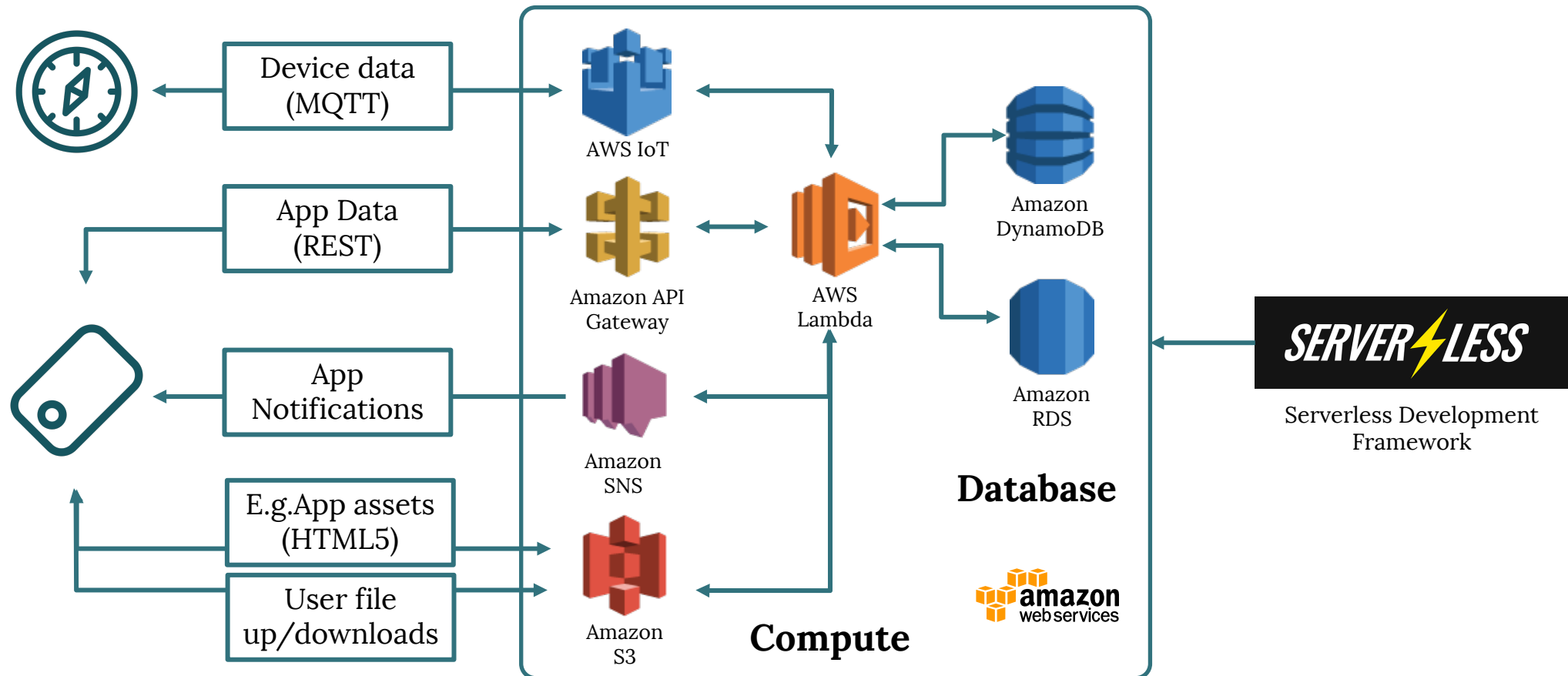
- Machine learning as a service
- Create & review ML models
- Solve binary classification, multi-class classification or regression prediction problems
- Batch & real time predictions



PRICING: (training) CPU time + # of predictions

Reference Architecture

CLOUD NATIVE APPLICATION ARCHITECTURE TOOLS À LA SC5



**AWS IS NOT THE ONLY ONE DOING
SERVERLESS**

How about other cloud vendors?

