SERVERLESS MESSENGER BOT WORKSHOP

Mikael Puittinen, Chief Technology Officer

Eetu Tuomala, Principal Cloud Specialist

25.10.2016

WORKSHOP OBJECTIVES

- Get familiar with AWS Services
- Get familiar with Serverless Framework 1.0
- Get familiar with Messenger Platform & Wit.ai
- Have fun building a weather bot on Messenger

AGENDA

09:30-10:30	Introduction to AWS Hands-on exercises: AWS Lambda / API Gateway
10:30-11:00	Break
11:00-12:30	Setup Node.js / AWS-SDK Learning Serverless 1.0 by doing Introduction to Messenger bot tech: MessengerPlatform, Wit.ai and our serverless messenger boilerplate
12:30-13:30	Lunch
13:30-15:00	Hand–on-exercises: building a weather bot
15:00-15:30	Break
15:30-17:30	Additional challenges Demos Feedback & discussion

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

This presentation is available for download at:

http://serverless.fi/docs/londonbots16.pdf

SC5

INTRODUCTION TO AWS

SC5

10 YEARS AGO...

MARCH 2006

Amazon Simple Storage Service (Amazon S3) launch

AUGUST 2006

Amazon Elastic Cloud Compute beta (Amazon EC2) launch

Building blocks for digital services

SOME AWS SERVICES FOR DEVELOPERS (10 OUT OF 55)

COMPUTE

 Lambda 2014

 ECS (EC2 Container Service) 2014

STORAGE

 S3 (Simple Storage Service) 2006

DATABASE

 DynamoDB 2012

 RDS (Relational Database Service) 2009

INTERNET OF THINGS

 IoT 2015

MOBILE SERVICES

 SNS (Simple Notification Service)

APPLICATION SERVICES

 API Gateway 2015

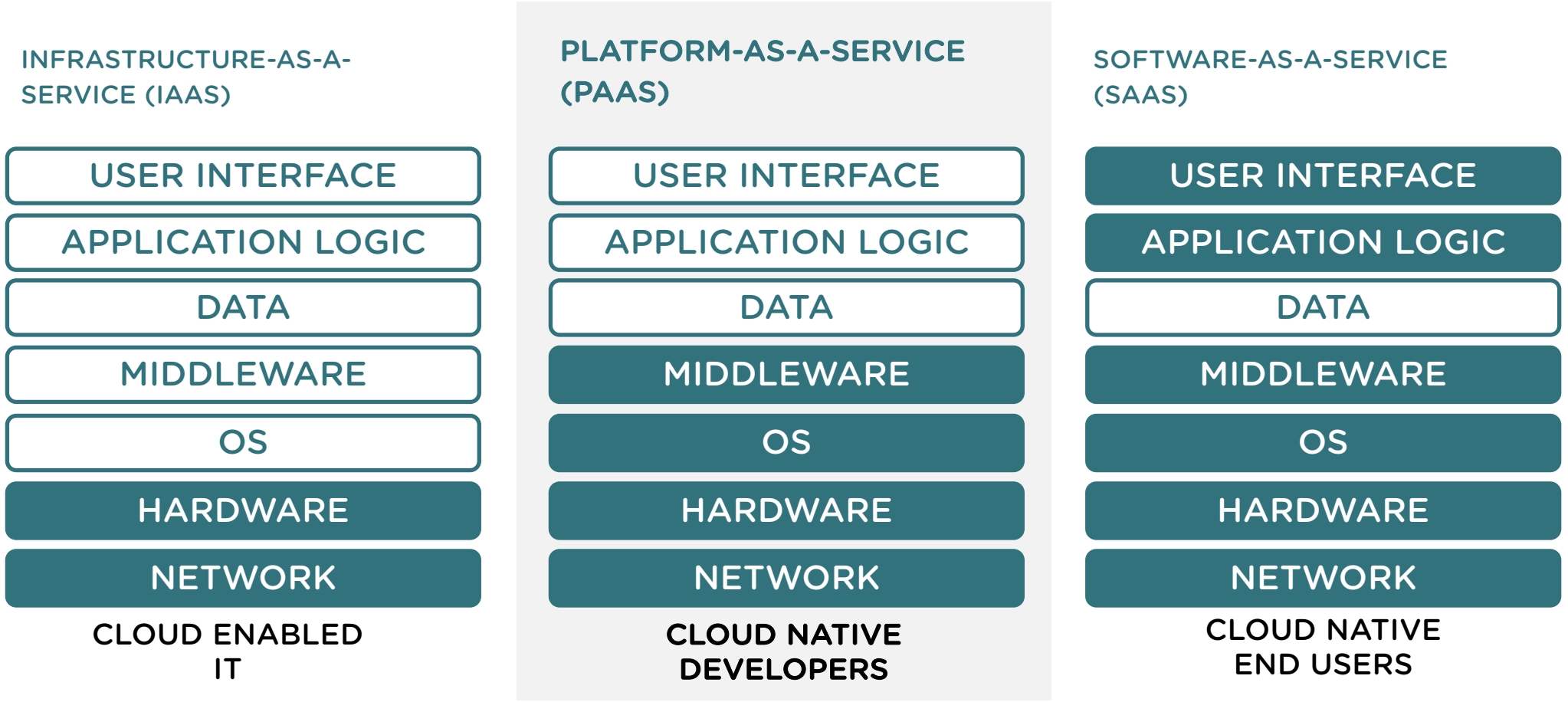
 SES (Simple Email Service) 2011

 SQS (Simple Queue Service) 2006

SC5

Pick the right cloud

DIFFERENT FLAVORS FOR DIFFERENT AUDIENCES



SC5

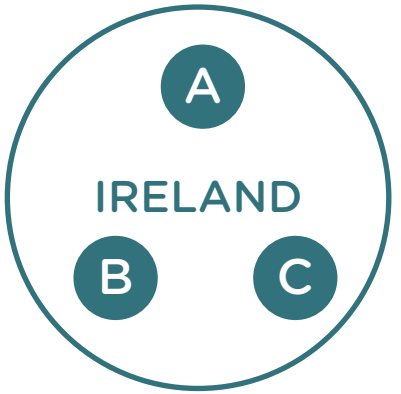
The foundation of the cloud

CLOUD (AWS) INFRASTUCTURE BASICS

REGIONS

Each region is independent, scope for running services. No automatic transfer of data / services cross-region.

DATA CENTERS



A region consists of multiple data centers. Multiple physically separated data centers provide fail-safe capabilities.

SC5

DEVELOPMENT ON DIFFERENT CLOUD FLAVORS

IAAS

- Think servers
- Work with command line / OS
- Need to plan durability (distribute over multiple data centres)
- Need to plan scalability
- Typically invoiced based on provisioned capacity

PAAS / MANAGED SERVICES

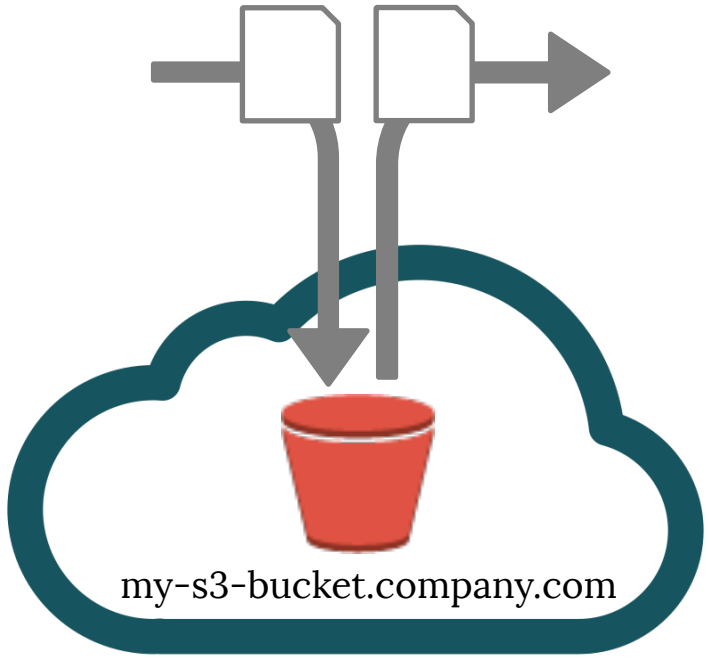
- Think services
- Work with APIs
- Durability built into managed services
- Scalability built into managed services
- Typically invoiced per actual use

DIVE INTO THE (AWS) MANAGED CLOUD SERVICES

SC5

SIMPLE STORAGE SERVICE (S3)

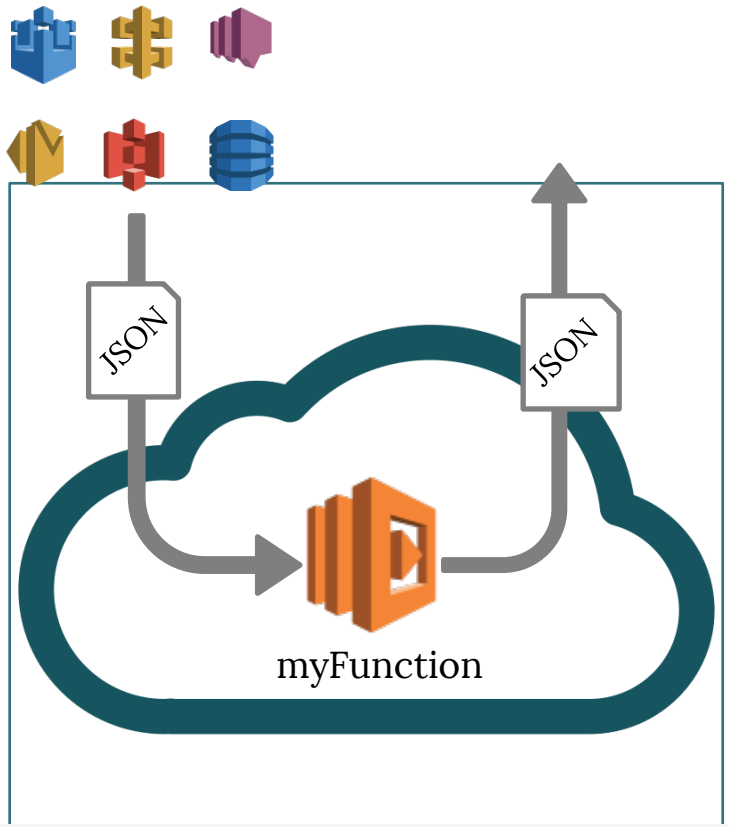
- (Unlimited) file storage service
- Application internal files
- Static web content (e.g. application HTML / CSS / JS / image assets)
- Can be complemented with CloudFront CDN to optimize costs and performance



PRICING: Storage volume + amount of requests

AWS LAMBDA

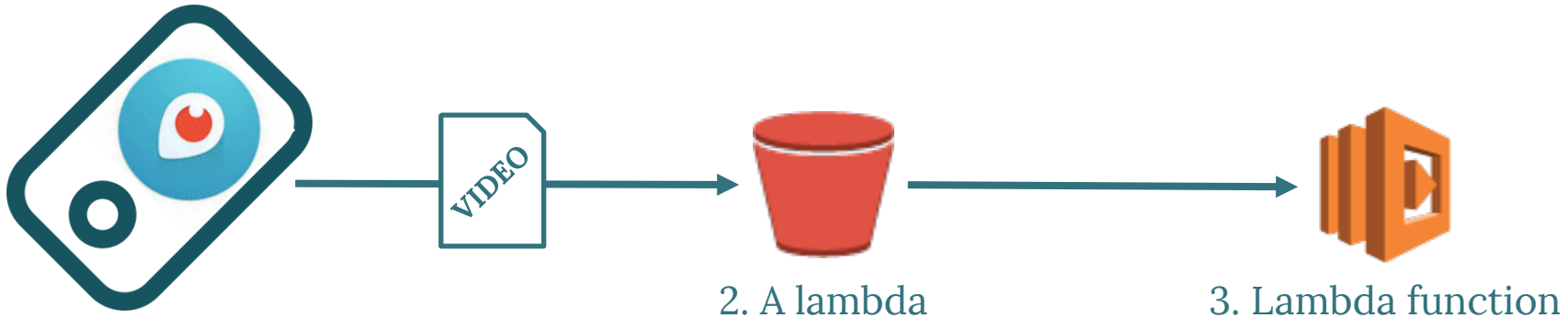
- Compute service for running code (functions) in AWS
- Event driven (API Gateway, SNS, SES, S3, DynamoDB, Schedule, ...)
- Provision memory & max time required by single function run
- Additional "instances" spawned automatically



PRICING: Utilized gigabyteseconds (rounded to 100ms)

