SERVERLESS SERVICES ON AWS

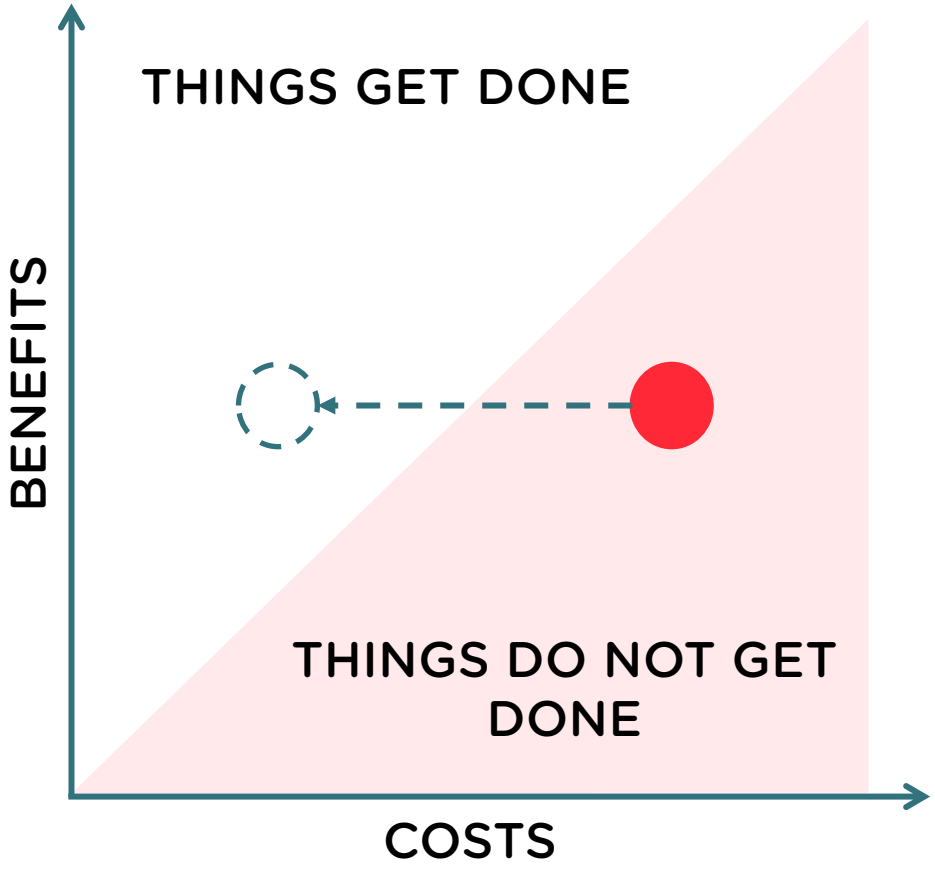
Mikael Puittinen, Chief Technology Officer

2.11.2016

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



SC5

SC5 BRIEFLY

CLOUD SOLUTIONS

BUSINESS APPLICATIONS

DIGITAL DESIGN

10 YEARS	**60+** CUSTOMERS	**200+** PROJECTS
85 HACKERS DESIGNERS	**HEL JKL**	**~7** MEUR 2016

A-lehdet, DNA, elisa, Enegia, Gasum, HAPPY OR NOT, KEMPPI, VEIKKAUS, POP Vakuutus, sanoma, TeliaSonera, WÄRTSILÄ, L&T, amazon web services Partner Network CONSULTING PARTNER

VISIT OUR WEB SITE FOR MORE INFO: HTTPS://SC5.IO

SC5

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

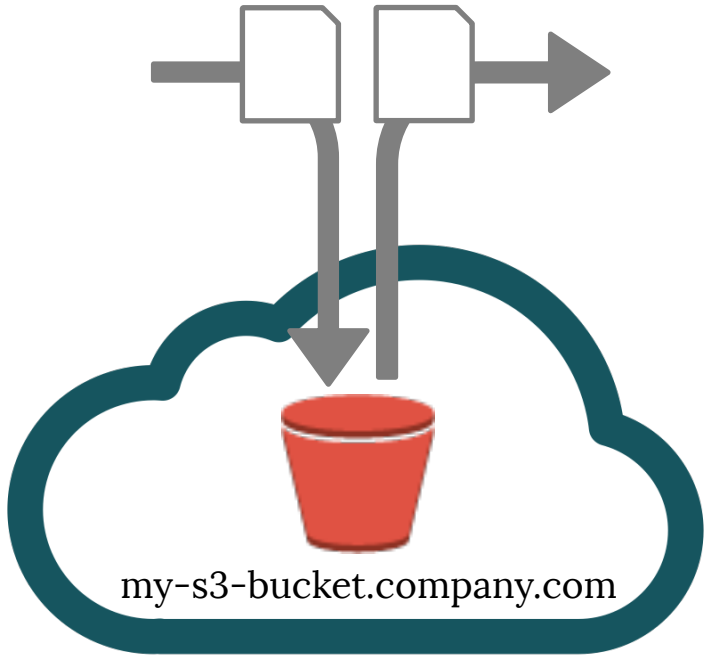
ANALYTICS

 Machine Learning 2015

SC5

SIMPLE STORAGE SERVICE (S3)

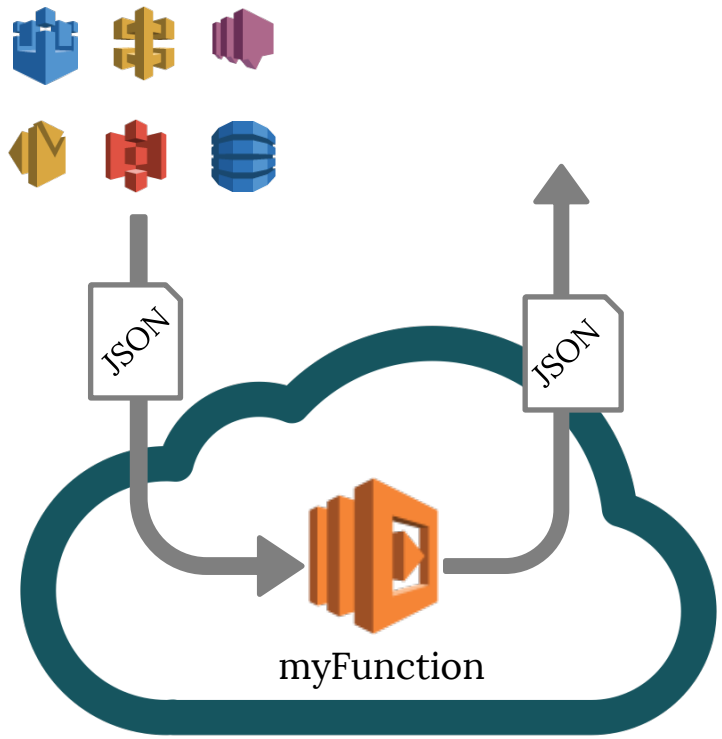
- (Unlimited) file storage service
- Application internal files (user file uploads / downloads)
- Static web content (e.g. application HTML / CSS / JS / image assets)
- Can be complemented with CloudFront CDN to optimize costs and performance



PRICING: Storage volume + amount of requests

AWS LAMBDA

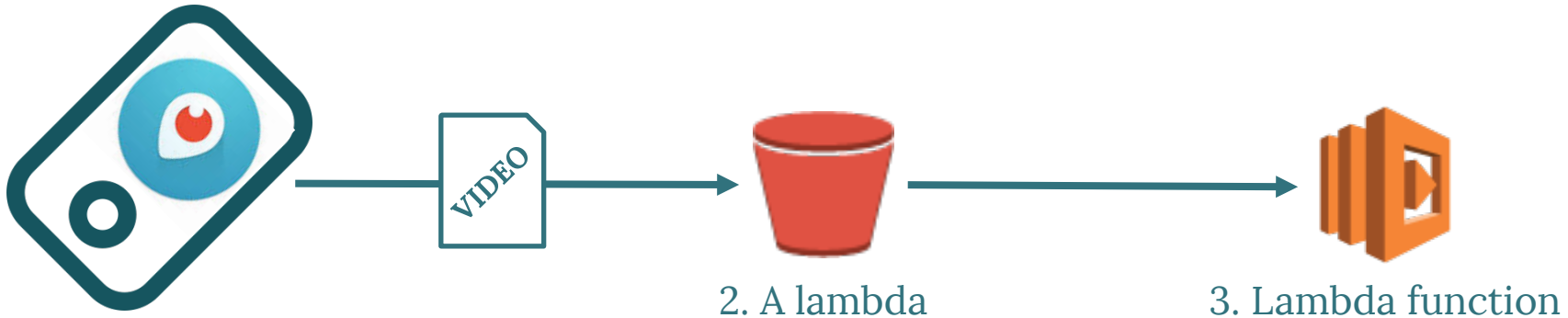
- Compute service for running code (functions) in AWS
- Event driven (API Gateway, SNS, SES, S3, DynamoDB, Schedule, ...)
- Provision memory & max time required by single function run
- Additional ”instances” spawned automatically (cold / hot start)



PRICING: Utilized gigabyteseconds (rounded to 100ms)