SIMILAR OFFERING ALSO FROM OTHER CLOUD VENDORS



AZURE FUNCTIONS
(PREVIEW)



OPENWHISK
(BETA)



GOOGLE CLOUD
FUNCTIONS
(ALPHA)

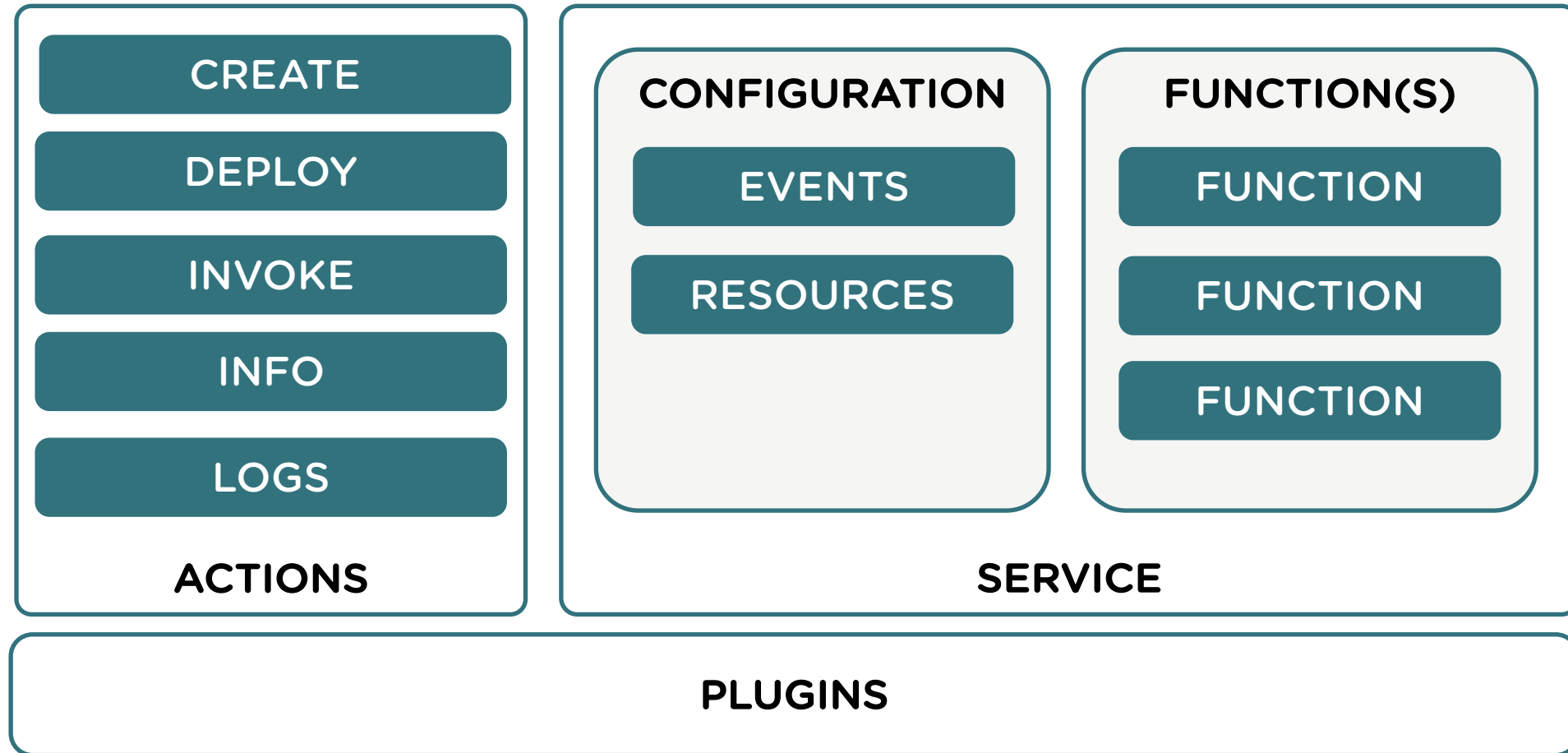
OVERVIEW OF DEVELOPMENT TOOLS FOR SERVERLESS APPS

NATIVE AWS TOOLS

- Current AWS tools stack for serverless is not really developer friendly
- Development via Console or CLI laboursome (need to define and manage IAM roles, build packages, etc. separately)
- CloudFormation can be used to automate the deployment, but packaging of Lambdas etc.. still needs to be automated
- Recommend to use 3rd party platforms for developing serverless solutions on AWS to gain full benefit of serverless technologies