Example 1

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

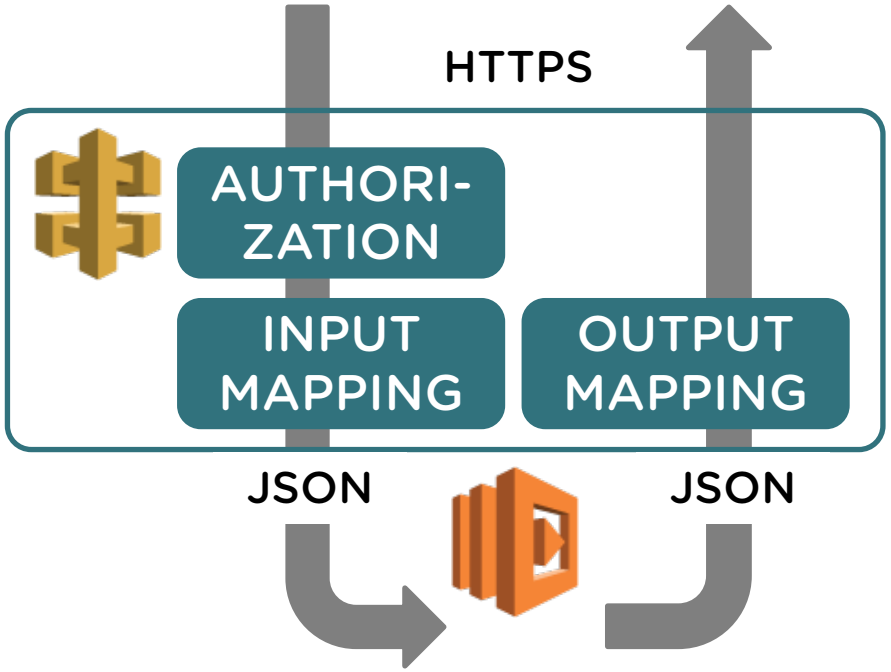
Logs are available in Cloudwatch

```
'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, customer 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 Database

DYNAMODB

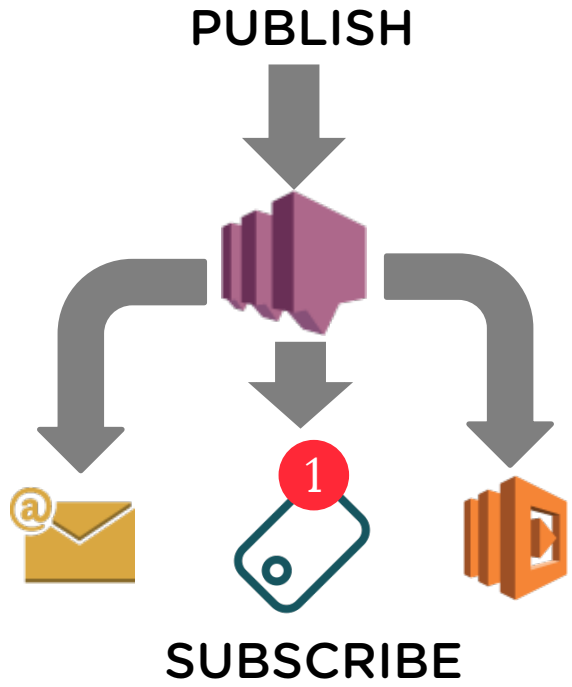
- noSQL database provided by AWS
- noSQL: scalable non-relational database with focus on speed
- Work with tables and indices, no server instances to manage
- Need to provision read / write capacity per table / index

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

SC5

SIMPLE NOTIFICATION SERVICE (SNS)

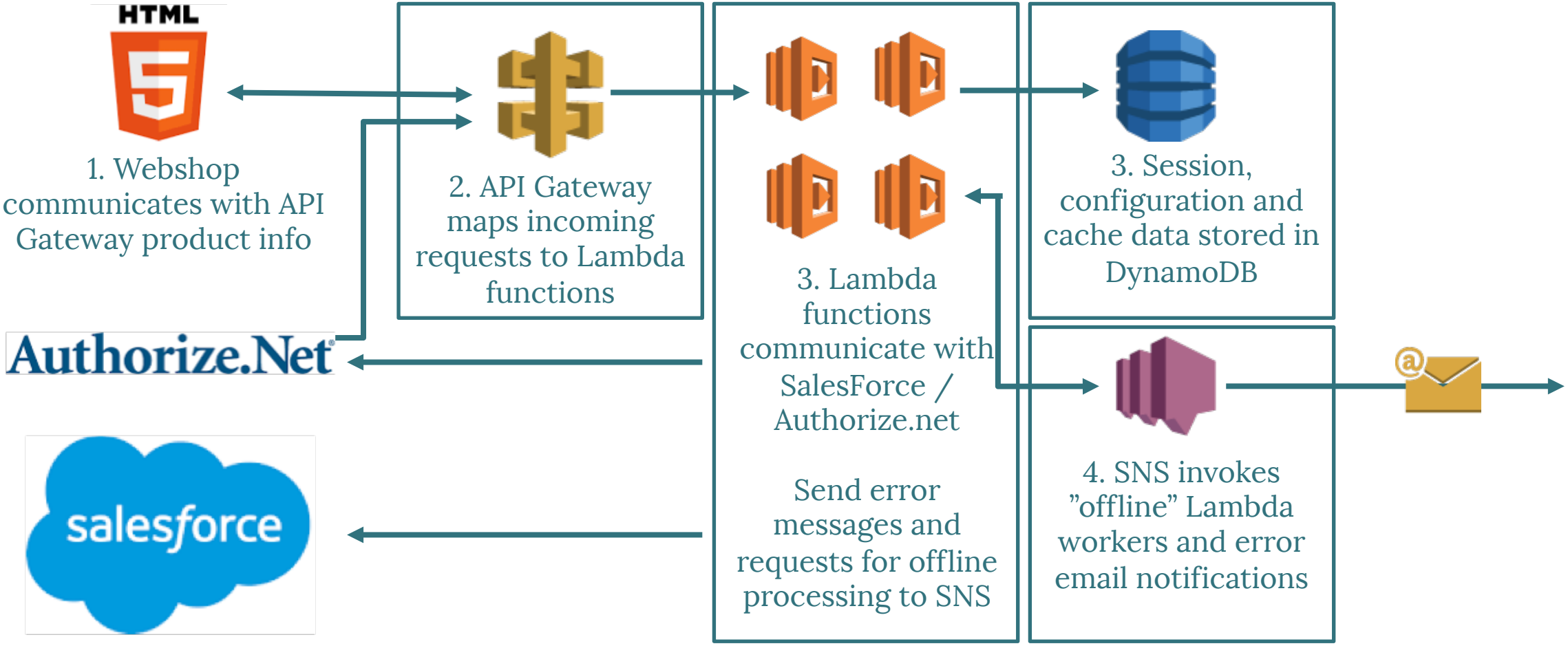
- Push notification service
- Mainly targeted for mobile notifications
- Can also be used for triggering e.g. Lambda functions, mobile, email notifications



PRICING: Amount of messages

Example 3

HAPPY OR NOT WEBSHOP



SC5

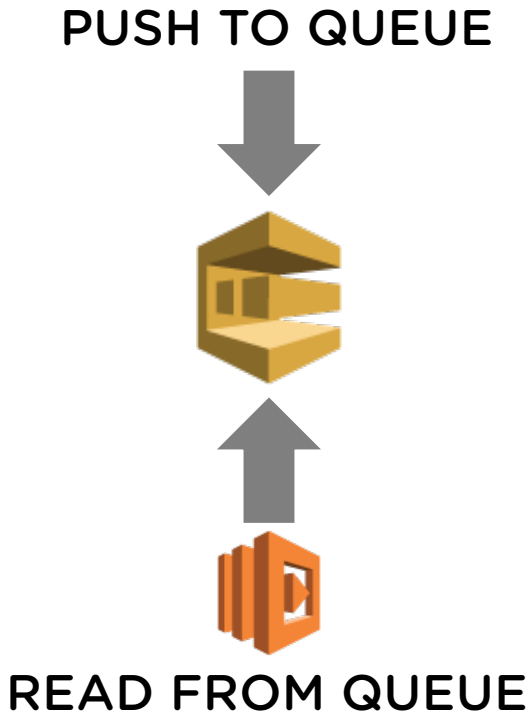
EXERCISE: SNS

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)

SIMPLE QUEUE SERVICE (SQS)

- Message queue service (pull)
- Delivery guaranteed
- Each message handled only once

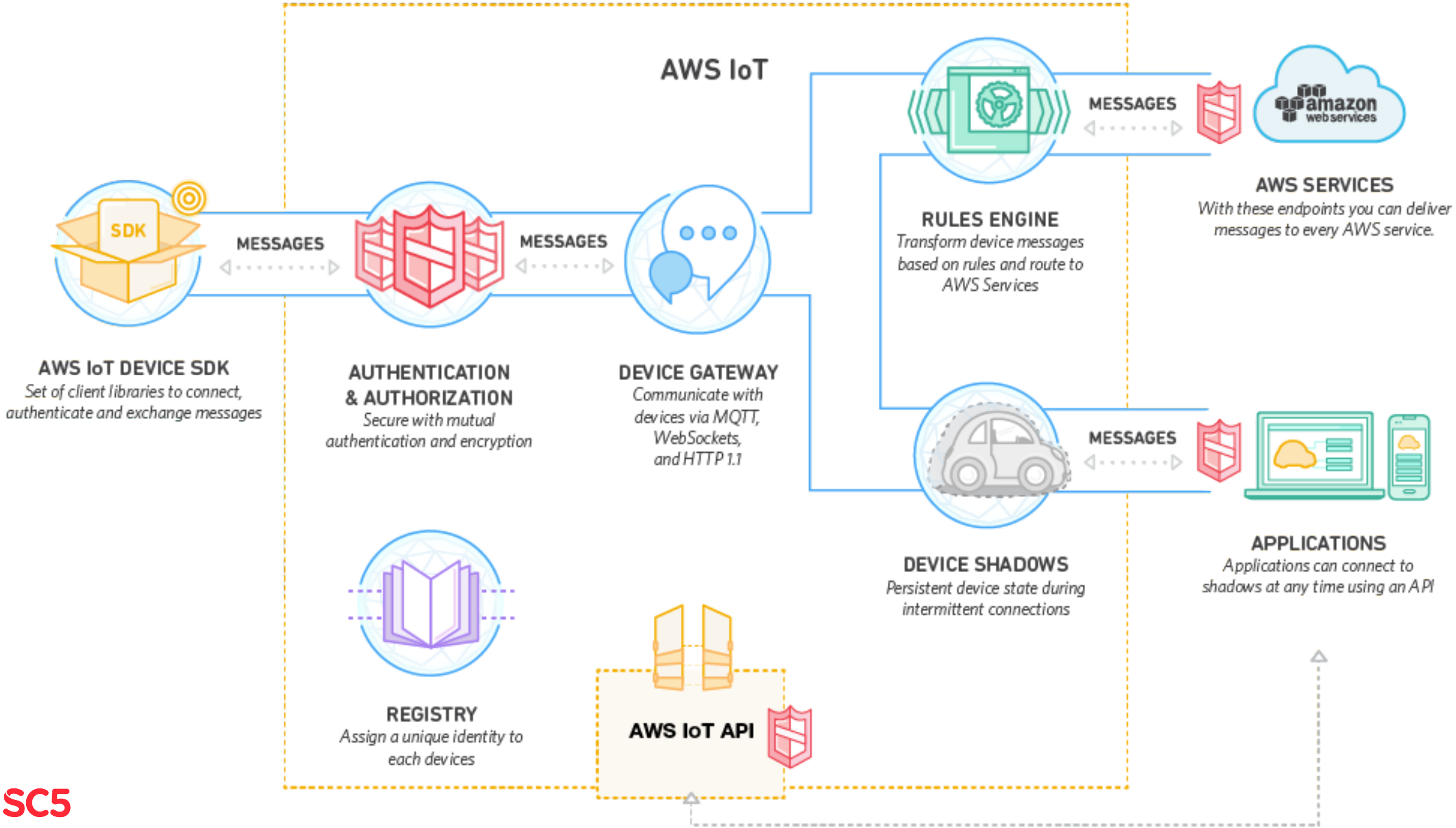


PRICING: Amount of messages