Example

PERISCOPE CONTENT FILTER



1. Periscope app uploads 3s video clips to S3 bucket

2. A lambda function is triggered by uploads to S3 bucket

3. Lambda function filters incoming clips for adult / copyrighted content

SC5

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

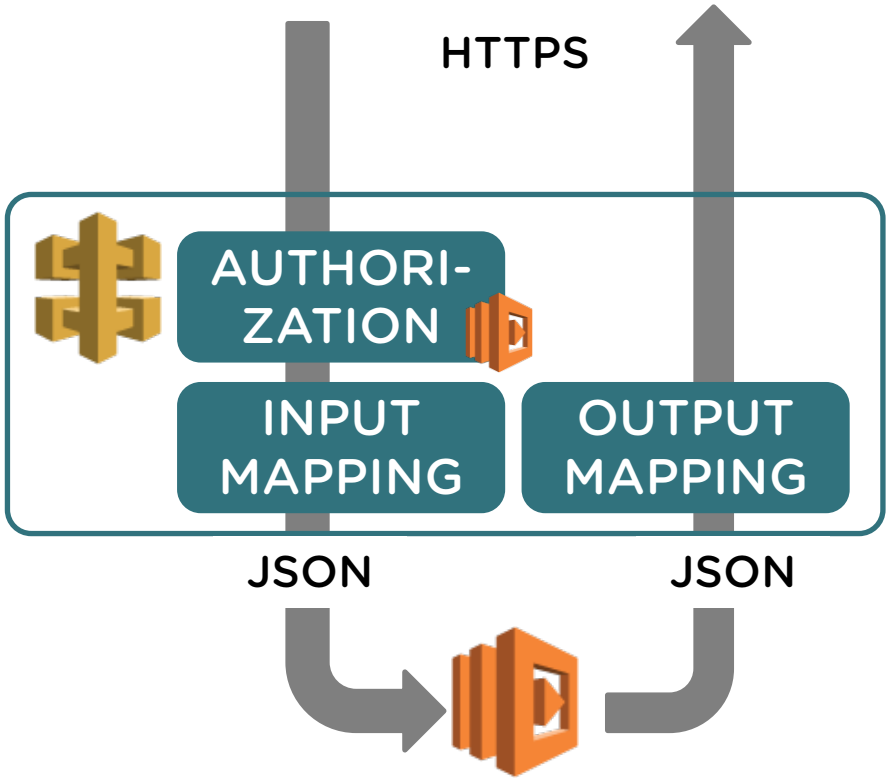
```
'use strict';

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

SC5

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

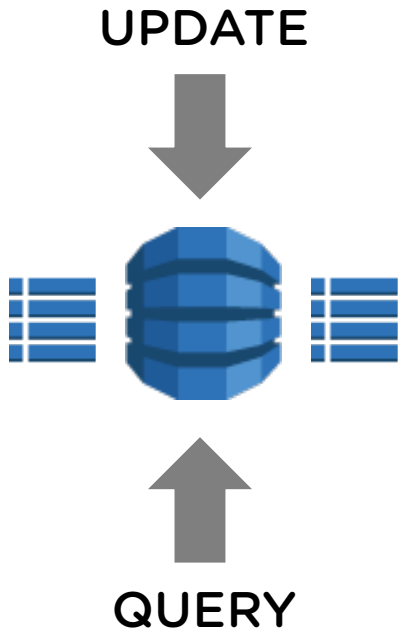
EXERCISE: API GATEWAY

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

Serverless noSQL Database

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

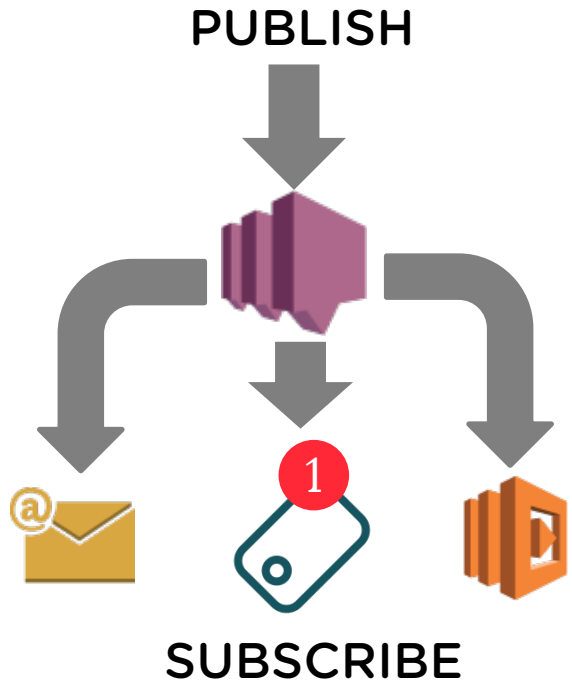


PRICING: Provisioned read / write capacity and storage (over 25Gb)

SC5

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

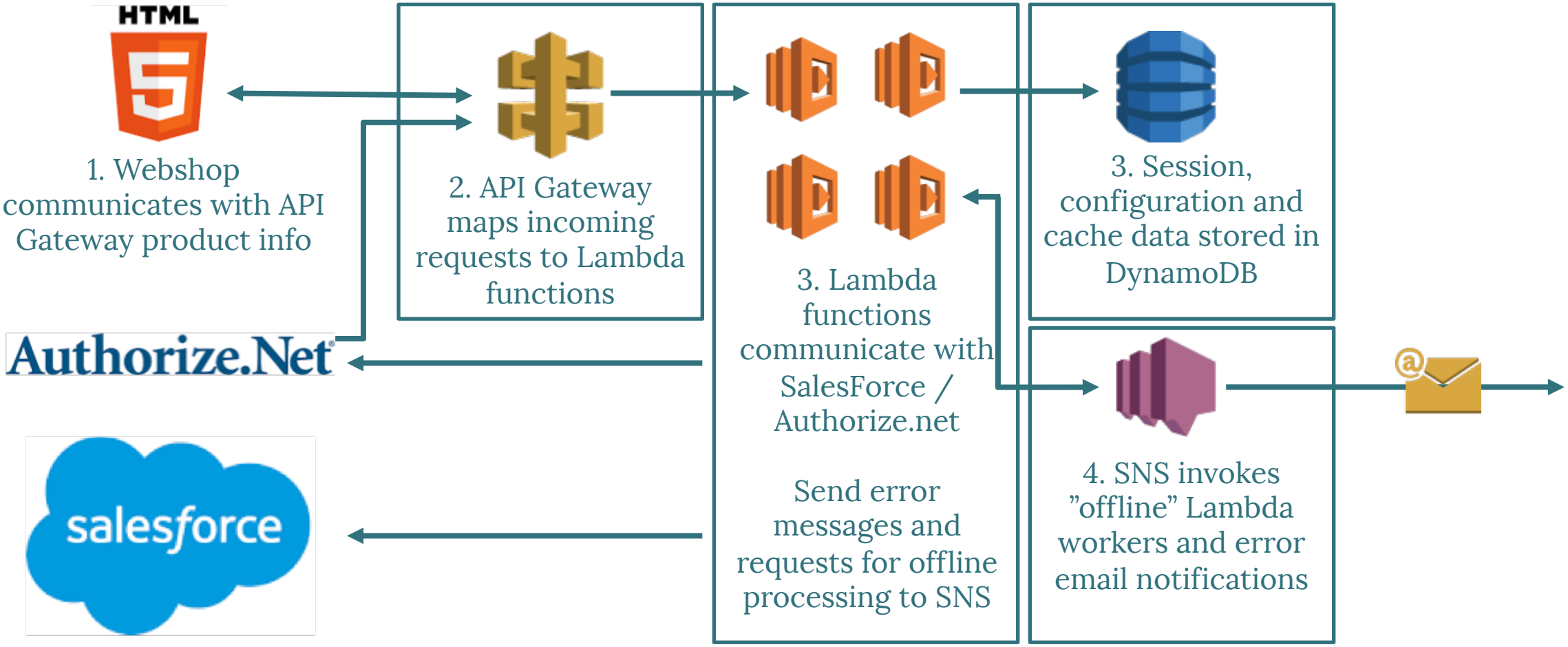
EXERCISE: SNS

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

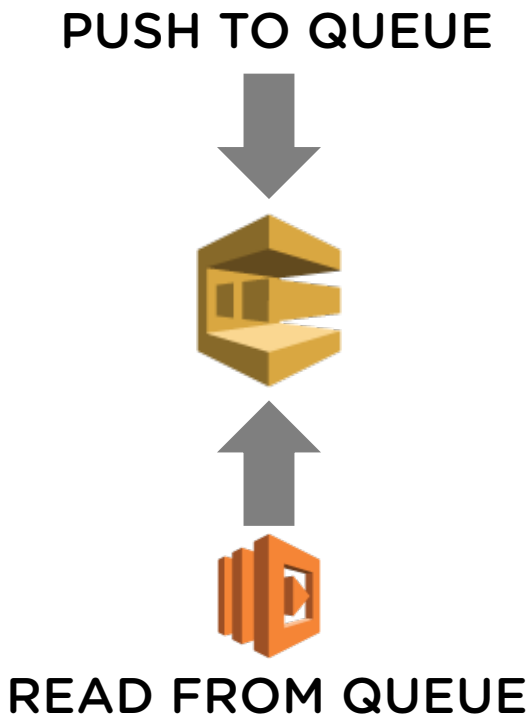


SC5

Serverless message queue