SERVER  *LESS*

SERVERLESS FRAMEWORK 1.0



SERVERLESS IN ACTION

```
[sc5serverless]:aws$ node -v
v4.3.2
[sc5serverless]:aws$ npm install -g serverless > /dev/null
[sc5serverless]:aws$ sls -v
1.0.3
[sc5serverless]:aws$ sls create -t aws-nodejs -p slsIntro -n slsIntro
Serverless: Creating new Serverless service...
Serverless: Creating the service in "/Users/mikael/aws/slsIntro"

-----
|_ _ .-.-.-.-.-.-.-.-.-.-._ _ .-.-.-.-.-.-.-.-.-.-_|_-.-.-.-.-.-.-.-.-.-.-| | | | | | | | | | |
| |__| -__| __| | | -__| __| | -__|__ --|__ --|
|_____|_____|| \___/|_____|__| |__|_____|_____|_____|
|   |               The Serverless Application Framework
|                   serverless.com, v1.0.3
|_____|'
-----'

Serverless: Successfully created service with template: "aws-nodejs"
[sc5serverless]:aws$ cd slsIntro/
[sc5serverless]:slsIntro$ ls
event.json          handler.js           serverless.yml
```

```
1 'use strict';
2
3 module.exports.hello = (event, context, callback) => {
4   const response = {
5     statusCode: 200,
6     body: JSON.stringify({
7       message: 'Go Serverless v1.0! Your function executed successfully!',
8       input: event,
9     }),
10  };
11
12  callback(null, response);
13
14  // Use this code if you don't use the http event with the LAMBDA-PROXY integration
15  // callback(null, { message: 'Go Serverless v1.0! Your function executed successfully!', event });
16 };
17
```



```
14 service: slsIntro
15
16 provider:
17   name: aws
18   runtime: nodejs4.3
19
20 # you can overwrite defaults here
21 # stage: dev
22 # region: us-east-1
23
24 # you can add statements to the Lambda function's IAM Role here
25 # iamRoleStatements:
26 #   - Effect: "Allow"
27 #     Action:
28 #       - "s3:ListBucket"
29 #       Resource: { "Fn::Join" : [ "", [ "arn:aws:s3:::", { "Ref" : "ServerlessDeploymentBucket" } ] ] }
30 #   - Effect: "Allow"
31 #     Action:
32 #       - "s3:PutObject"
33 #     Resource:
34 #       Fn::Join:
35 #         - ""
36 #         - "arn:aws:s3:::"
37 #         - "Ref" : "ServerlessDeploymentBucket"
38
39 # you can add packaging information here
40 #package:
41 #  exclude:
42 #    - exclude-me.js
43 #  artifact: my-service-code.zip
44
45 functions:
46   hello:
47     handler: handler.hello
48     events:
49       - http:
50         path: hello
51         method: get
52
53 # The following are a few example events you can configure
```

Register http event for Lambda

```
1. bash

[sc5serverless]:slsIntro$ sls deploy
Serverless: Packaging service...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service .zip file to S3...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.....
Serverless: Stack update finished...

Service Information
service: slsIntro
stage: dev
region: us-east-1
api keys:
  None
endpoints:
  GET - https://z163kf7ty7.execute-api.us-east-1.amazonaws.com/dev/hello
functions:
  slsIntro-dev-hello: arn:aws:lambda:us-east-1:548412044841:function:slsIntro-dev-hello

[sc5serverless]:slsIntro$ curl https://z163kf7ty7.execute-api.us-east-1.amazonaws.com/dev/hello?foo=bar
{"message":"Go Serverless v1.0! Your function executed successfully!","input":{"resource":"/hello","path":"/hello","httpMethod":"GET","headers":{"Accept":"*/*","CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-Viewer":"true","CloudFront-Is-Mobile-Viewer":"false","CloudFront-Is-SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-Viewer-Country":"FI","Host":"z163kf7ty7.execute-api.us-east-1.amazonaws.com","User-Agent":"curl/7.49.1","Via":"1.1 940b367f846b05ee5d0f25268ff80731.cloudfront.net (CloudFront)","X-Amz-Cf-Id":"L27xTk_n7LMEzgF2ALg-69SY7Vr5EJEE0Bt0rrB9_VSr61bEj4MF3w==","X-Forwarded-For":"217.30.179.246, 54.240.145.6","X-Forwarded-Port":"443","X-Forwarded-Proto":"https"},"queryStringParameters":{"foo":"bar"},"pathParameters":null,"stageVariables":
```