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)

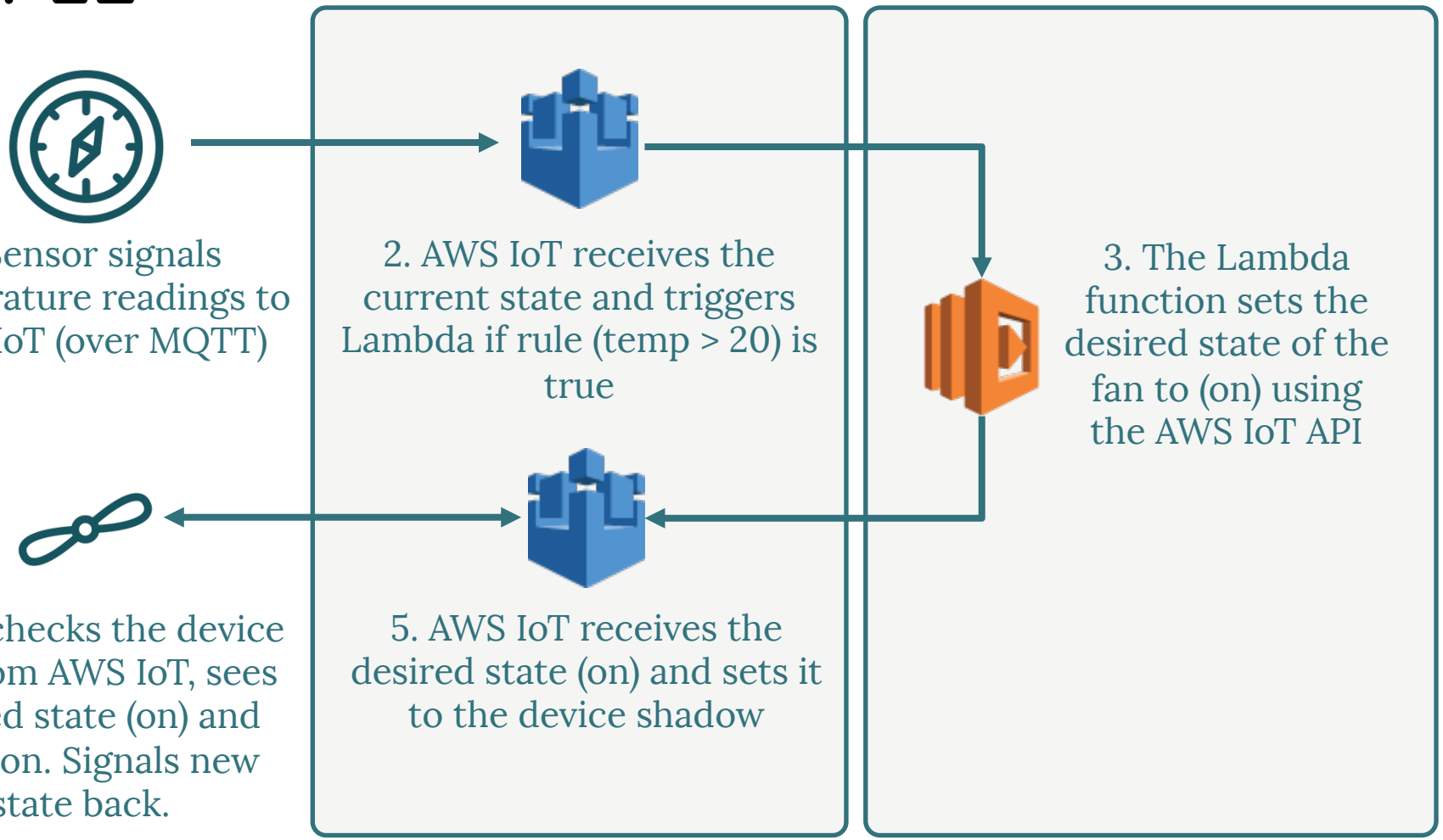
PRICING: Amount of messages


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 4

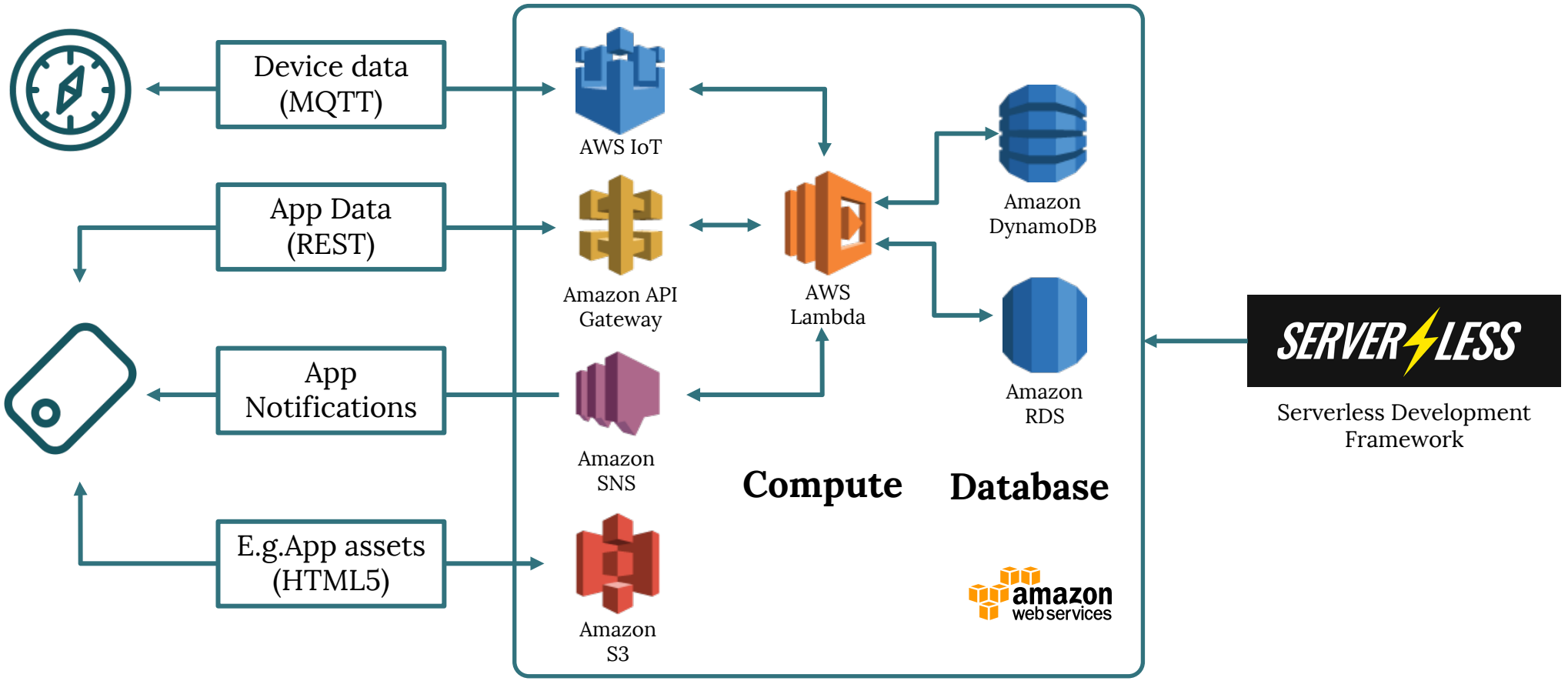
SIMPLE AWS IOT HOME AUTOMATION EXAMPLE



SC5

Reference Architecture

CLOUD NATIVE APPLICATION ARCHITECTURE TOOLS À LA SC5



SC5

OTHER CLOUD VENDORS

How about other cloud vendors?

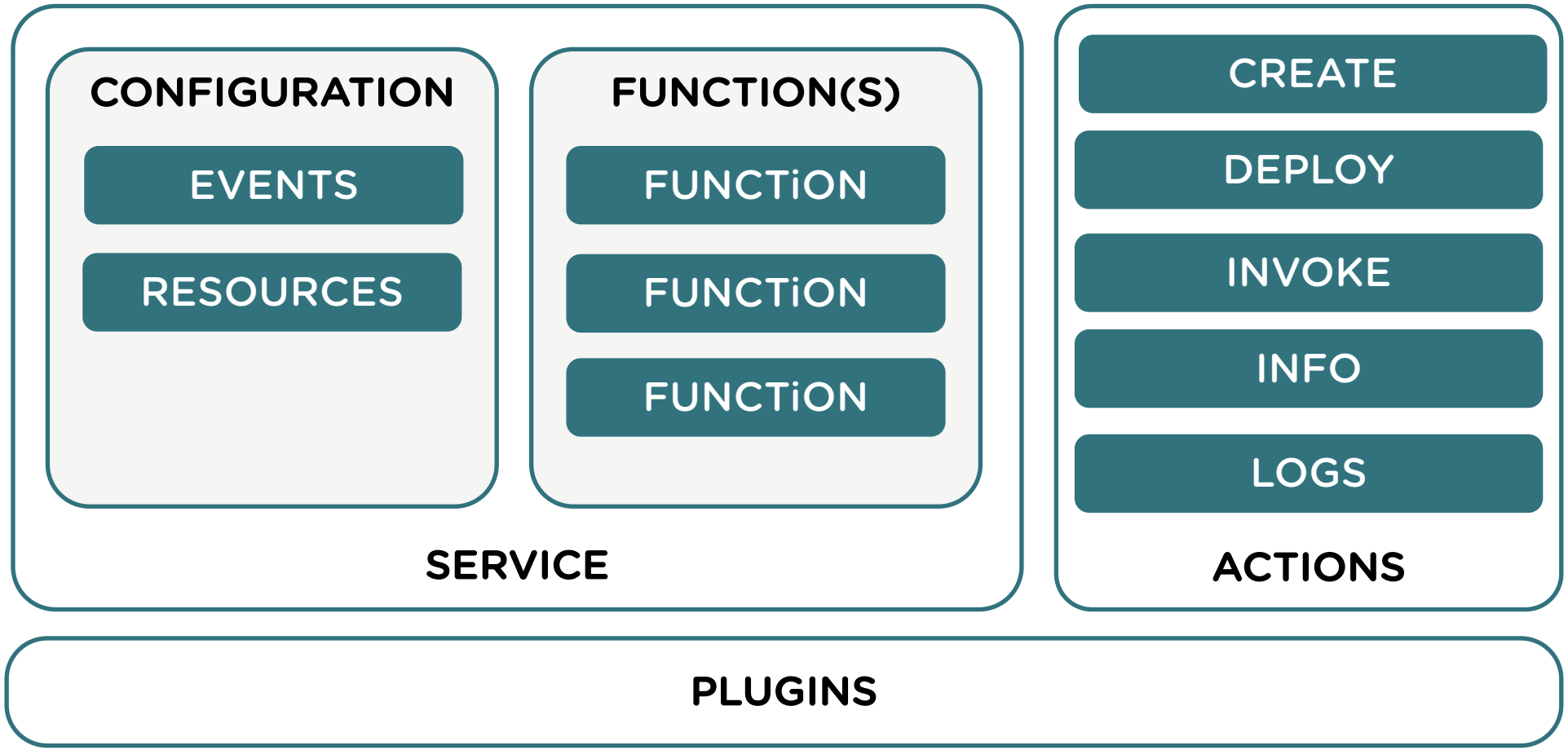
SIMILAR OFFERING ALSO FROM OTHER CLOUD VENDORS

AWS	MICROSOFT AZURE	GOOGLE CLOUD	IBM BLUEMIX
S3	Azure Storage	Google Cloud Storage	Object Storage
DynamoDB	DocumentDB	Google Cloud Datastore	Cloudant
Lambda	Azure Functions (preview)	Google Cloud Functions (alpha)	OpenWhisk (experimental)
SNS	Azure Notification Hub	Google Cloud Pub/Sub	IBM Push Notifications
SQS	Azure Queue Storage	Google Cloud Pub/Sub	IBM MQ
IoT	Azure IoT Suite	Use GC building blocks	Watson IoT

INTRODUCTION TO SERVERLESS FRAMEWORK 1.0

SC5

SERVERLESS FRAMEWORK 1.0



SC5

GETTING SET UP

Instructions for getting set up:

http://serverless.fi/docs/workshop-preps

LEARNING SERVERLESS BY DOING

SC5

CREATING A SERVICE

- We'll start by creating a service "slsIntro"