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)



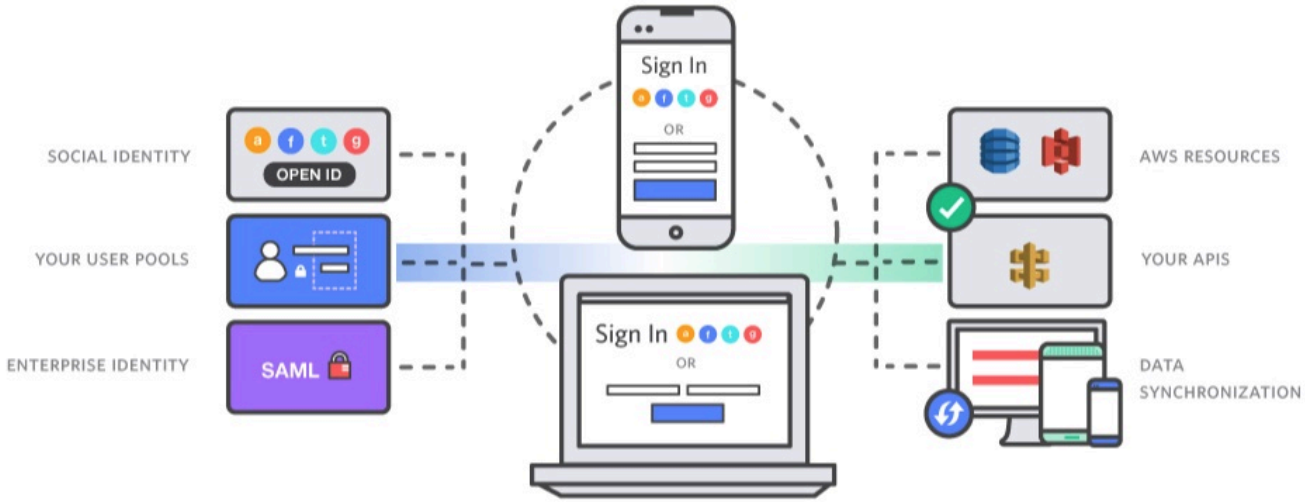
PRICING: Amount of messages

SC5

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
- ...



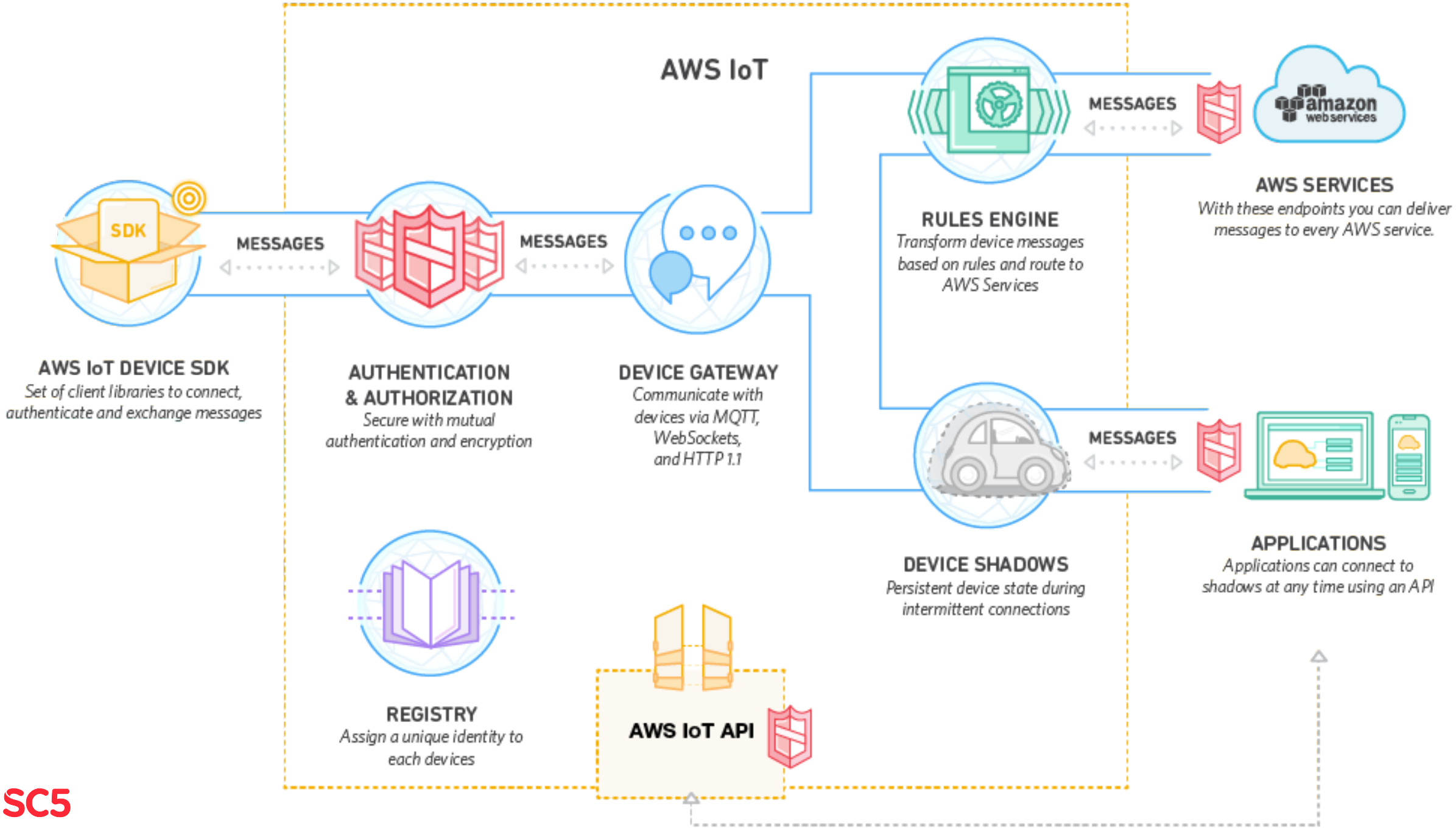
PRICING: # of Monthly Active Users

SC5

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)

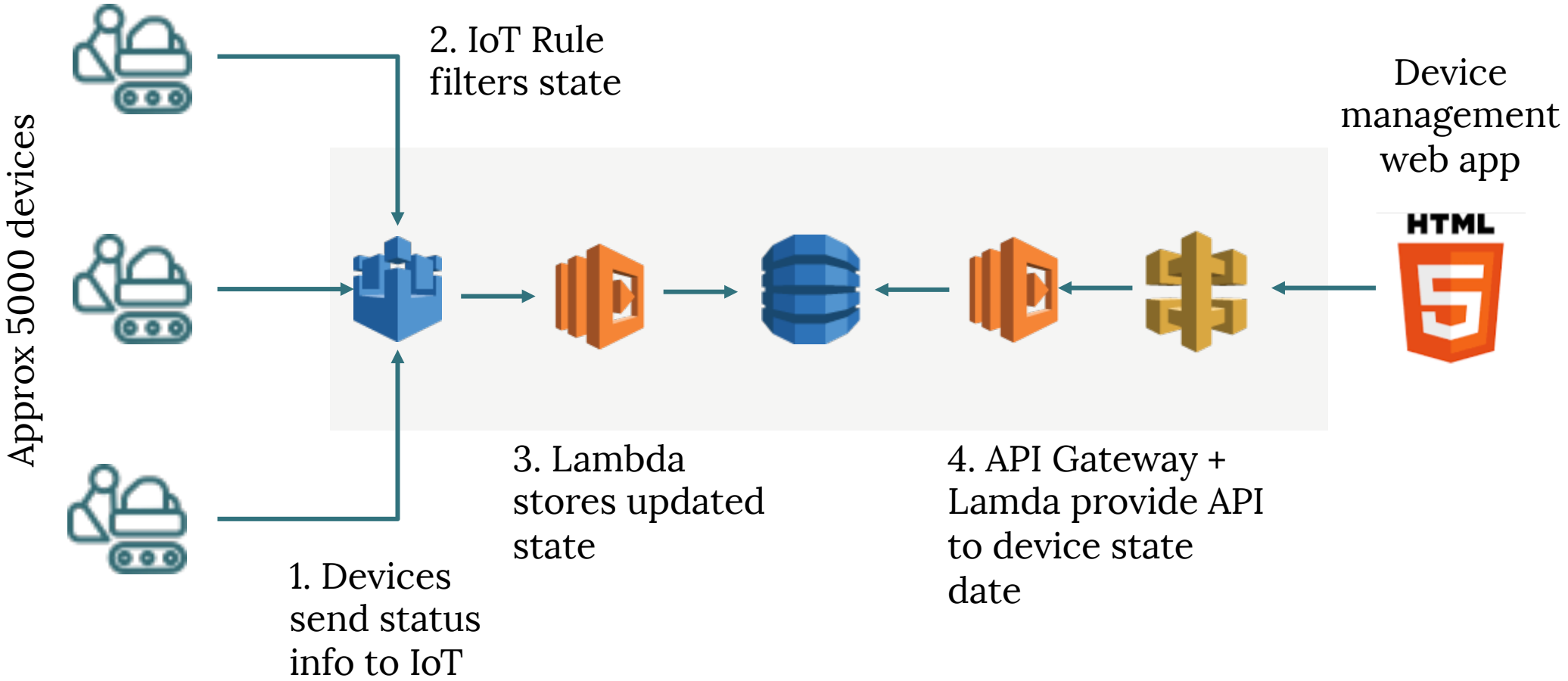
PRICING: # of messages


AWS IoT
SDK
MESSAGES
MESSAGES
MESSAGES
MESSAGES
amazon web services
AWS IoT DEVICE SDK
Set of client libraries to connect, authenticate and exchange messages
AUTHENTICATION & AUTHORIZATION
Secure with mutual authentication and encryption
DEVICE GATEWAY
Communicate with devices via MQTT, WebSockets, and HTTP 1.1
RULES ENGINE
Transform device messages based on rules and route to AWS Services
AWS SERVICES
With these endpoints you can deliver messages to every AWS service.
DEVICE SHADOWS
Persistent device state during intermittent connections
APPLICATIONS
Applications can connect to shadows at any time using an API
REGISTRY
Assign a unique identity to each devices