LIMITATIONS OF "VANILLA" SERVERLESS

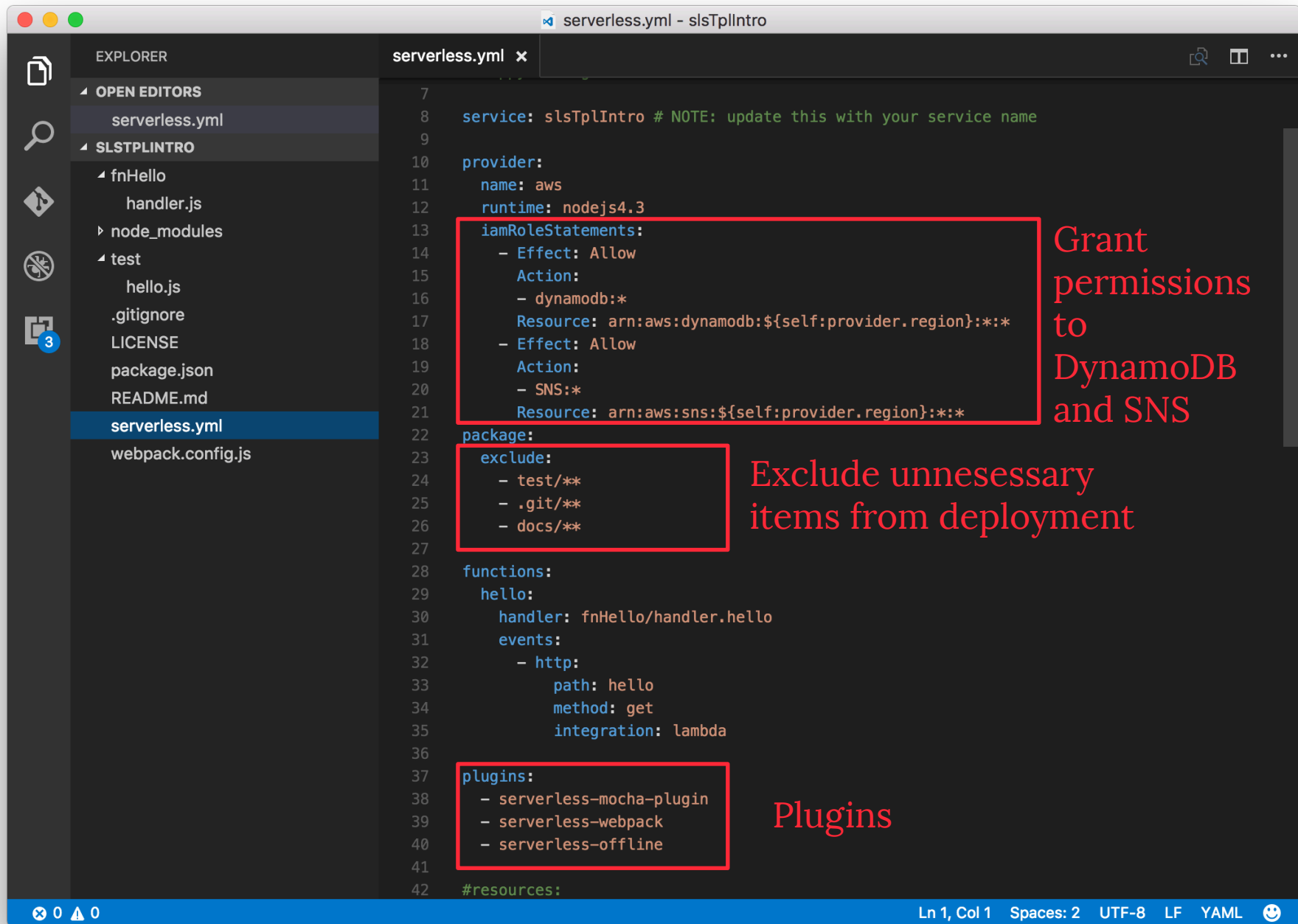
- No ability to run code / test offline
- Deployment produces an unoptimal package => slow "cold start"
- Need to define IAM Roles for AWS Resources (DynamoDB / SNS)

Let's give the SC5 Serverless boilerplate a try!