```
> sls create –t aws-nodejs –n slsIntro –p slsIntro
```

- Parameters explained:
  - *-t aws-nodejs* : use template for create Node.js based AWS service
  - *-n slsIntro* : name the service "slsIntro"
  - *-p slsIntro* : create the service into directory slsIntro

AWS-NODEJS STRUCTURE

`event.json`	Input file for *invoke* action
`handler.js`	(Default) source file for handler.js
`serverless.yml`	Service configuration (functions, endpoints, resources)

Lets's check contents for each file!

SC5

TUNE THE FUNCTION

- Dump the response also with *console.log* for debugging purposes
- Add e.g. *date: new Date()* to the body of your response.

ADD SERVERLESS-MOCHA-PLUGIN TO YOUR SERVICE (FOR LOCAL DEBUGGING)

- Initialize your package.json

```
> npm init
```

- Install serverless-mocha-plugin

```
> npm install —save-dev serverless-mocha-plugin
```

Register the plugin to *serverless.yml* by adding *the following:*

```
plugins:
  - serverless-mocha-plugin
```

SC5

CREATE AND INVOKE SERVERLESS-MOCHA-PLUGIN TESTS

- Create a test for function hello with

```
> sls create test –f hello
```

- Check the test contents from *test/hello.js*
- Invoke the test with

```
> sls invoke test –f hello
```

- Fix test to pass

DEFINE HTTP ENDPOINTS FOR FUNCTION

- In serverless.yml, add the event definitions (below in red)
- Note: rooted path (e.g. /hello) do not currently work

```
functions:
  hello:
    handler: handler.hello
    events:
      - http:
          path: hello
          method: get
      - http:
          path: hello
          method: post
```

SC5

DEPLOY THE FUNCTIONS AND ENDPOINTS

- Deploy the function with

```
> sls deploy
```

- You could use –s <*stage*> to deploy to other stages (e.g. prod)
- Use Postman / curl to test the endpoints. The endpoint URL is available after deployment or with

```
> sls info
```

- Log output with

```
> sls logs –f hello -t
```

- See what happens to the event with the default API Gateway mapping in Serverless 1.0
- Try adding *integration: lambda* to your endpoints and redeploy

ADD SERVERLESS-OFFLINE PLUGIN TO YOUR SERVICE (FOR LOCAL DEBUGGING)

- Install the serverless-offline plugin

```
> npm install –save-dev serverless-offline
```

- Register the plugin to *serverless.yml* by adding the following to plugins:

```
- serverless-offline
```

- Start the and try the endpoint

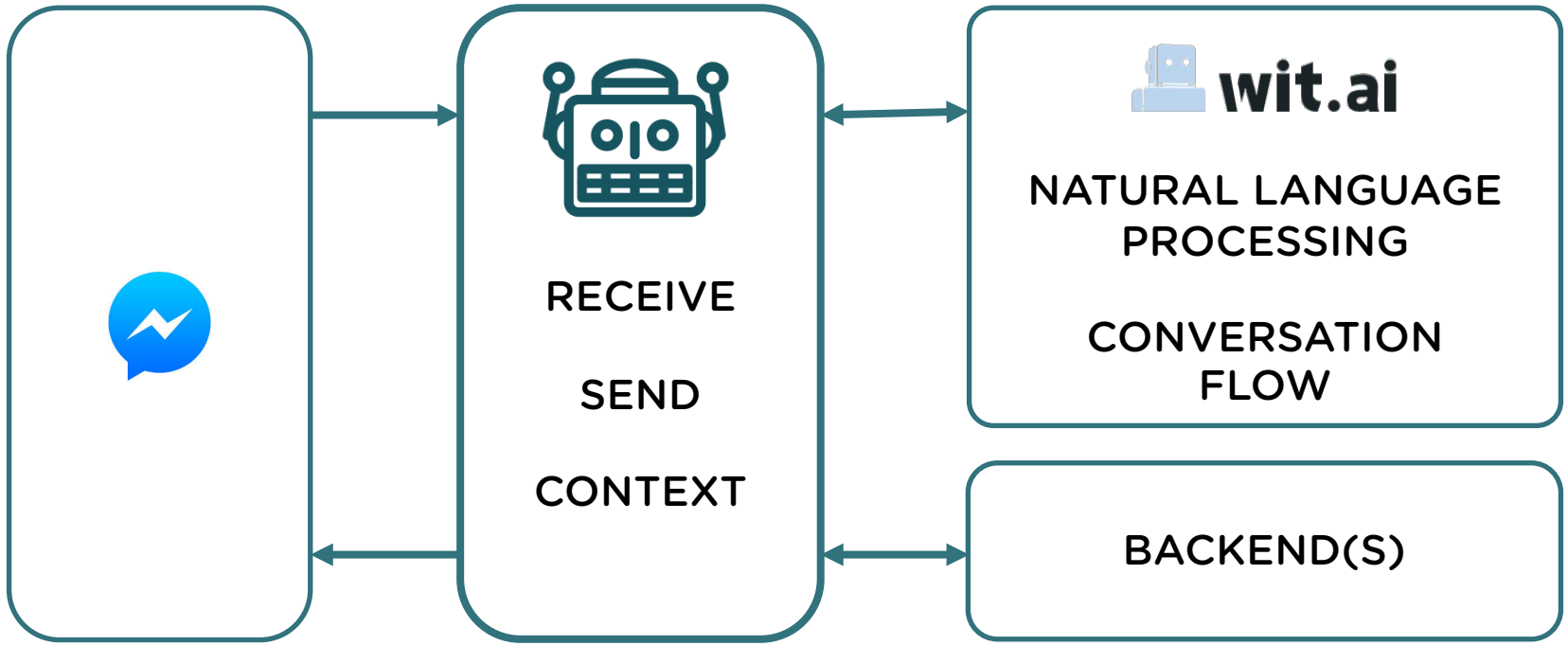
```
> sls offline start