AWS IoT API

SC5

Example: IoT

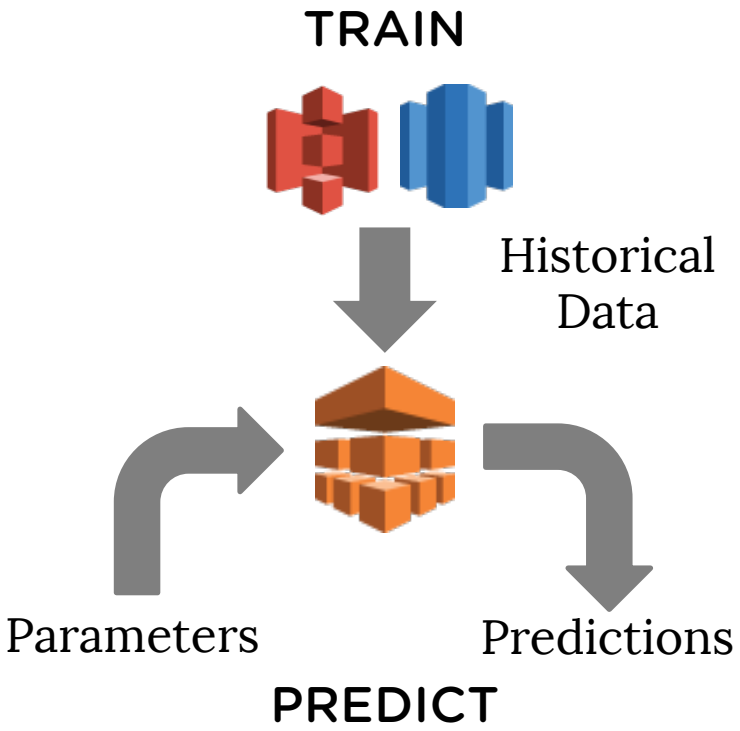
DEVICE MANAGEMENT CASE



SC5

MACHINE LEARNING

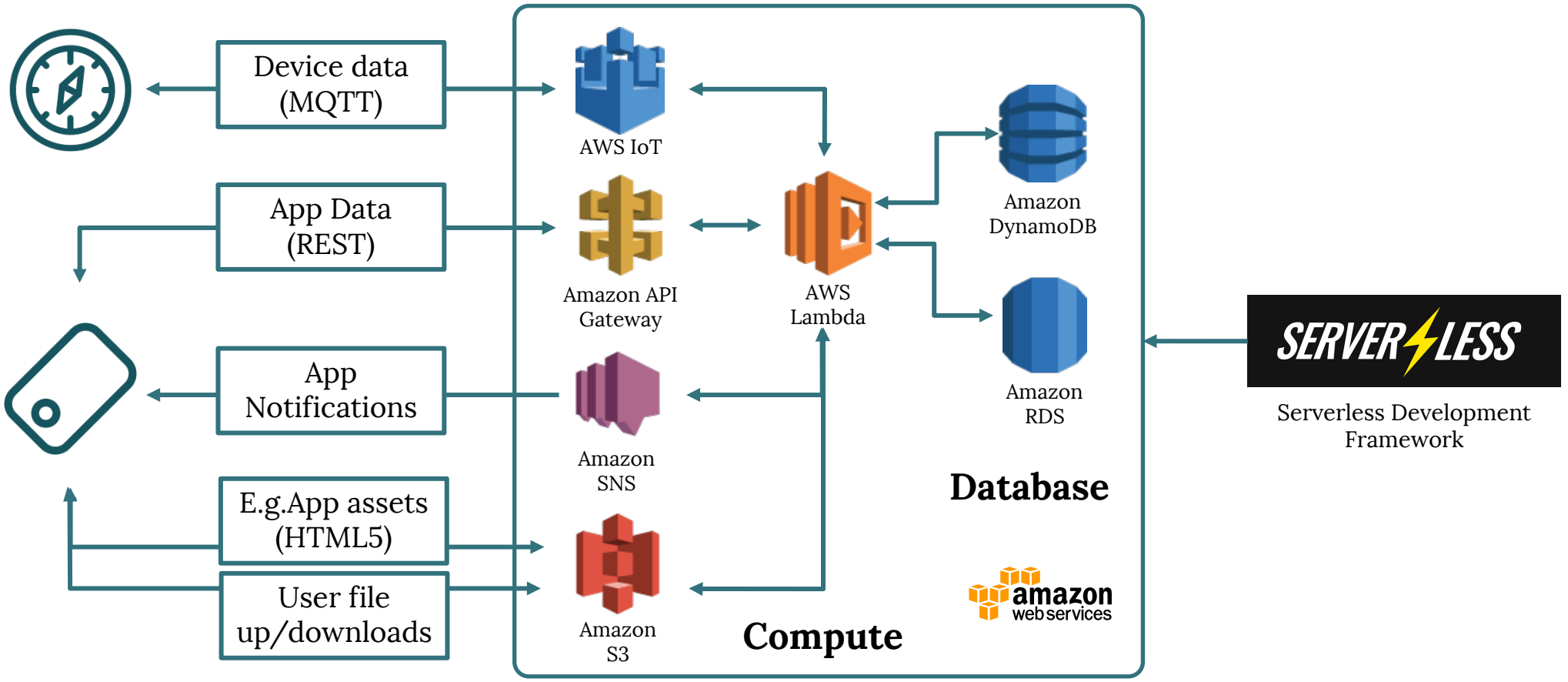
- Machine learning as a service
- Create & review ML models
- Solve binary classification, multi-class classification or regression prediction problems
- Batch & real time predictions



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

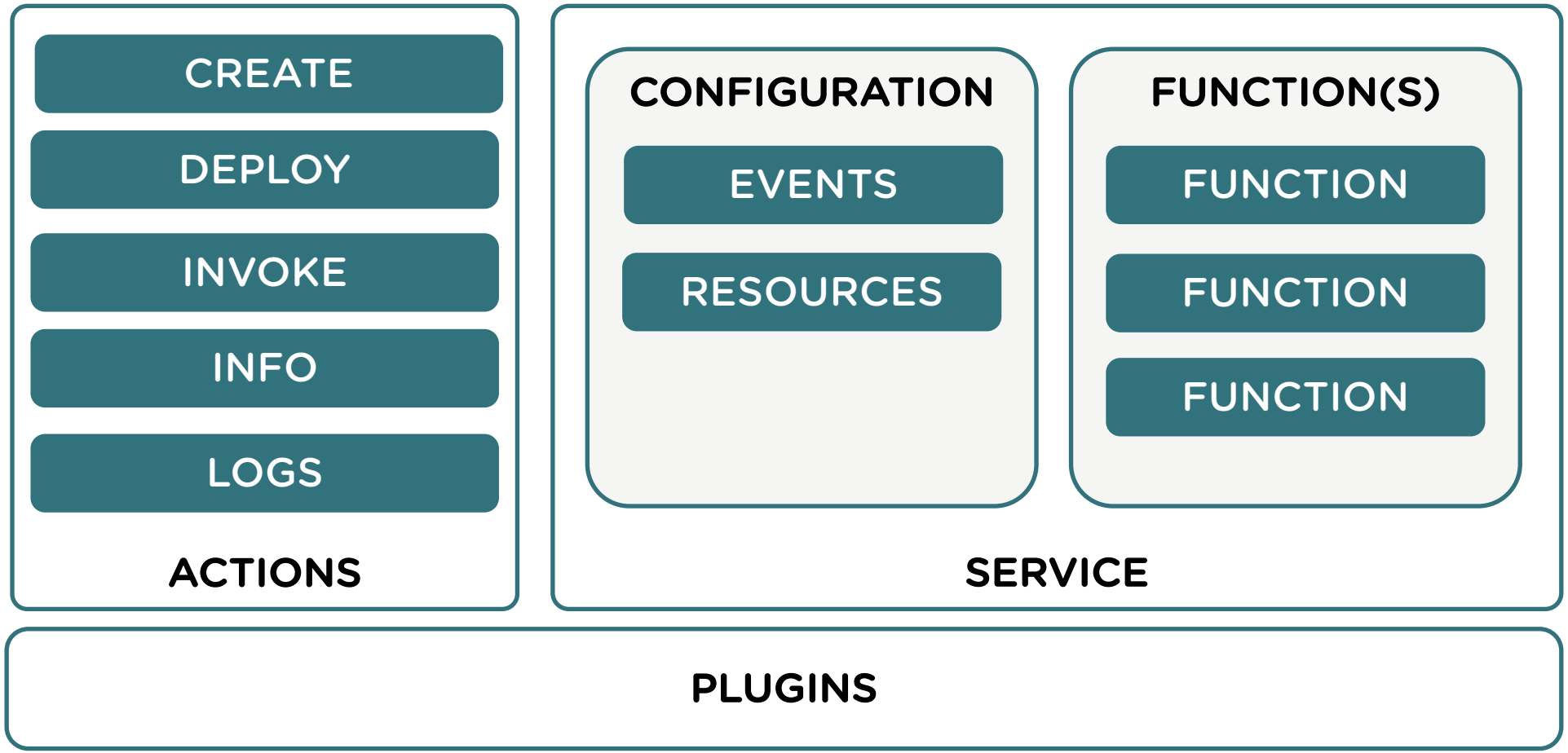
SC5

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


SERVERLESS

SERVERLESS FRAMEWORK 1.0

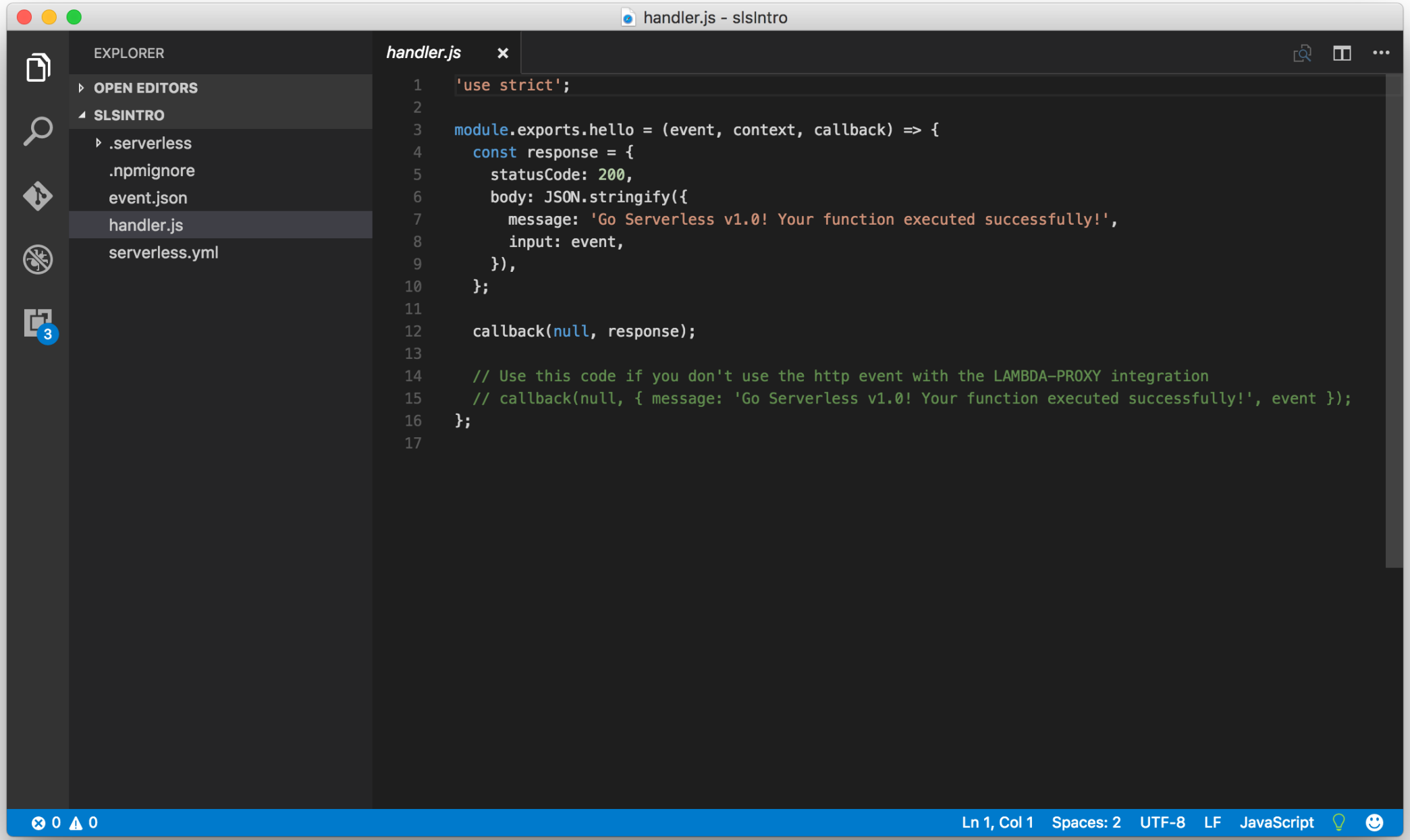


SERVERLESS IN ACTION

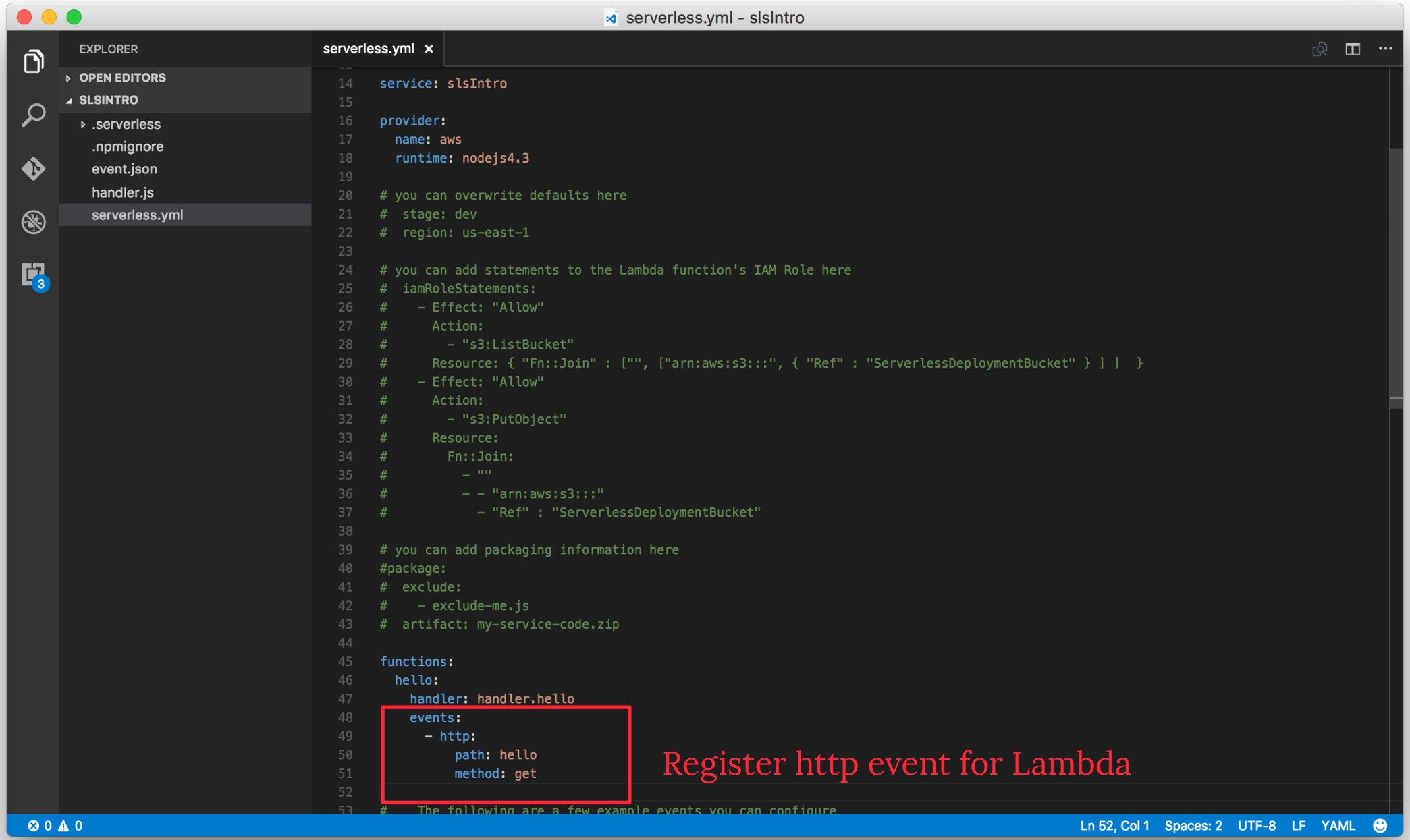
```
[sc5serverless]:aws$ node -v
v4.3.2
[sc5serverless]:aws$ npm install -g serverless > /dev/null
[sc5serverless]:aws$ sls -v
1.0.3
[sc5serverless]:aws$ sls create -t aws-nodejs -p slsIntro -n slsIntro
Serverless: Creating new Serverless service…
Serverless: Creating the service in "/Users/mikael/aws/slsIntro"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