<https://github.com/SC5/sc5-serverless-boilerplate>

```
1. bash
[sc5serverless]:aws$ sls install -u https://github.com/SC5/sc5-serverless-boilerplate
Serverless: Downloading and installing "sc5-serverless-boilerplate"...
Serverless: Successfully installed "sc5-serverless-boilerplate".
[sc5serverless]:aws$ mv sc5-serverless-boilerplate/ slsTplIntro
[sc5serverless]:aws$ cd slsTplIntro/
[sc5serverless]:slsTplIntro$ perl -pi -e "s/sc5-serverless-boilerplate/slsTplIntro/" serverless.yml
package.json
[sc5serverless]:slsTplIntro$ ls
LICENSE          fnHello          serverless.yml    webpack.config.js
README.md        package.json      test
[sc5serverless]:slsTplIntro$ npm install
```

(Hope to see / will submit a PR for a `-n` option to `sls install` that handles the renaming / substitution part)



The screenshot shows a VS Code editor window titled 'hello.js - slsTplIntro'. The Explorer sidebar on the left shows a project structure with folders '.serverless', 'fnHello', 'node_modules', and 'test'. The 'test' folder is expanded, showing 'hello.js' as the selected file. The main editor area displays the content of 'hello.js', which is a Mocha test file generated by the 'serverless-mocha-plugin'. The code includes imports for 'require', 'serverless-mocha-plugin', 'chai', and 'mochaPlugin'. It defines a 'liveFunction' object with 'region' and 'lambdaFunction' properties. The test suite 'hello' is described with a 'before' hook that initializes the 'liveFunction'. A single test 'implement tests here' is defined, which calls 'wrapped.run' with a 'foo' property set to 'bar' and expects the response to match 'bar' and 'successful'.

```
1 'use strict';
2 // tests for hello
3 // Generated by serverless-mocha-plugin
4
5 const mod = require('../fnHello/handler.js');
6 const mochaPlugin = require('serverless-mocha-plugin');
7 const lambdaWrapper = mochaPlugin.lambdaWrapper;
8 const expect = mochaPlugin.chai.expect;
9
10 const liveFunction = {
11   region: process.env.SERVERLESS_REGION,
12   lambdaFunction: process.env.SERVERLESS_PROJECT + '-hello'
13 };
14
15 const wrapped = lambdaWrapper.wrap(mod, { handler: 'hello' });
16
17 describe('hello', () => {
18   before(function (done) {
19     // lambdaWrapper.init(liveFunction); // Run the deployed lambda
20
21     done();
22   });
23
24   it('implement tests here', (done) => {
25     wrapped.run({
26       'foo': 'bar'
27     }, (err, response) => {
28       expect(response.event.foo).to.equal('bar');
29       expect(response.message).to.match(/successful/);
30       done();
31     });
32   });
33 });
34
```

```
1. node
[sc5serverless]:slsTplIntro$ sls invoke test

hello
Got request: { foo: 'bar' }
  ✓ implement tests here

1 passing (19ms)

[sc5serverless]:slsTplIntro$ sls offline start
Serverless: Starting Offline: dev/us-east-1.

Serverless: Routes for hello:
Serverless: GET /hello

Serverless: Offline listening on http://localhost:3000
```

Command provided by
serverless-mocha-plugin

Command provided by
serverless-offline

```
1. bash
node  %1  bash  %2
[sc5sandbox]:slsTplIntro$ curl 'http://localhost:3000/hello?a=1&b=2'
{"message":"Go Serverless v1.0! Your function executed successfully!","event":{"body":{},"method":"GET","principalId":"offlineContext_authorizer_principalId","headers":{"Host":"localhost:3000","User-Agent":"curl/7.49.1","Accept":"*//*"},"query":{"a":"1","b":"2"},"path":{},"identity":{"accountId":"offlineContext_accountId","apiKey":"offlineContext_apiKey","caller":"offlineContext_caller","cognitoAuthenticationProvider":"offlineContext_cognitoAuthenticationProvider","cognitoAuthenticationType":"offlineContext_cognitoAuthenticationType","sourceIp":"127.0.0.1","user":"offlineContext_user","userAgent":"curl/7.49.1","userArn":"offlineContext_userArn"},"stageVariables":{},"isOffline":true}}[sc5sandbox]:slsTplIntro$
```

```
1. bash
[sc5serverless]:slsTplIntro$ sls deploy
Serverless: Creating Stack...
Serverless: Checking Stack create progress...
.....
Serverless: Stack create finished...
Serverless: Bundling with Webpack...
Time: 49ms
      Asset      Size  Chunks             Chunk Names
fnHello/handler.js 1.66 kB      0  [emitted]  fnHello/handler
Serverless: Packaging service...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading service .zip file to S3...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.....
Serverless: Stack update finished...

Service Information
service: slsTplIntro
stage: dev
region: us-east-1
api keys:
  None
endpoints:
  GET - https://5x1hhllwwd.execute-api.us-east-1.amazonaws.com/dev/hello
functions:
  slsTplIntro-dev-hello: arn:aws:lambda:us-east-1:548412044841:function:slsTplIntro-dev-hello

[sc5serverless]:slsTplIntro$
```