> curl http://localhost:3000/dev/hello?foo=bar
```

SERVERLESS-WEBPACK PLUGIN

- Optimize package size and ensure that all 3rd party modules are included in your deployment
- We'll check that with during the bot exercise

INTRODUCTION TO MESSENGER BOT COMPONENTS

MESSENGER BOT ARCHITECTURE



SC5

MESSENGER PLATFORM

SC5

Messenger Platform Beta Launched April 2016

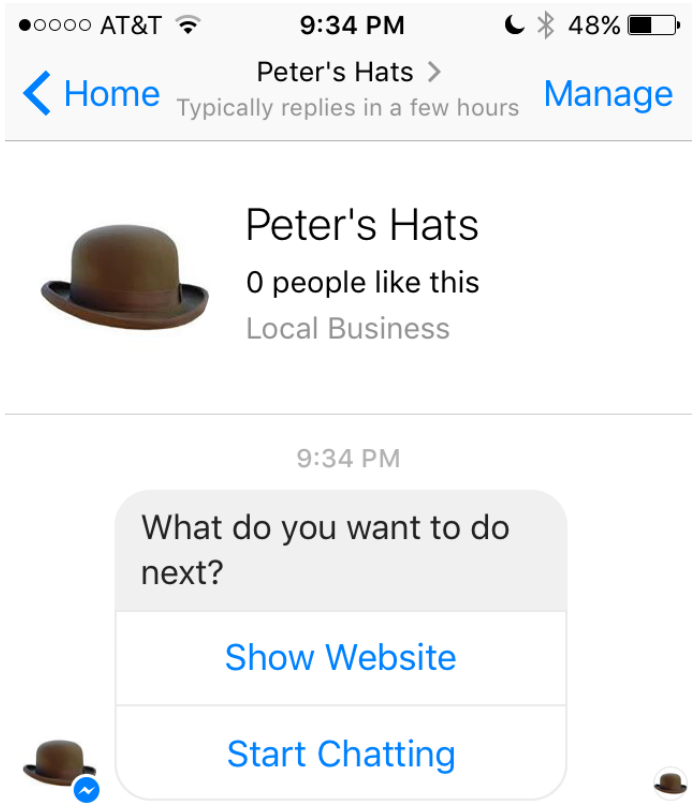
Over 1Bn monthly users

Over 33k bots

RICH UI ELEMENTS: TEXT

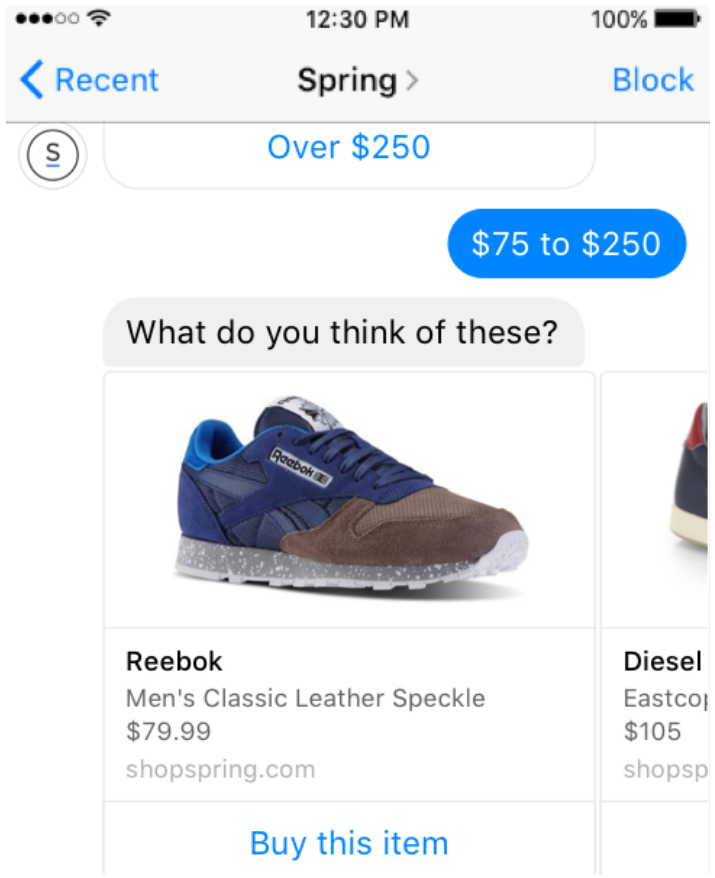
SC5

RICH UI ELEMENTS: BUTTONS



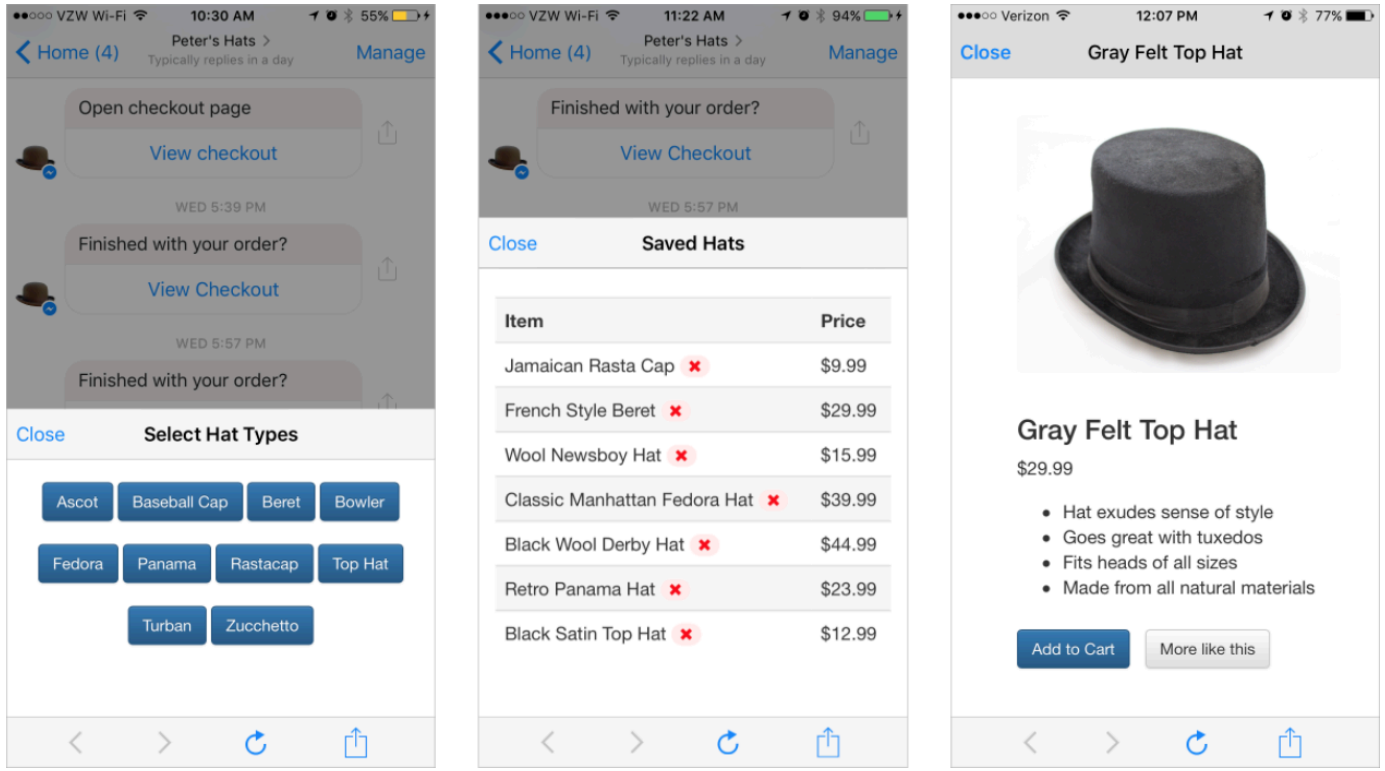
SC5

RICH UI ELEMENTS: CAROUSEL

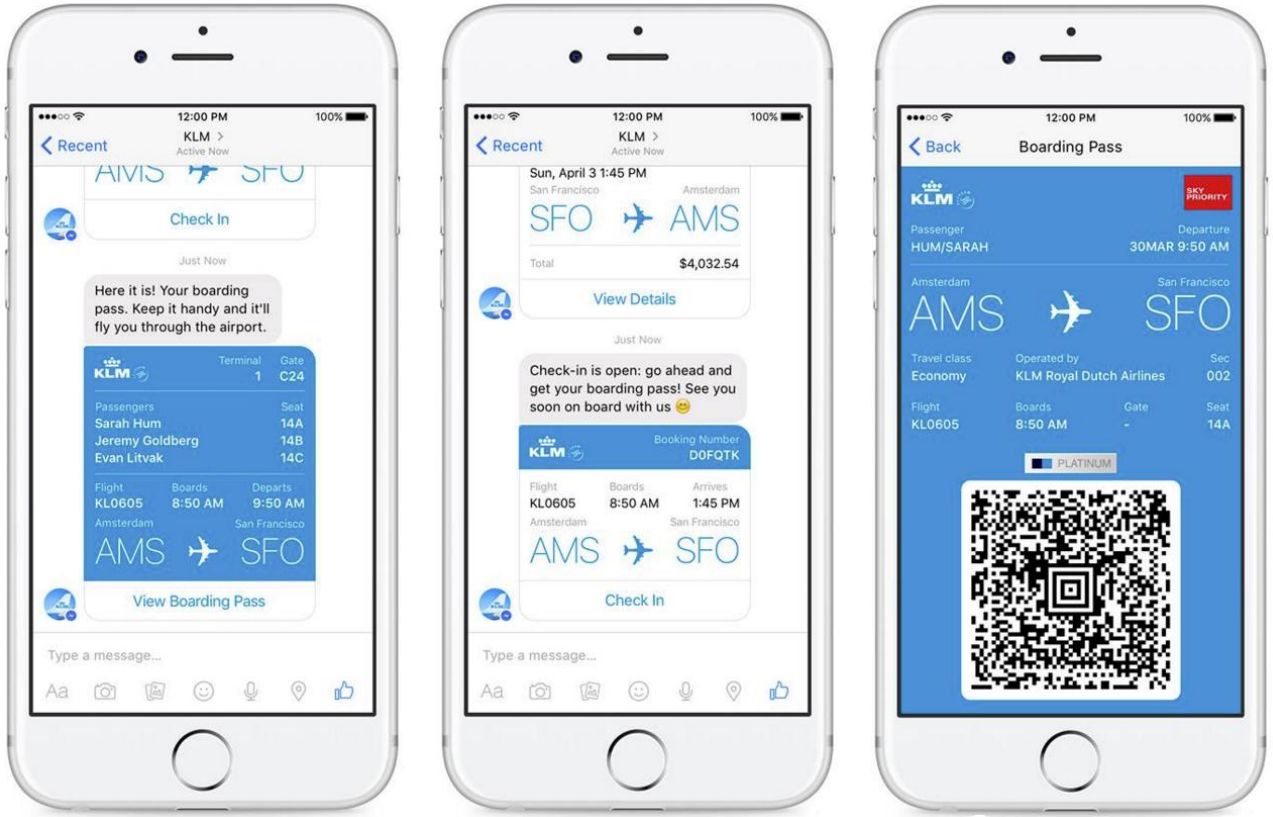


SC5

RICH UI ELEMENTS: WEB VIEWS



EXAMPLE: KLM MESSENGER



SC5

MESSENGER BOT SETUP QUICK GUIDE

1. Create a Facebook page
2. Register a Facebook application
3. Create an endpoint for your bot and hook it to the Facebook app
4. Listen to messages from Messenger
5. Post back responses to the Messenger Platform
6. Get your Facebook application approved to reach the public

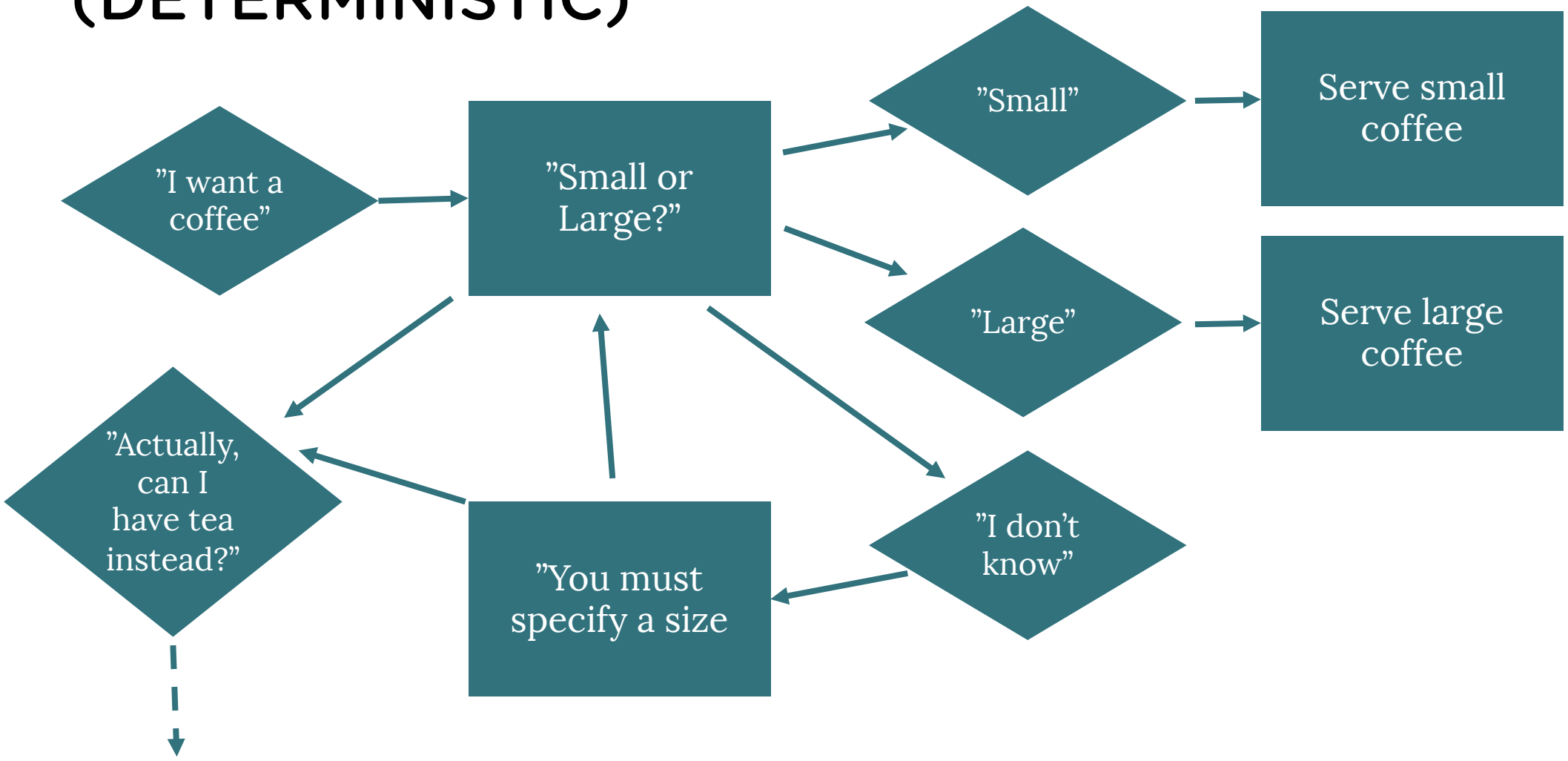
SC5

Messenger Platform API documentation available at:

https://developers.facebook.com/docs/messenger-platform/

WIT.AI

CONVERSATION WORKFLOW (DETERMINISTIC)



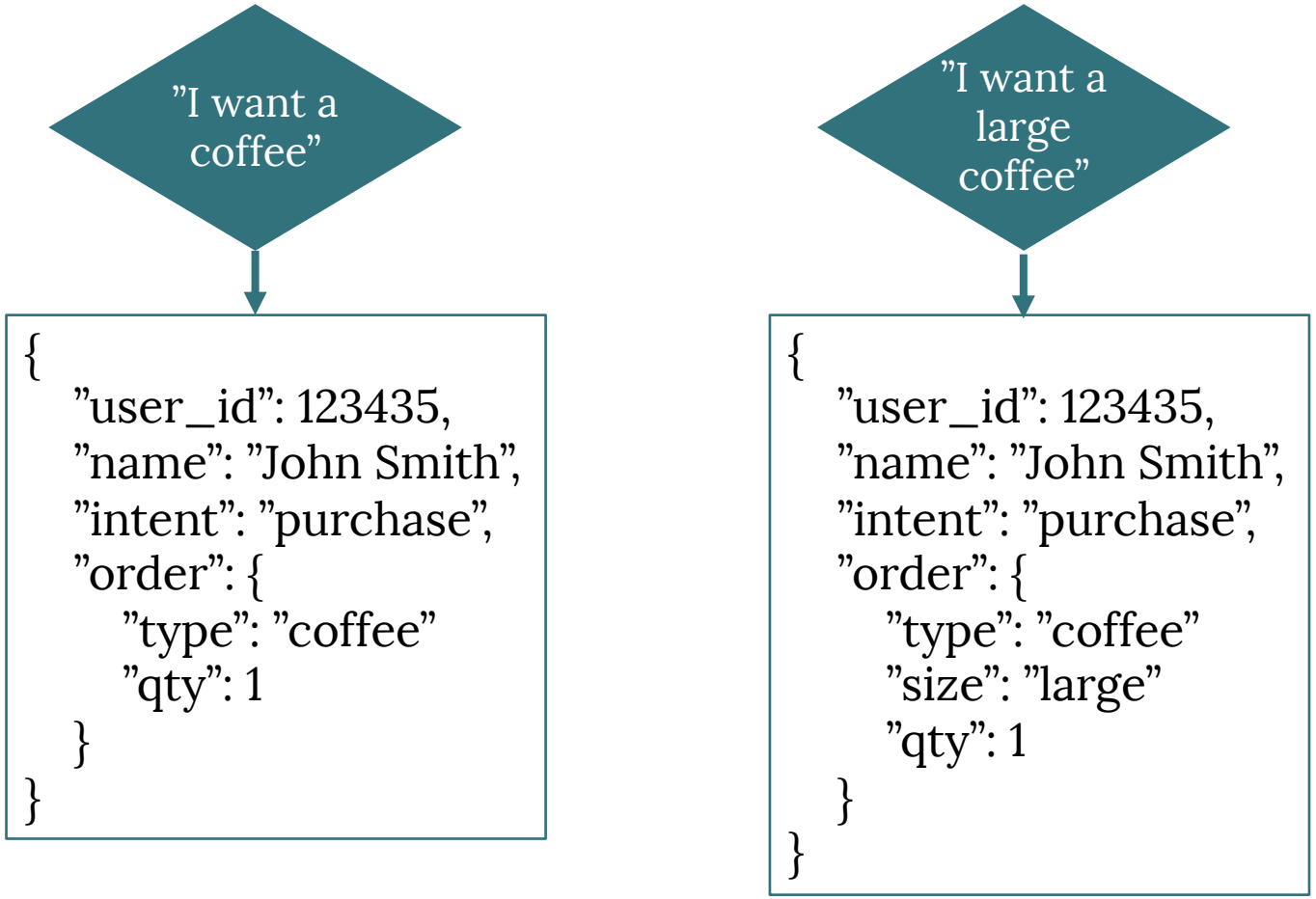
SC5

WIT.AI APPROACH

- Stories to define ”rules”/”heuristics” for the conversation
- Context to define current state

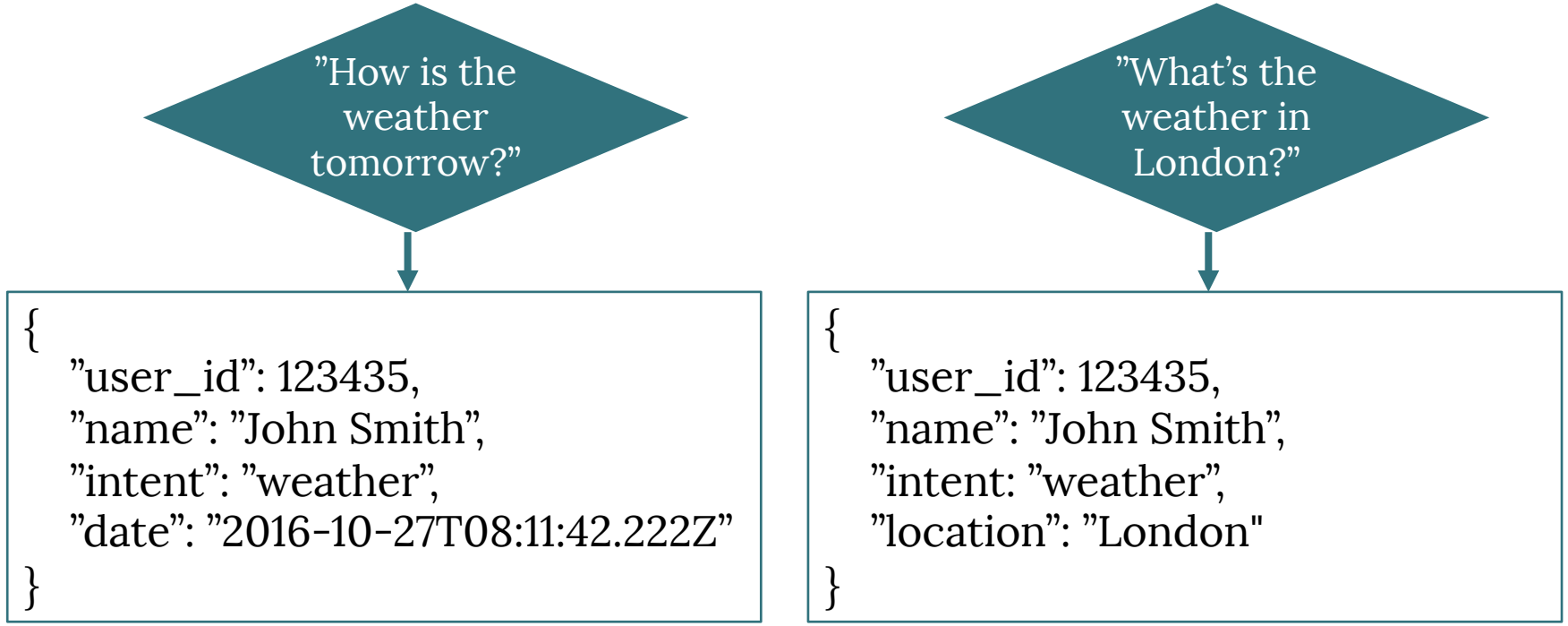
=> No need for hard coded flows. A combination of stories with rules based on context allow to define complex converstations

CONTEXTS



SC5

NATURAL LANGUAGE PROCESSING



SC5

SAMPLE WIT.AI CONVERSATION (1)

REQUEST

What's the weather in Brussels?

RESPONSE