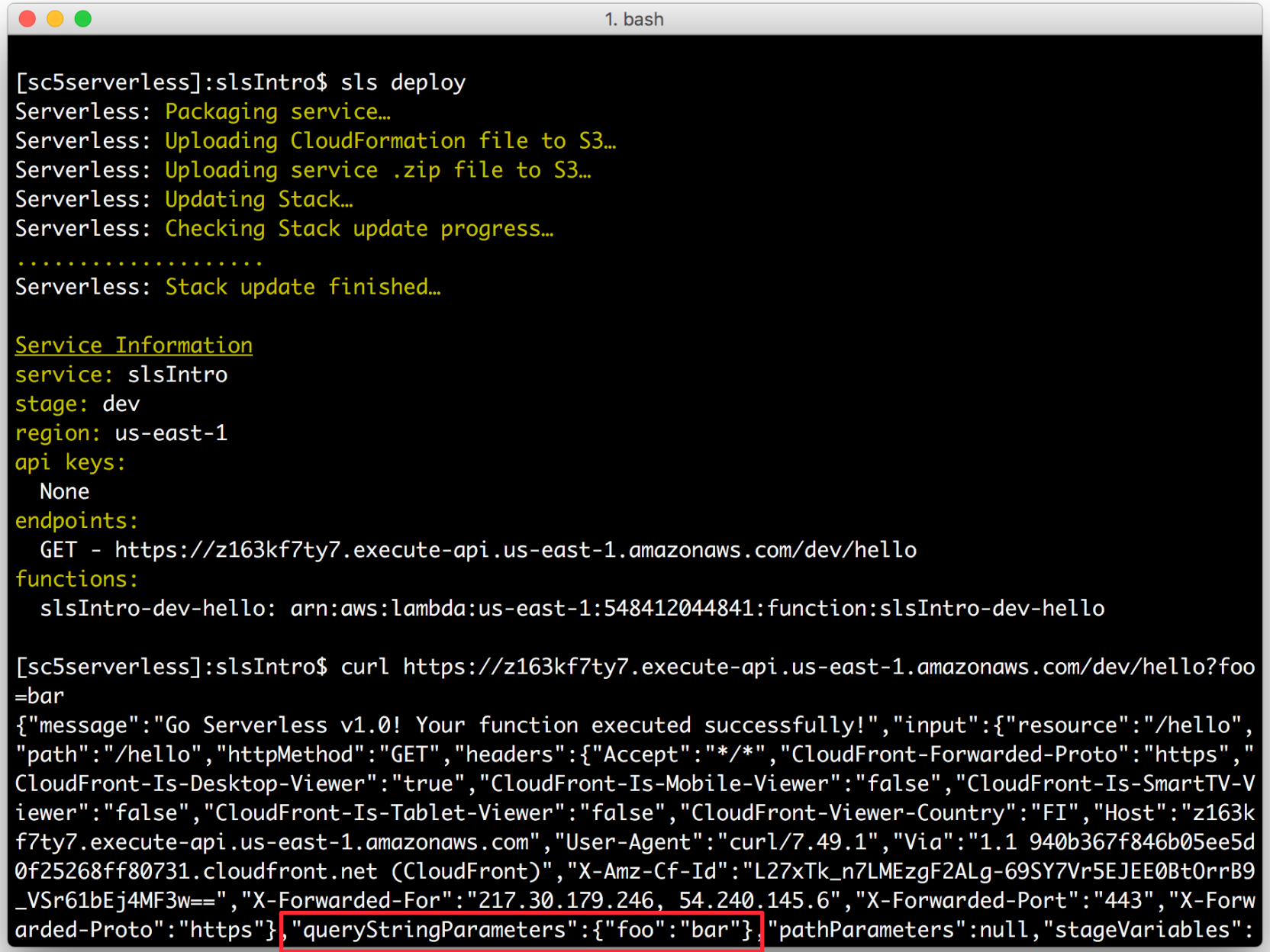
SC5


handler.js - slsIntro
EXPLORER
OPEN EDITORS
SLSINTRO
.serverless
.npmignore
event.json
handler.js
serverless.yml
handler.js
'use strict';
module.exports.hello = (event, context, callback) => {
const response = {
statusCode: 200,
body: JSON.stringify({
message: 'Go Serverless v1.0! Your function executed successfully!',
input: event,
}),
};
callback(null, response);
// Use this code if you don't use the http event with the LAMBDA-PROXY integration
// callback(null, { message: 'Go Serverless v1.0! Your function executed successfully!', event });
};
0 0
Ln 1, Col 1
Spaces: 2
UTF-8
LF
JavaScript

SC5


serverless.yml - slsIntro
EXPLORER
OPEN EDITORS
SLSINTRO
.serverless
.npmignore
event.json
handler.js
serverless.yml
serverless.yml
service: slsIntro
provider:
name: aws
runtime: nodejs4.3
you can overwrite defaults here
stage: dev
region: us-east-1
you can add statements to the Lambda function's IAM Role here
iamRoleStatements:
- Effect: "Allow"
Action:
- "s3:ListBucket"
Resource: { "Fn::Join" : ["", ["arn:aws:s3:::", { "Ref" : "ServerlessDeploymentBucket" }]] }
- Effect: "Allow"
Action:
- "s3:PutObject"
Resource:
Fn::Join:
- ""
- - "arn:aws:s3:::"
- "Ref" : "ServerlessDeploymentBucket"
you can add packaging information here
#package:
exclude:
- exclude-me.js
artifact: my-service-code.zip
functions:
hello:
handler: handler.hello
events:
- http:
path: hello
method: get
The following are a few example events you can configure
0 0
Ln 52, Col 1
Spaces: 2
UTF-8
LF
YAML
Register http event for Lambda