Use -v for more verbose output


```
1. bash
[sc5serverless]:slsTplIntro$ curl https://5x1hhllwwd.execute-api.us-east-1.amazonaws.com/dev/hello?key=value1
{"message":"Go Serverless v1.0! Your function executed successfully!","event":{"body":{},"method":"GET","principalId":"","stage":"dev","headers":{"Accept":"*/*","CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-Viewer":"true","CloudFront-Is-Mobile-Viewer":"false","CloudFront-Is-SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-Viewer-Country":"FI","Host":"5x1hhllwwd.execute-api.us-east-1.amazonaws.com","User-Agent":"curl/7.49.1","Via":"1.1 951bc6ecd5fb2c9732f14df07a0958a9.cloudfront.net (CloudFront)","X-Amz-Cf-Id":"02JL_c1AJ60rxnkdazHjLPKDpqEvbxNw-yUg3mmrPeRrMsyX7sTTRw==","X-Forwarded-For":"85.76.117.31, 205.251.218.9","X-Forwarded-Port":"443","X-Forwarded-Proto":"https"},"query":{"key":"value1"},"path":{},"identity":{"cognitoIdentityPoolId":"","accountId":"","cognitoIdentityId":"","caller":"","apiKey":"","sourceIp":"85.76.117.31","accessKey":"","cognitoAuthenticationType":"","cognitoAuthenticationProvider":"","userArn":"","userAgent":"curl/7.49.1","user":""},"stageVariables":{}}}[sc5serverless]:slsTplIntro$
[sc5serverless]:slsTplIntro$ sls logs -f hello
START RequestId: 2bea4e42-a067-11e6-9cf3-45f477716116 Version: $LATEST
2016-11-01 21:12:46.467 (+02:00)      2bea4e42-a067-11e6-9cf3-45f477716116      Got request: { body:
  {},
  method: 'GET',
  principalId: '',
  stage: 'dev',
  headers:
    { Accept: '*/*',
      'CloudFront-Forwarded-Proto': 'https',
      'CloudFront-Is-Desktop-Viewer': 'true',
      'CloudFront-Is-Mobile-Viewer': 'false',
      'CloudFront-Is-SmartTV-Viewer': 'false',
      'CloudFront-Is-Tablet-Viewer': 'false',
      'CloudFront-Viewer-Country': 'FI',
      Host: '5x1hhllwwd.execute-api.us-east-1.amazonaws.com',
```

Use -t for continuous log

```
1. bash
[sc5serverless]:slsTplIntro$ sls info

Service Information
service: slsTplIntro
stage: dev
region: us-east-1
api keys:
  None
endpoints:
  GET - https://5x1hh1lwwd.execute-api.us-east-1.amazonaws.com/dev/hello
functions:
  slsTplIntro-dev-hello: arn:aws:lambda:us-east-1:548412044841:function:slsTplIntro-dev-hello

[sc5serverless]:slsTplIntro$
```

ADDITIONAL RESOURCES

Boilerplates

- <https://github.com/SC5/serverless-messenger-boilerplate> : Create bots using Messenger Platform and Wit.ai
- <https://github.com/SC5/serverless-authentication-boilerplate> : Create API Gateway custom authorizers

Workshops

- <http://serverless.fi/docs/blog-workshop.pdf> : Build a simple serverless chat service
- <http://serverless.fi/docs/messenger-bot-workshop.pdf> : Build a Messenger weather bot using Wit.ai and Serverless



THANK YOU!

mikael.puittinen@sc5.io