```
{
  "type": "merge",
  "entities": {
    "location": [{
      "body": "Brussels",
      "value": {
        "type": "value",
        "value": "Brussels",
        "suggested": true},
      "start": 11,
      "end": 19,
      "entity": "location"}
    ]
  },
  "confidence": 1
}
```

SAMPLE WIT.AI CONVERSATION (2)

REQUEST


```
{
  ”loc”: ”Brussels”
}
```


RESPONSE


```
{
  “type”: “action”,
  “action”: “fetch-forecast”
}
```

SAMPLE WIT.AI CONVERSATION (3)

REQUEST


```
{
  ”loc”: ”Brussels”,
  ”forecast”: ”Sunny”
}
```


RESPONSE


```
{
  “type”: “msg”,
  “msg”: “It’s gonna be sunny in Brussels”
}
```

The *node-wit* module provides higher level method *runActions*() that hides the logic of iterating the contexts with Wit.ai.

Wit.ai HTTP API documentation available at:

https://wit.ai/docs/http

Node.js SDK available at

https://github.com/wit-ai/node-wit

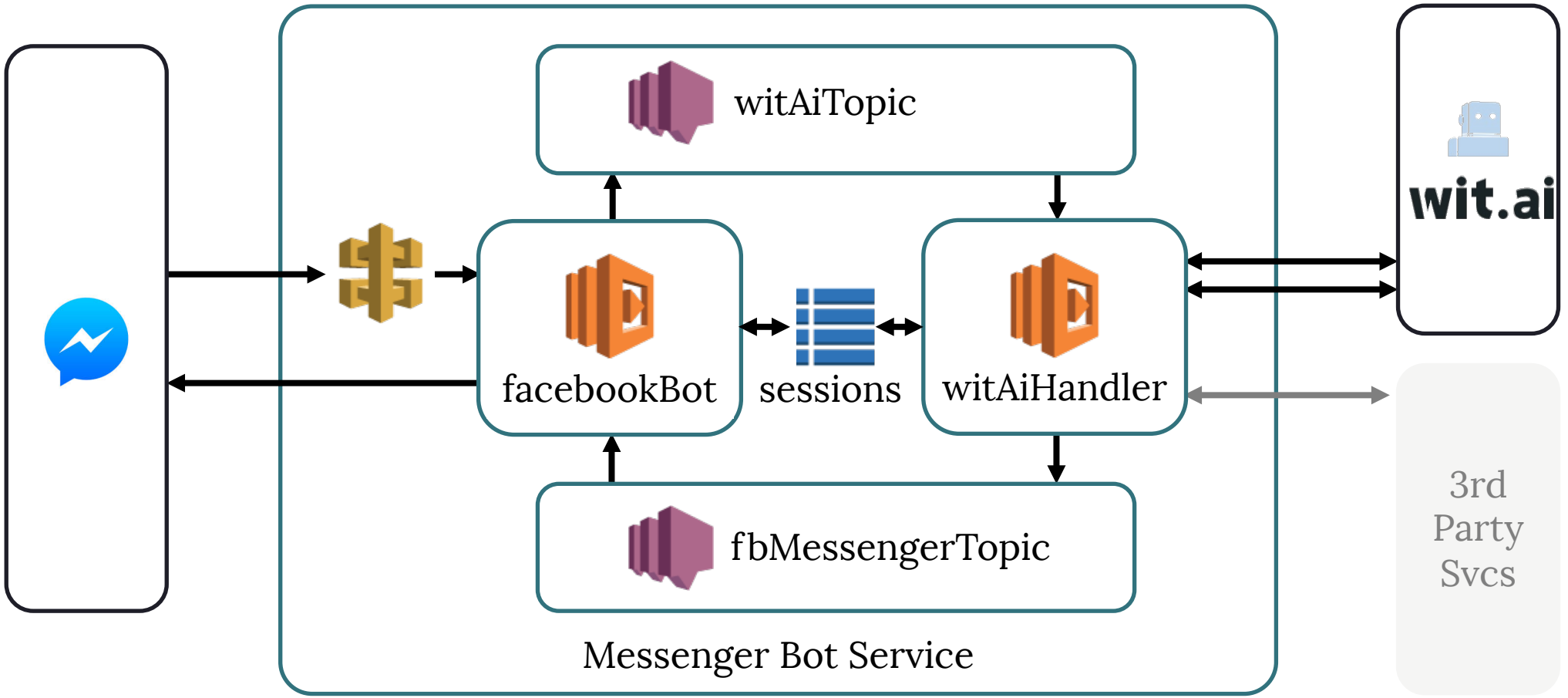
OUR BOILERPLATE

SC5

serverless-messenger-boilerplate and its documentation available at

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

BOILERPLATE ARCHITECTURE



SC5

BOILERPLATE STRUCTURE

`example.env`	Template for .env (Facebook / Wit.ai secrets) <= COPY TO .env
`facebook-bot/`	facebookBot function
`  fb-messenger.js`	Facebook Messenger interop
`  handler.js`	Function entrypoint
`lib/`	Common libraries used by both functions
`  messageQueue.js`	Library for reading / publishing to SNS
`  session.js`	Library for reading / updating user session (DynamoDB)
`package.json`	ADD NODE MODULES REQUIRED BY YOUR BOT HERE
`serverless.yml`	Service configuration (functions, endpoints, resources)
`test/`	Serverless-mocha-plugin tests
`  facebookBot.js`	Tests for facebookBot function
`  witAiHandler.js`	Tests for witAiHandler function
`webpack.config`	Configuration for Webpack deployment (serverless-webpack plugin)
`wit-ai/`	witAiHandler function
`  handler.js`	Entrypoint for function
`  my-wit-actions.js`	Your bot logic (Wit.ai actions) <= *IMPLEMENT YOUR BOT LOGIC HERE*
`  wit-ai.js`	Bot logic built interop on Wit.ai

SC5

.ENV FILE (FROM EXAMPLE.ENV)

FACEBOOK_BOT_VERIFY_TOKEN	User defined verification token for the Facebook App
FACEBOOK_BOT_PAGE_ACCESS_TOKEN	Facebook-generated access token for the page
WIT_AI_TOKEN	API token for Wit.ai
FACEBOOK_ID_FOR_TESTS	Facebook ID used to post messages in tests. Available from the sessions table
SERVERLESS_PROJECT	Service name (keep in sync with service name in serverless.yml)

DECOUPLING WITH SNS

- Lambda SNS subscriptions: see serverless.yml
- Posting and decoding messages done with messageQueue.js
- Sample endpoint at facebookBot/handler.js
- In production environments, SNS could be strengthened e.g. with SQS to guarantee delivery

LOCAL DEVELOPMENT AND TESTING (SERVERLESS-MOCHA-PLUGIN)

- serverless-mocha-plugin included in boilerplate with predefined tests for facebookBot and witAiHandler functions in the *test/* directory
- Run all tests : *sls invoke test*
- Run tests for facebookBot : *sls invoke test –f facebookBot*
- Run tests for witAiHandler: *sls invoke test –f witAiHandler*
- (Create new tests with *sls create test –f functionName*)
- Comment out line that sets *process.env.SILENT* in test/*.js if you want messages to be sent during testing

OPTIMIZED DEPLOYMENT (SERVERLESS-WEBPACK)

- serverless-webpack plugin included to streamline deployment package and accelerate cold start of Lambda functions
- Pre-configured in *webpack.config*

BUILDING A WEATHER BOT

Code snippets for the workshop are available from

http://serverless.fi/docs/messenger-workshop/

SETUP PROJECT

1. CREATE SERVERLESS PROJECT

```
> sls install –u https://github.com/SC5/serverless-messenger-boilerplate
> mv serverless-messenger-boilerplate weather-bot
```

Change service name to “weather-bot” in *serverless.yml* and *.env*, then

```
> npm install
> cp example.env .env
```

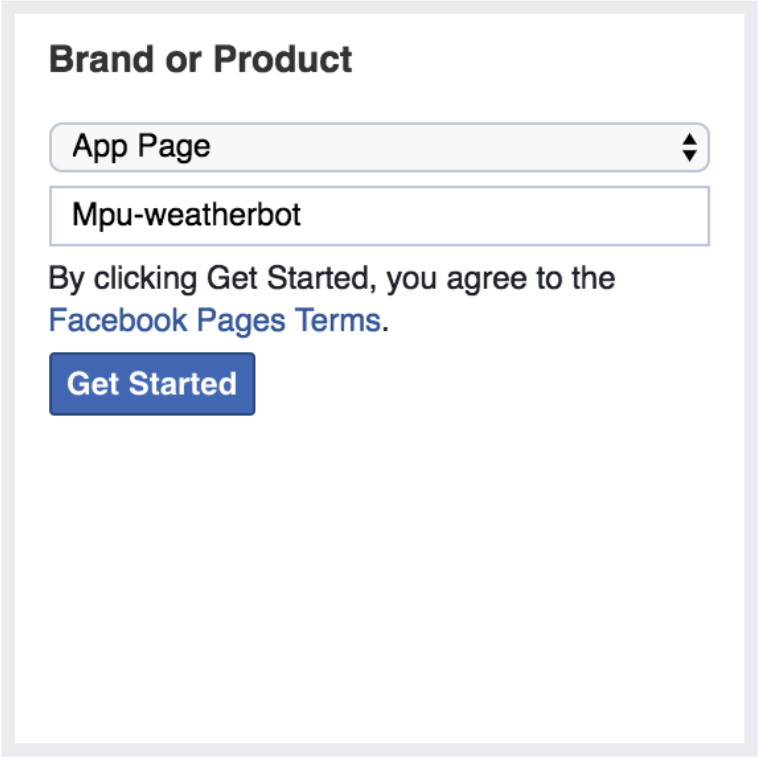
This creates a new service *weather-bot* based on *serverless-messenger-boilerplate*, installs the node modules required by the project.

SETUP FACEBOOK PAGE AND APPLICATION

SC5

CREATE A FACEBOOK PAGE

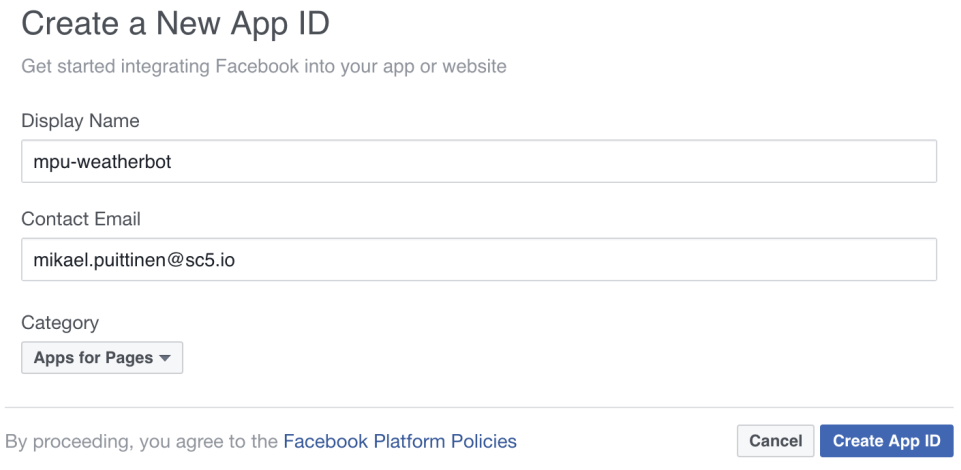
1. Go to http://facebook.com/pages/create
2. Create new page
3. Skip all following steps



SC5

CREATE FACEBOOK APPLICATION

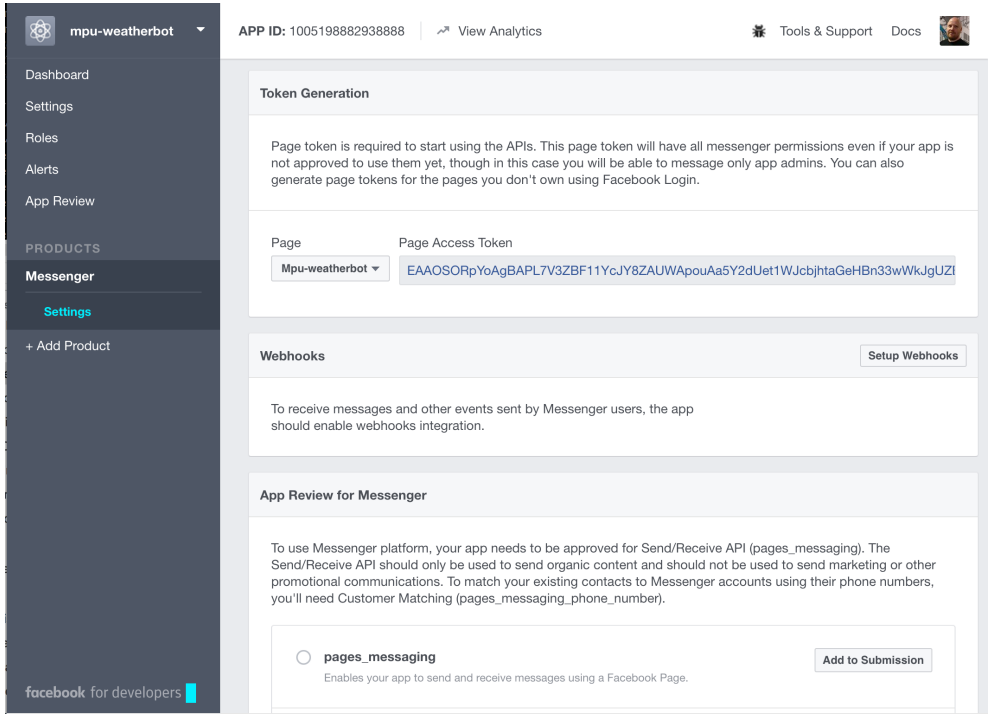
1. Go to https://developers.facebook.com/quickstarts/?platform=web
2. Click ”Skip and Create App ID”
3. Fill in info
4. Create page



SC5

SETUP MESSENGER TOKEN

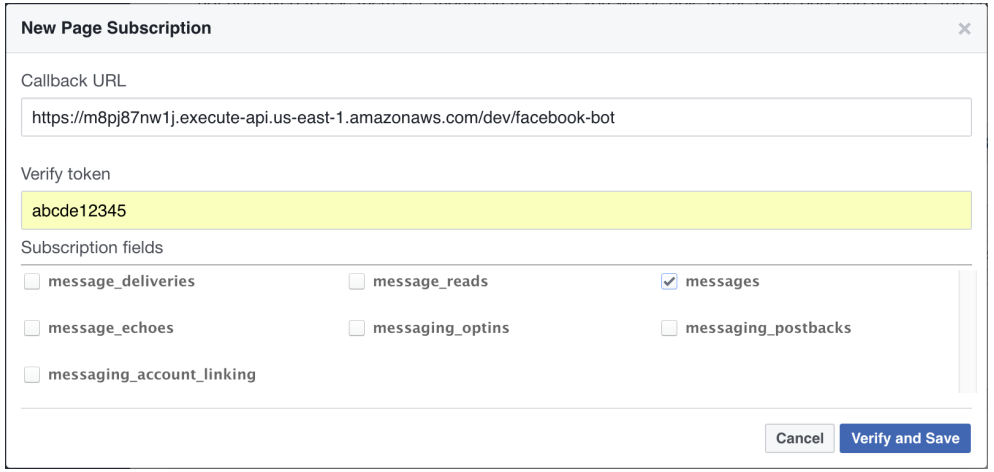
1. ”Get Started” for Messenger
2. Generate a token for your page
3. Copy token to .env (*FACEBOOK_BOT_TOKEN*)



SC5

SETUP MESSENGER WEBHOOK

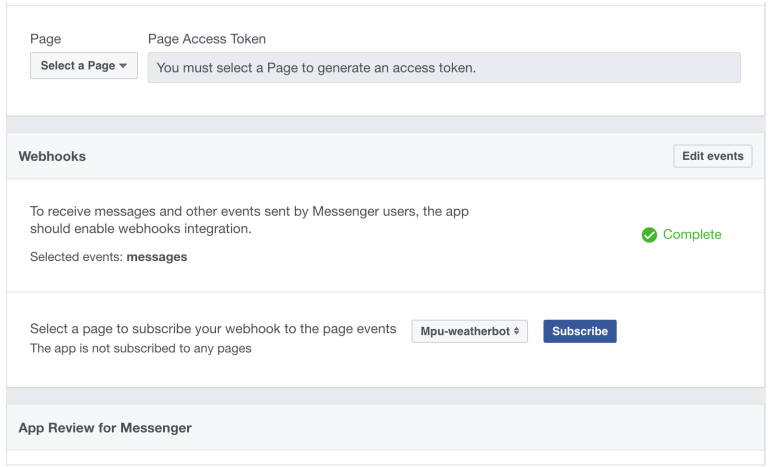
1. Initiate ”Setup Webhooks”
2. Enter the endpoint URL that you got during deployment
3. Enter your verify token from .env (*FACEBOOK_VERIFY*)



SC5

SUBSCRIBE THE WEBHOOK TO THE FACEBOOK PAGE

1. Under ”Webhooks”, select your page and subscribe



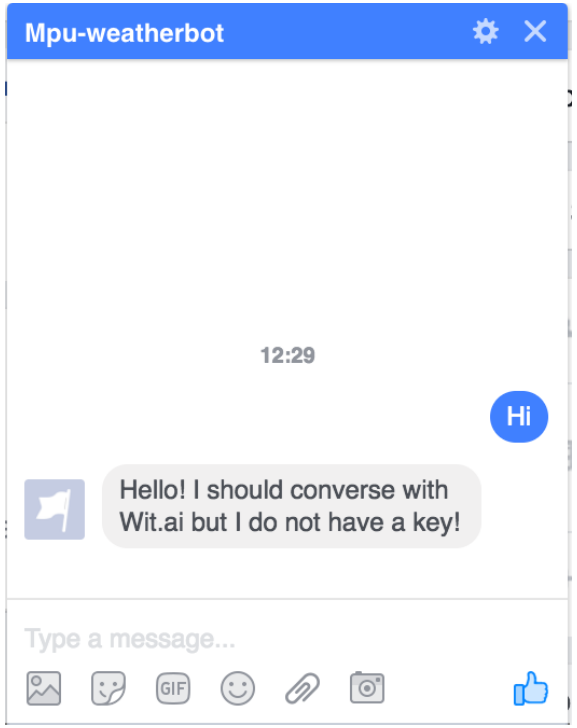
SC5

TRY IT

- Deploy your service