SC5


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-V
iewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-Viewer-Country":"FI","Host":"z163k
f7ty7.execute-api.us-east-1.amazonaws.com","User-Agent":"curl/7.49.1","Via":"1.1 940b367f846b05ee5d
0f25268ff80731.cloudfront.net (CloudFront)","X-Amz-Cf-Id":"L27xTk_n7LMEzgF2ALg-69SY7Vr5EJEE0BtOrrB9
_VSr61bEj4MF3w==","X-Forwarded-For":"217.30.179.246, 54.240.145.6","X-Forwarded-Port":"443","X-Forw
arded-Proto":"https"},"queryStringParameters":{"foo":"bar"},"pathParameters":null,"stageVariables":

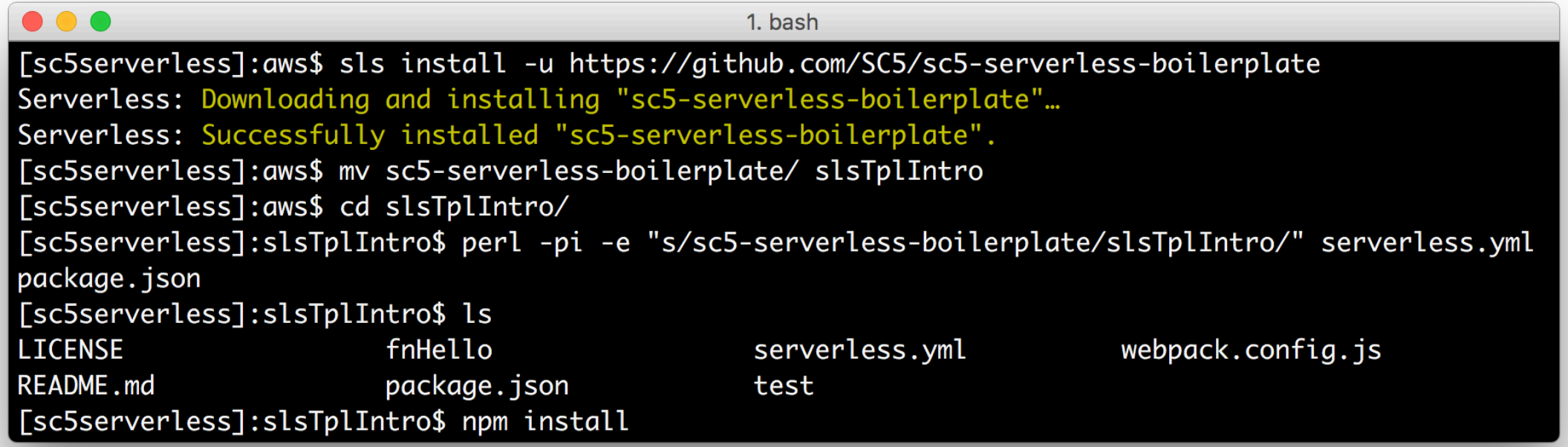
SC5

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)

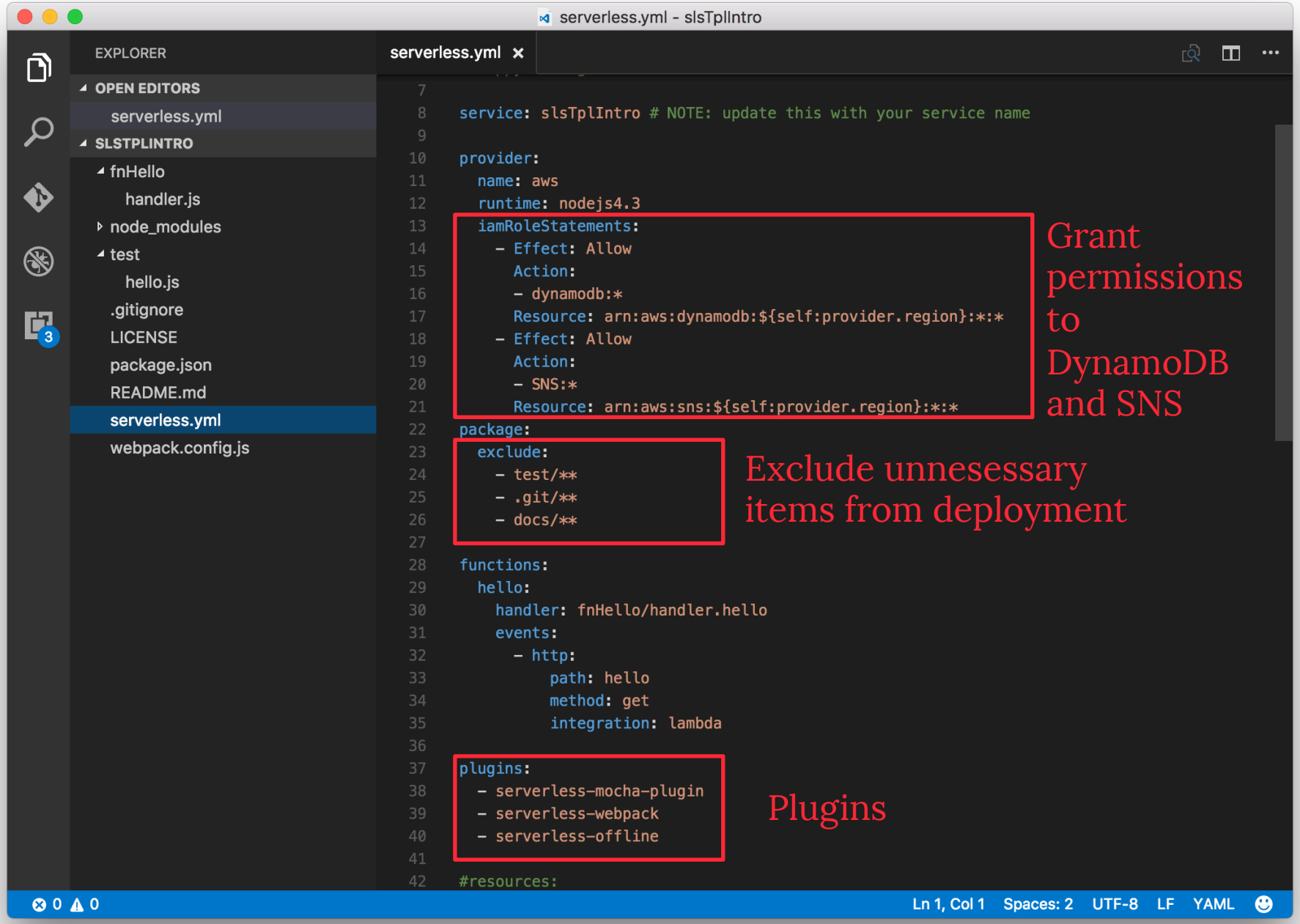
Let’s give the SC5 Serverless boilerplate a try!

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

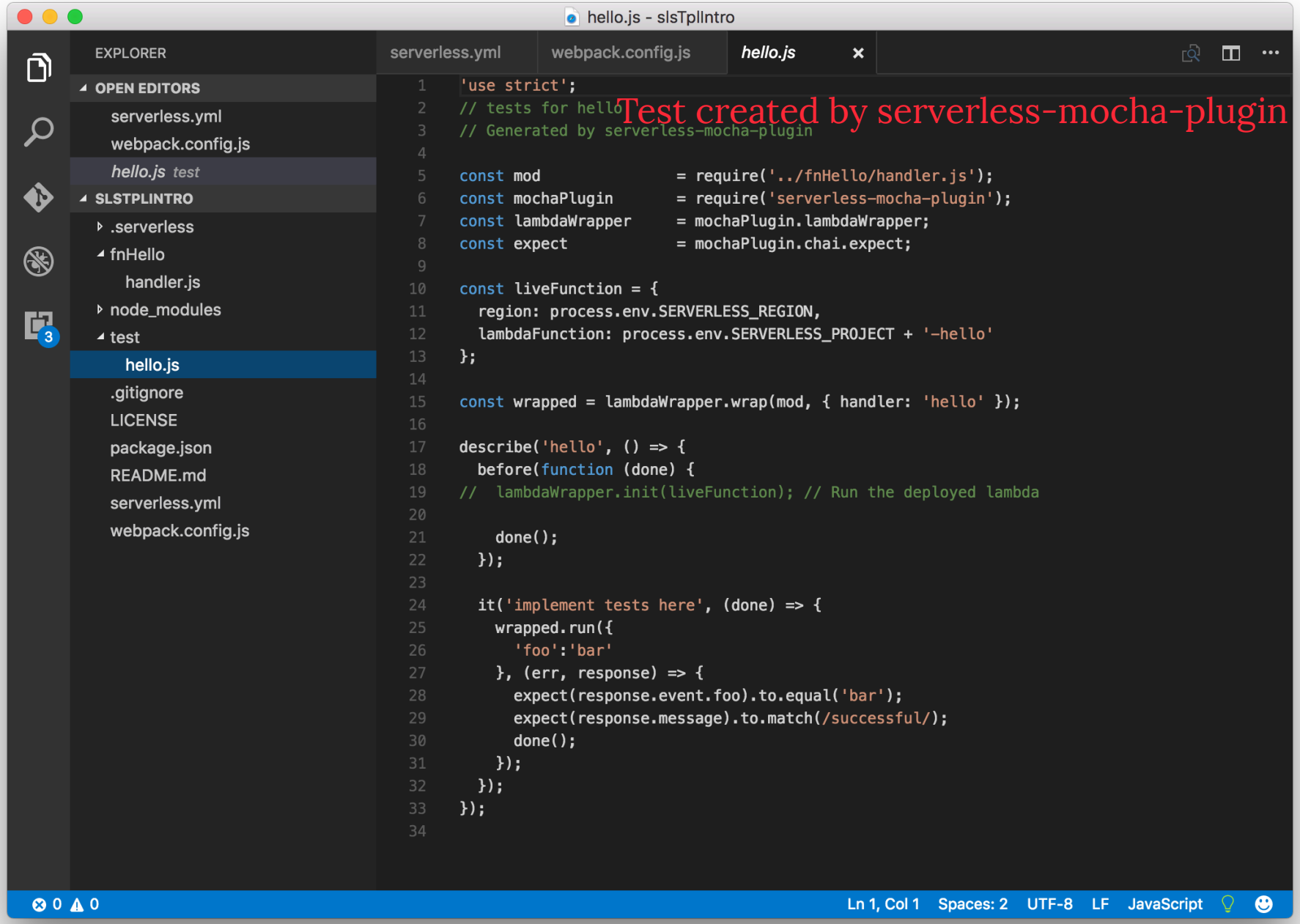


(Hope to see / will submit a PR for a -n option to sls install that handles the renaming / substitution part)

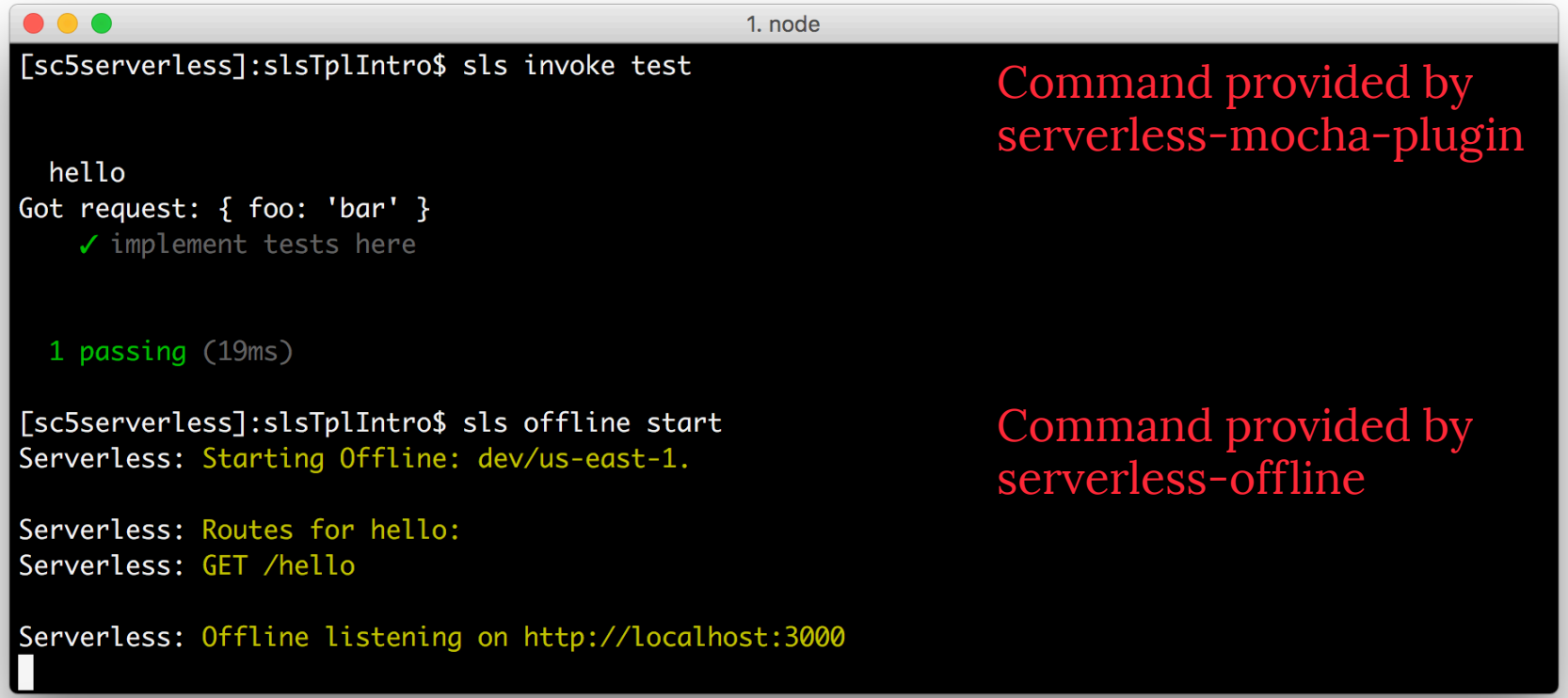
SC5

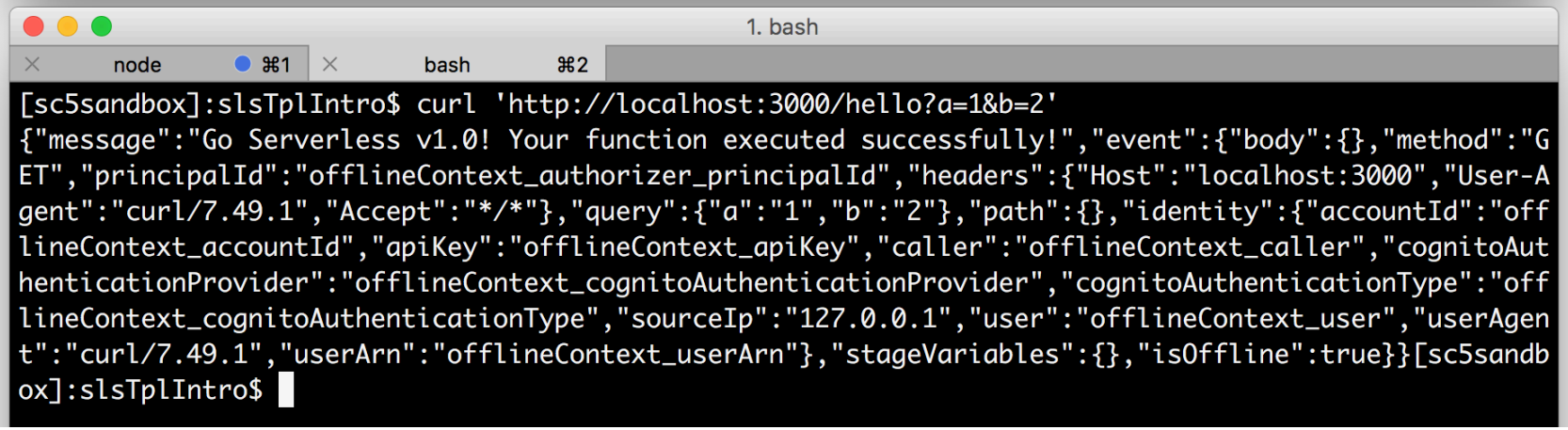

serverless.yml - slsTplIntro
EXPLORER
OPEN EDITORS
serverless.yml
SLSTPLINTRO
fnHello
handler.js
node_modules
test
hello.js
.gitignore
LICENSE
package.json
README.md
serverless.yml
webpack.config.js
serverless.yml
service: slsTplIntro # NOTE: update this with your service name
provider:
name: aws
runtime: nodejs4.3
iamRoleStatements:
- Effect: Allow
Action:
- dynamodb:*
Resource: arn:aws:dynamodb:${self:provider.region}:*:*
- Effect: Allow
Action:
- SNS:*
Resource: arn:aws:sns:${self:provider.region}:*:*
package:
exclude:
- test/**
- .git/**
- docs/**
functions:
hello:
handler: fnHello/handler.hello
events:
- http:
path: hello
method: get
integration: lambda
plugins:
- serverless-mocha-plugin
- serverless-webpack
- serverless-offline
#resources:
Grant permissions to DynamoDB and SNS
Exclude unnesessary items from deployment
Plugins
0 0
Ln 1, Col 1
Spaces: 2
UTF-8
LF
YAML