```
> sls deploy function –f facebookBot
```

- Send message to your Facebook page



SC5

FACEBOOK APPLICATION ACCESS

Access to unapproved applications is limited to developers only.

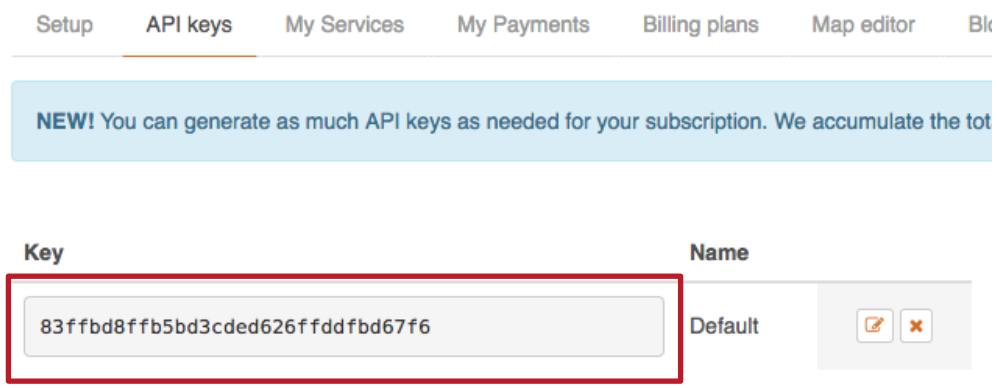
If you want to give access to other users, you should provide ”Developer” / ”Testers” roles to them from the ”Roles” panel (accessible from the left sidebar)

SET UP OPENWEATHER API

SC5

REGISTER TO OPENWEATHERMAP

1. Go to https://openweathermap.org/appid
2. Sign up
3. Copy API key and set it to *WEATHER_API_TOKEN* in .env

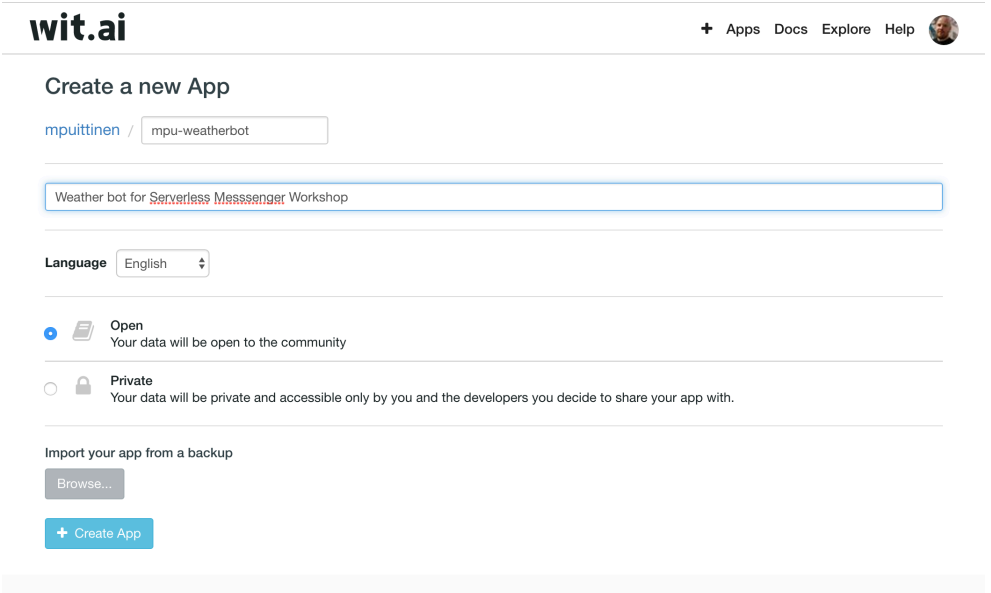


SC5

SETUP WIT.AI

REGISTER + CREATE WIT.AI APP

1. Go to https://wit.ai
2. Register (if not already done)
3. Create a new App

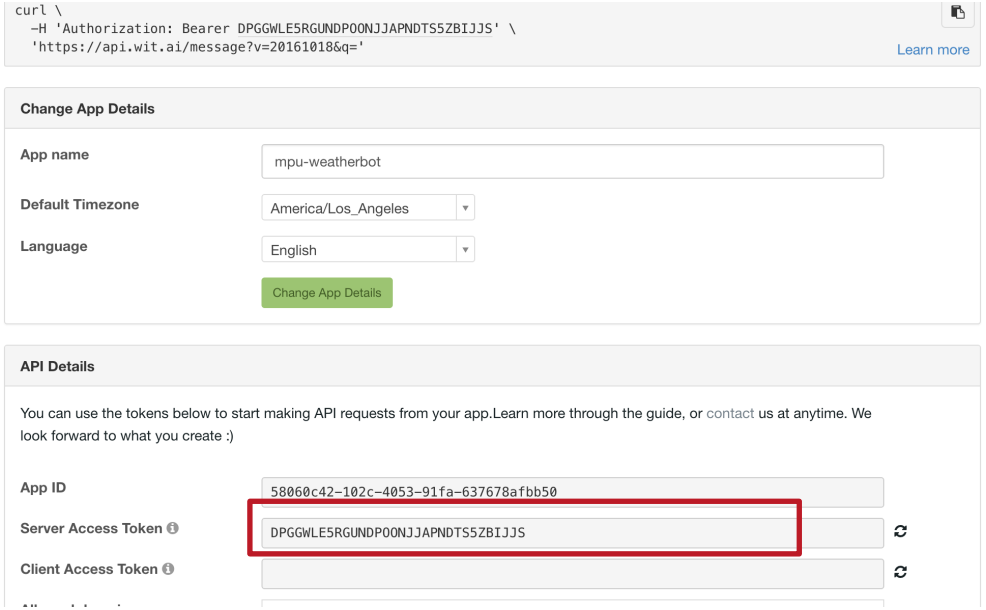


SC5

CONNECT THE ENDPOINT WITH WIT.AI

1. Go to Settings
2. Copy the Server Access Token ID to .env (*WIT_AI_TOKEN*)
3. Deploy your service function

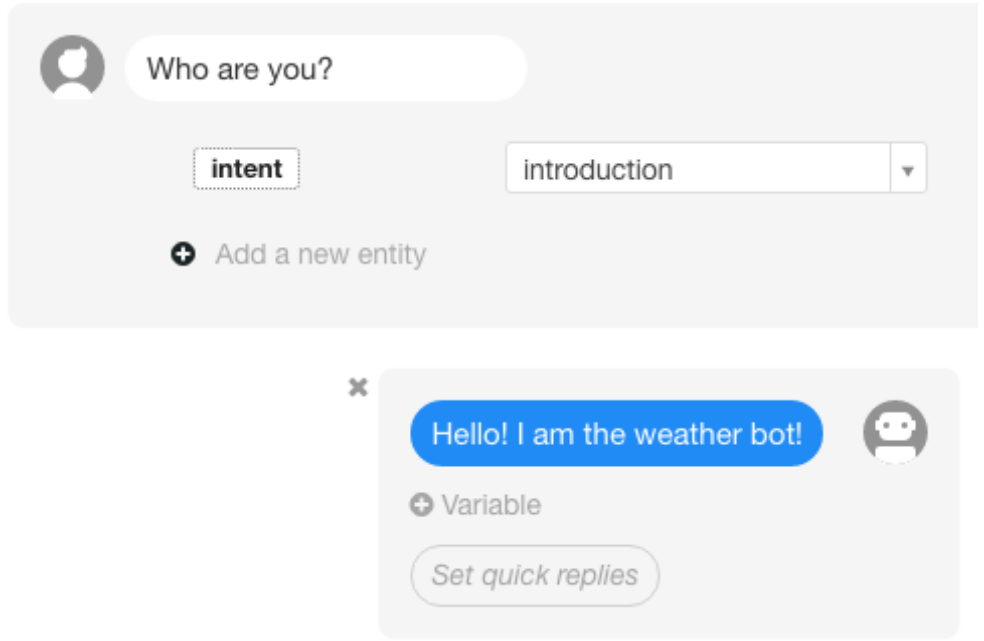
```
> sls deploy function –f facebookBot
```



SC5

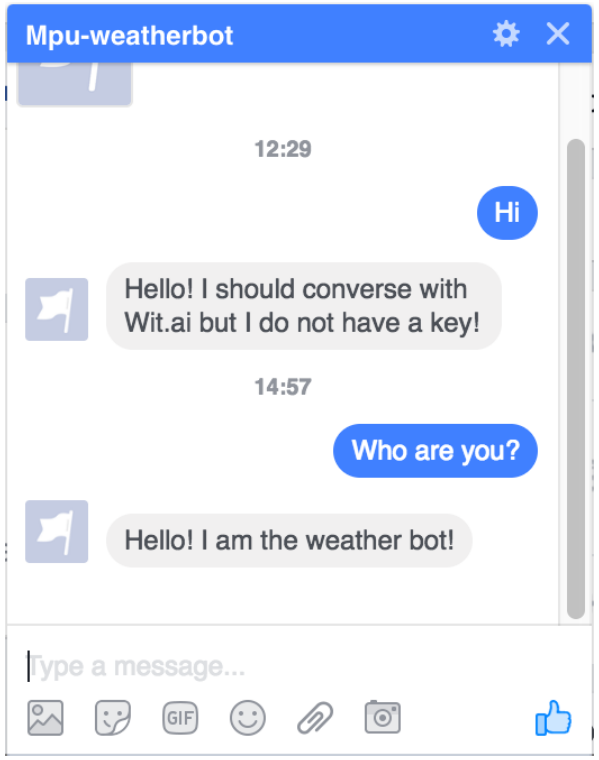
LET THE BOT INTRODUCE ITSELF

1. Create a new story
2. Add user input and intent
3. Add a response with ”Bot Sends”
4. Remember to save the story



SC5

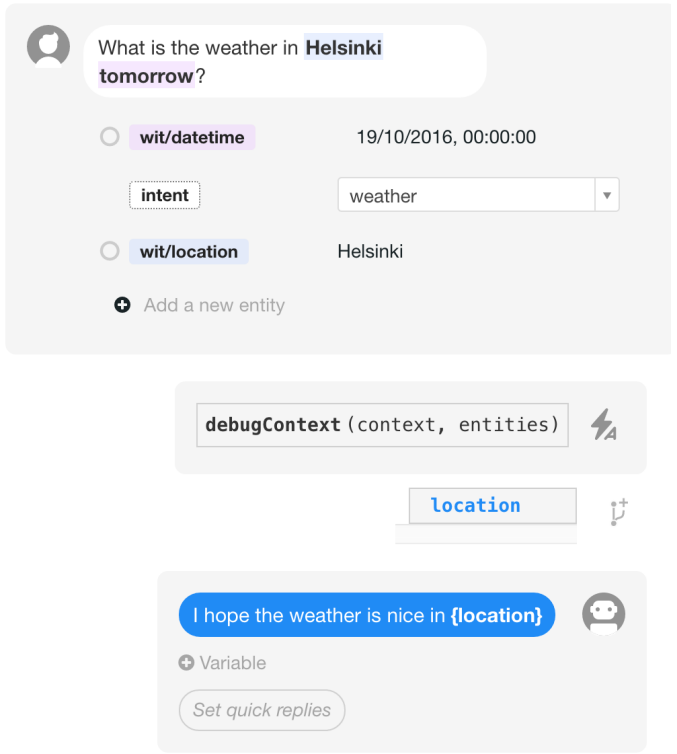
TRY IT!


Mpu-weatherbot
12:29
Hi
Hello! I should converse with Wit.ai but I do not have a key!
14:57
Who are you?
Hello! I am the weather bot!
Type a message...
GIF

SC5

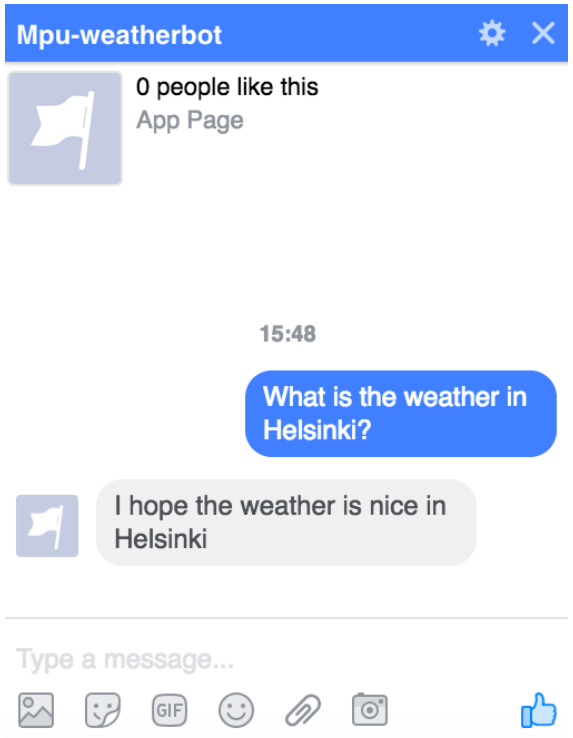
WEATHER STORY (WITH FIXED RESPONSE)

1. Add a new story
2. Type in your user input
3. Add entity *intent* with value *weather*
4. Select the location and add new entity *wit/location*
5. Select the date and add new entity *wit/datetime*
6. Add action *debugContext* with ”Bot Executes”
7. Set context key *location*
8. Add a response using ”Bot sends”
9. Save



SC5

TRY IT (WITH DEBUG)!



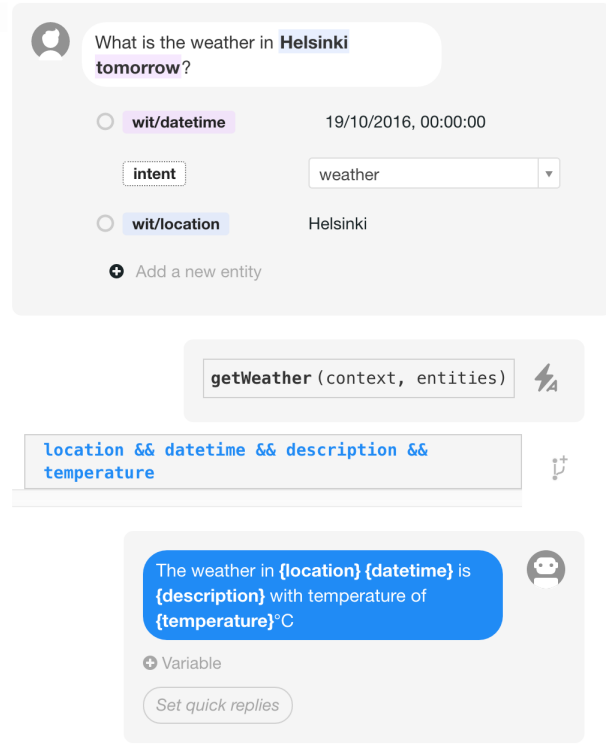