SC5

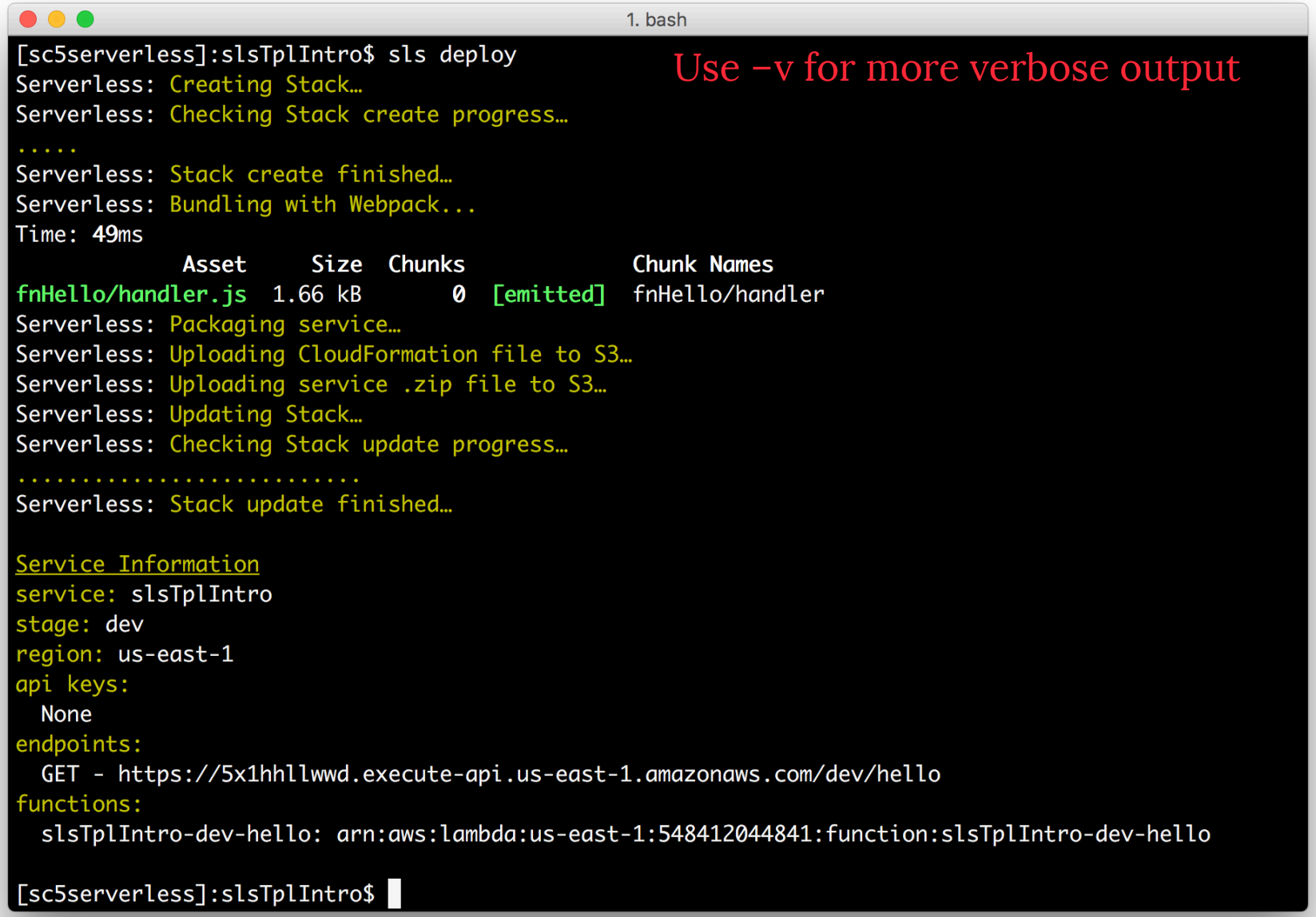

hello.js - slsTplIntro
EXPLORER
OPEN EDITORS
serverless.yml
webpack.config.js
hello.js test
SLSTPLINTRO
.serverless
fnHello
handler.js
node_modules
test
hello.js
.gitignore
LICENSE
package.json
README.md
serverless.yml
webpack.config.js
serverless.yml
webpack.config.js
hello.js
Test created by serverless-mocha-plugin
'use strict';
// tests for hello
// Generated by serverless-mocha-plugin
const mod = require('../fnHello/handler.js');
const mochaPlugin = require('serverless-mocha-plugin');
const lambdaWrapper = mochaPlugin.lambdaWrapper;
const expect = mochaPlugin.chai.expect;
const liveFunction = {
region: process.env.SERVERLESS_REGION,
lambdaFunction: process.env.SERVERLESS_PROJECT + '-hello'
};
const wrapped = lambdaWrapper.wrap(mod, { handler: 'hello' });
describe('hello', () => {
before(function (done) {
// lambdaWrapper.init(liveFunction); // Run the deployed lambda
done();
});
it('implement tests here', (done) => {
wrapped.run({
'foo':'bar'
}, (err, response) => {
expect(response.event.foo).to.equal('bar');
expect(response.message).to.match(/successful/);
done();
});
});
});
0 0
Ln 1, Col 1
Spaces: 2
UTF-8
LF
JavaScript

SC5

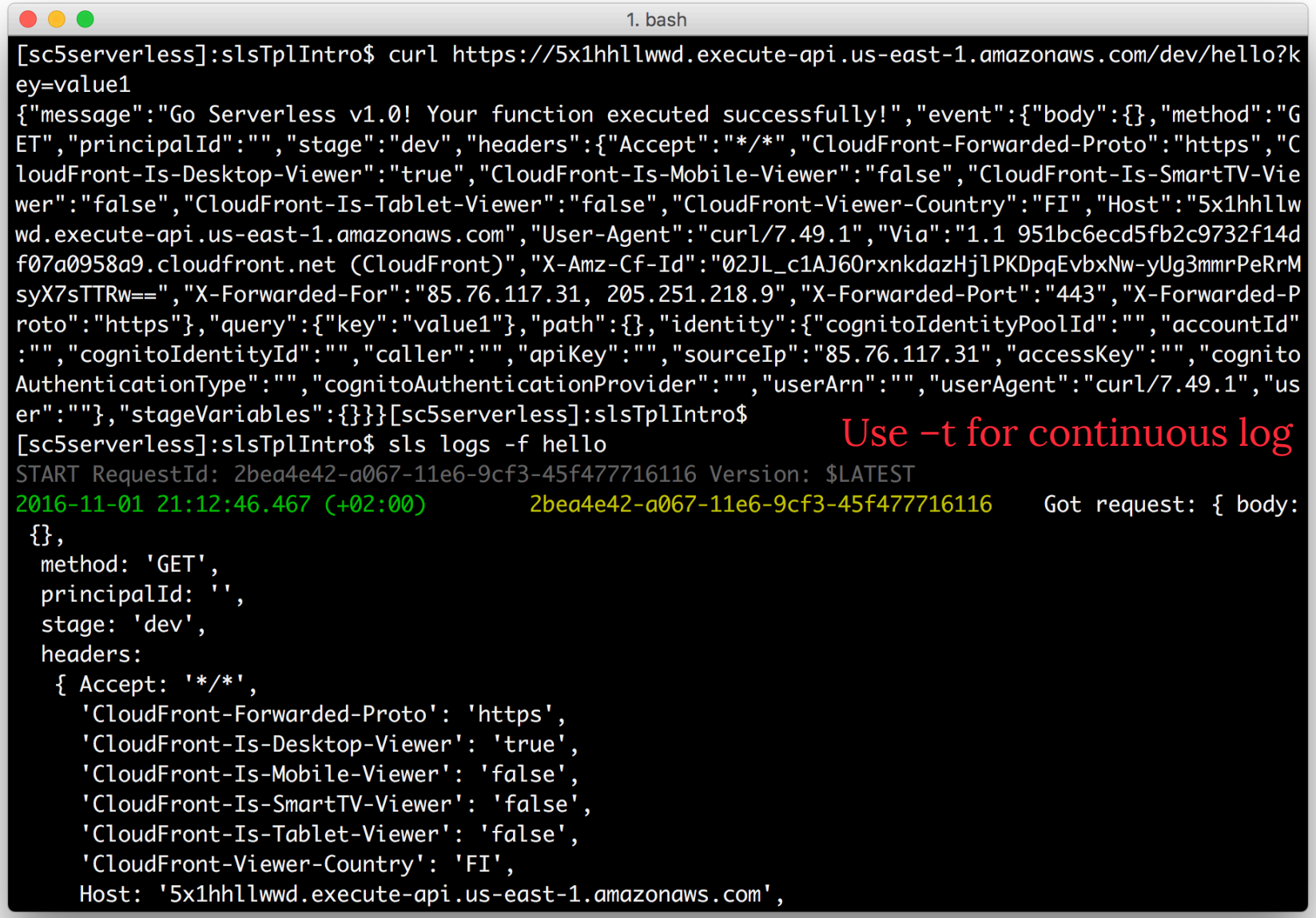

1. node
[sc5serverless]:slsTplIntro$ sls invoke test
hello
Got request: { foo: 'bar' }
✓ implement tests here
1 passing (19ms)
[sc5serverless]:slsTplIntro$ sls offline start
Serverless: Starting Offline: dev/us-east-1.
Serverless: Routes for hello:
Serverless: GET /hello
Serverless: Offline listening 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$

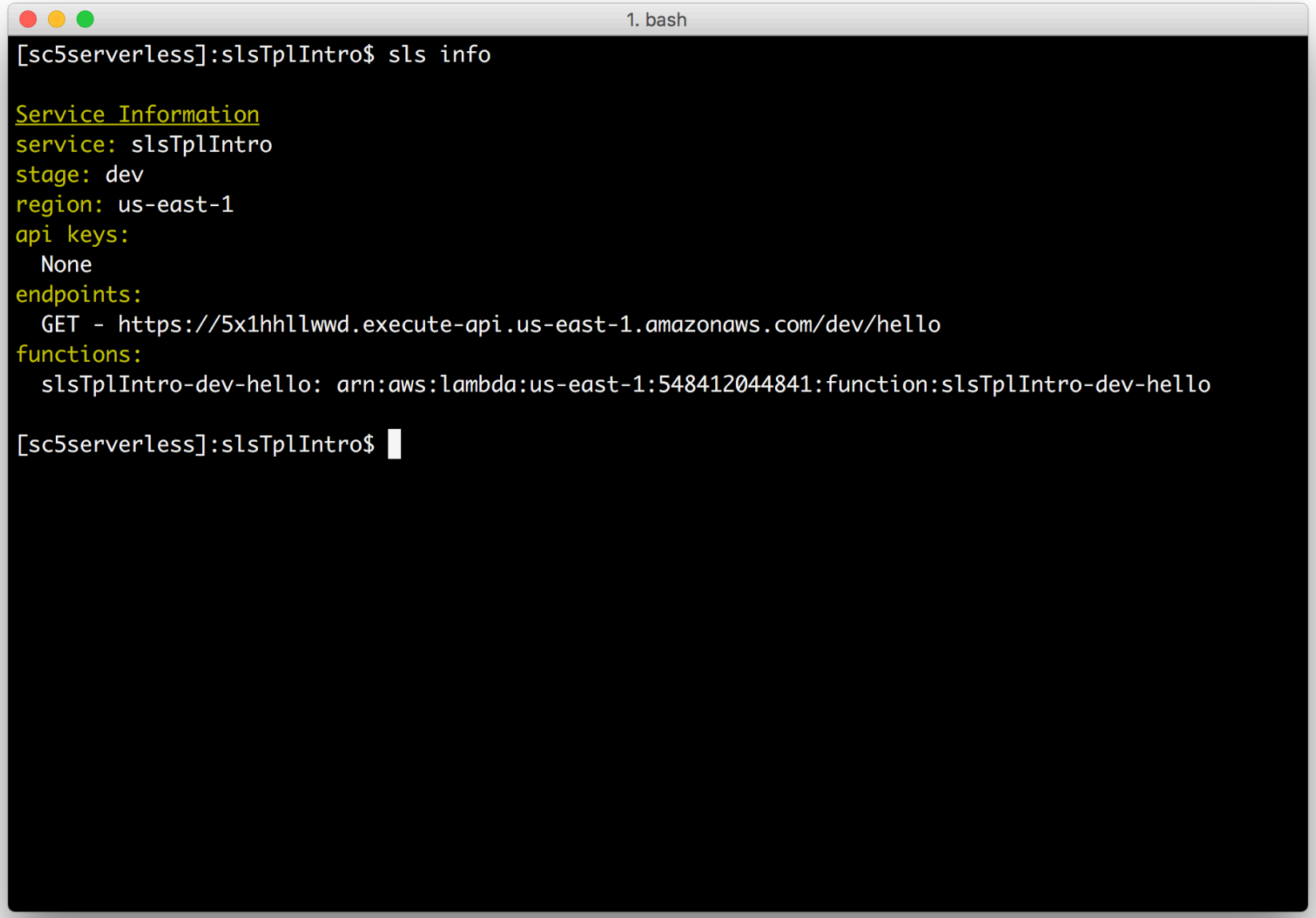
SC5


1. bash
[sc5serverless]:slsTplIntro$ sls deploy
Use –v for more verbose output
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$

SC5


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_c1AJ6OrxnkdazHjlPKDpqEvbxNw-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

SC5


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

SC5

OTHER BOILERPLATES

- serverless-messenger-boilerplate : Create bots using Messenger Platform and Wit.ai
- serverless-authentication-boilerplate : Create API Gateway custom authorizers

THANK YOU!

mikael.puittinen@sc5.io