```
> sls deploy function –f facebookBot
```

```
START RequestId: bc745ce1-9530-11e6-8a4c-f7159450ad15 Version: $LATEST
2016-10-18 15:45:23.932 (+03:00)        bc745ce1-9530-11e6-8a4c-f7159450ad15    WIT.AI DEBUG:
2016-10-18 15:45:23.932 (+03:00)        bc745ce1-9530-11e6-8a4c-f7159450ad15    {
  "sessionId": "948003351971940-1476794723470",
  "context": {},
  "text": "What is the weather in Helsinki?",
  "entities": {
    "location": [
      {
        "confidence": 0.9998750679246946,
        "type": "value",
        "value": "Helsinki",
        "suggested": true
      }
    ],
    "intent": [
      {
        "confidence": 0.9995693812970181,
        "value": "weather"
      }
    ]
  }
}
END RequestId: bc745ce1-9530-11e6-8a4c-f7159450ad15
REPORT RequestId: bc745ce1-9530-11e6-8a4c-f7159450ad15  Duration: 684.66 ms     Billed Duration:
700 ms          Memory Size: 1024 MB      Max Memory Used: 63 MB
```

SC5

ADD LOGIC TO THE STORY

1. Remove action *debugContext* and context key *location*
2. Add action *getWeather*
3. Add context key *location && datetime && description*
4. Add response

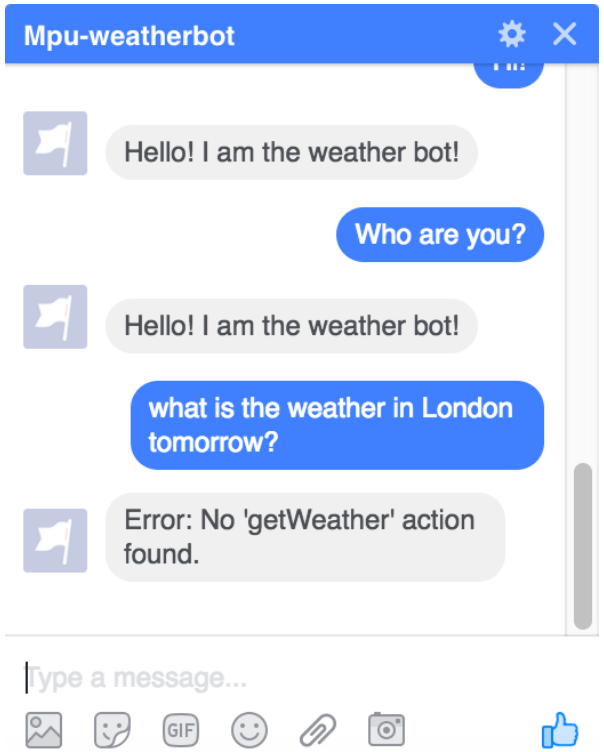


SC5

TRY IT!

The boilerplate returns an error because the *getWeather* action has not been implemented.

The earlier *debugContext* action is built-in the boilerplate.



SC5

IMPLEMENT LOGIC FOR FETCHING WEATHER

1. Copy snippet "*getWeather.js*" to *my-wit-actions.js*
2. Copy file *weather.js* to the *facebook-bot* directory

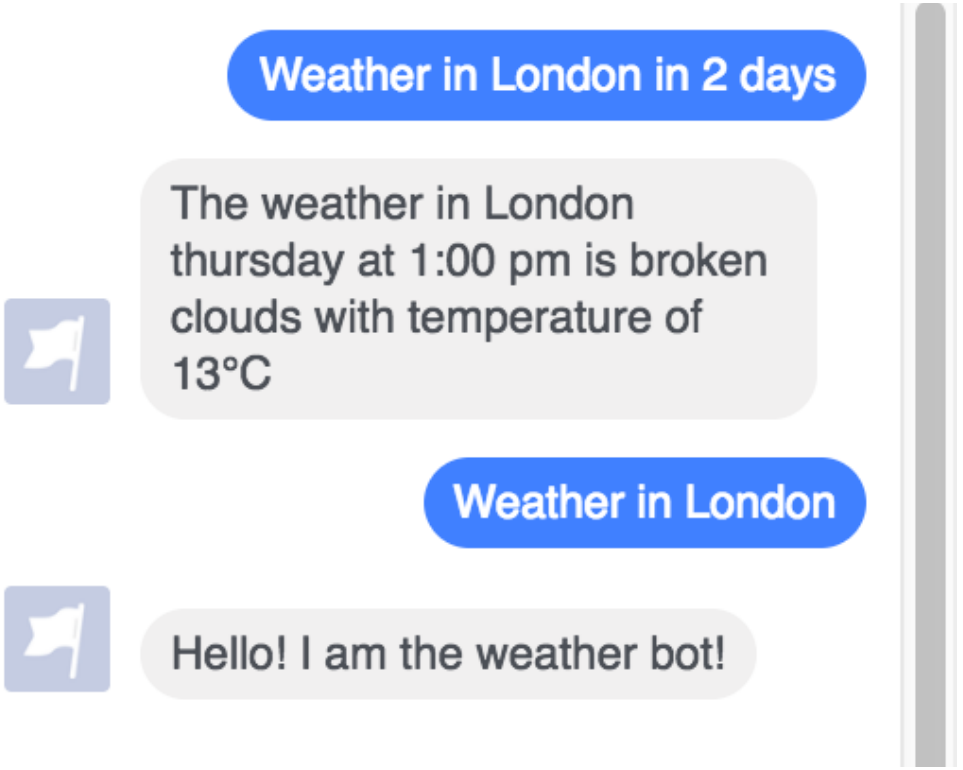
```
> sls deploy function -f facebookBot
```

```
const moment = require('moment');
const weather = require('./weather');

const actions = {
  getWeather: (data) => new Promise((resolve, reject) => {
    const context = data.context;
    const entities = data.entities;

    const missingLocation = entities.location === undefined;
    const location = entities.location ? entities.location[0].value : null;
    const datetime = entities.datetime ? entities.datetime[0].value : null;

    if (missingLocation) {
      const contextData = Object.assign({}, context, { missingLocation });
      resolve(contextData);
    } else if (datetime) {
      weather.forecastByLocationName(location, datetime)
        .then((weatherData) => {
          const contextData = Object.assign({}, context, weatherData);
          if (datetime) {
            Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
          }
          resolve(contextData);
        })
        .catch(reject);
    } else {
      weather.weatherByLocationName(location)
        .then((weatherData) => {
          const contextData = Object.assign({}, context, weatherData);
          if (datetime) {
            Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
          }
          resolve(contextData);
        })
        .catch(reject);
    }
  }),
```

TRY IT!

1. An inquiry with date and location information works!
2. But if we drop out the date, we do not get the right response.

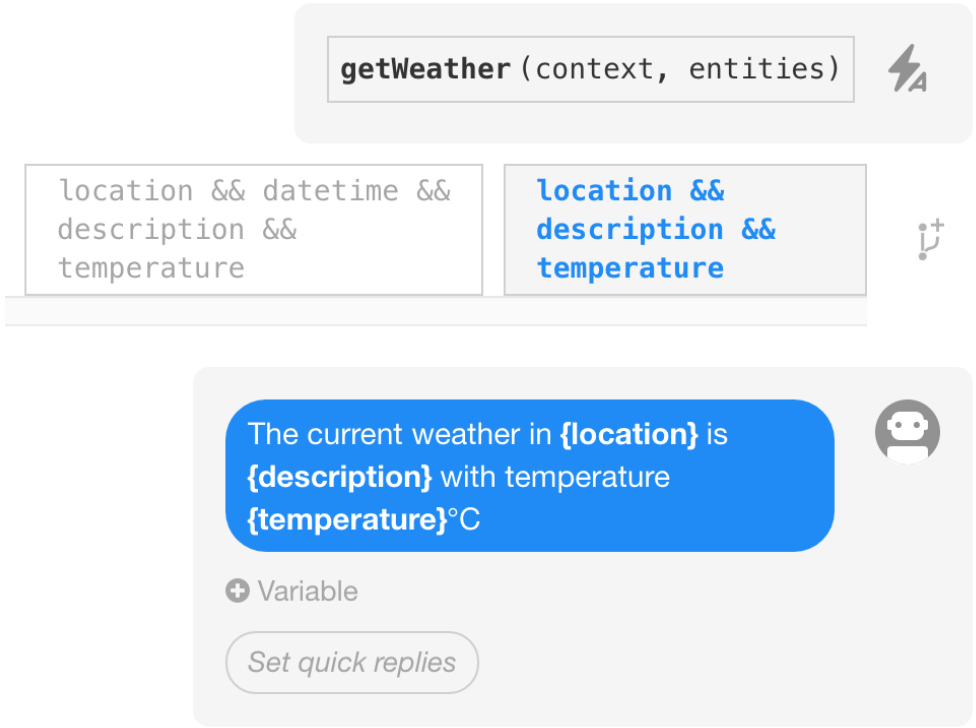
Let's fix that!



SC5

FORK THE STORY WITHOUT A DATE

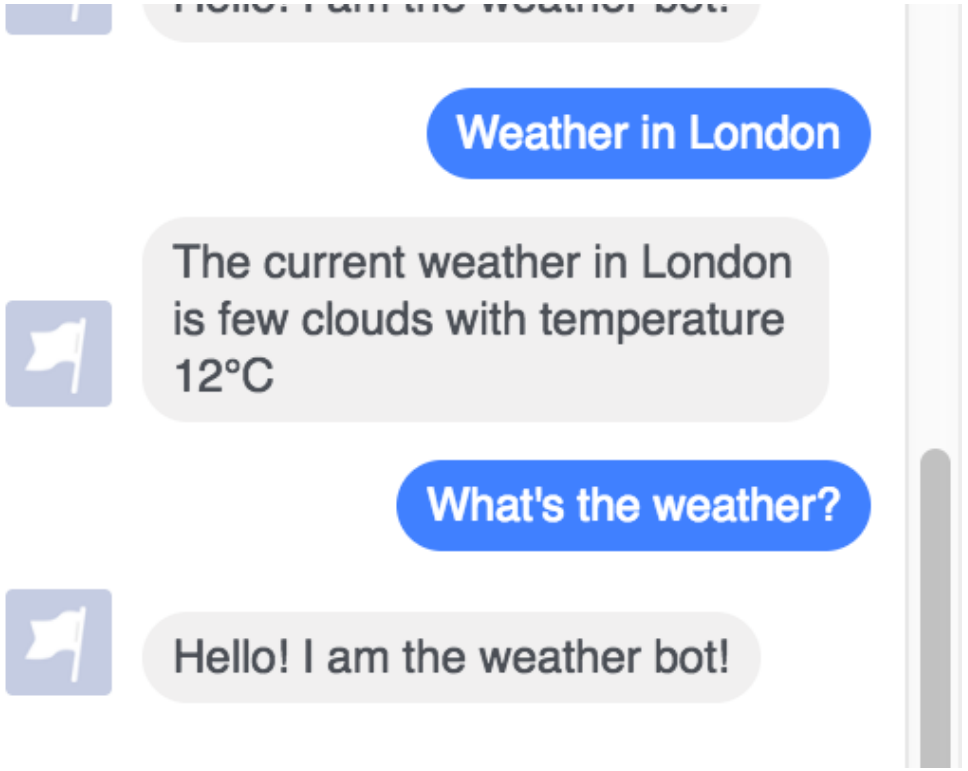
1. Click the fork icon next to the context keys
2. Create new context key *location && description && temperature* (without *datetime*)
3. Create new response
4. Save



SC5

TRY IT!

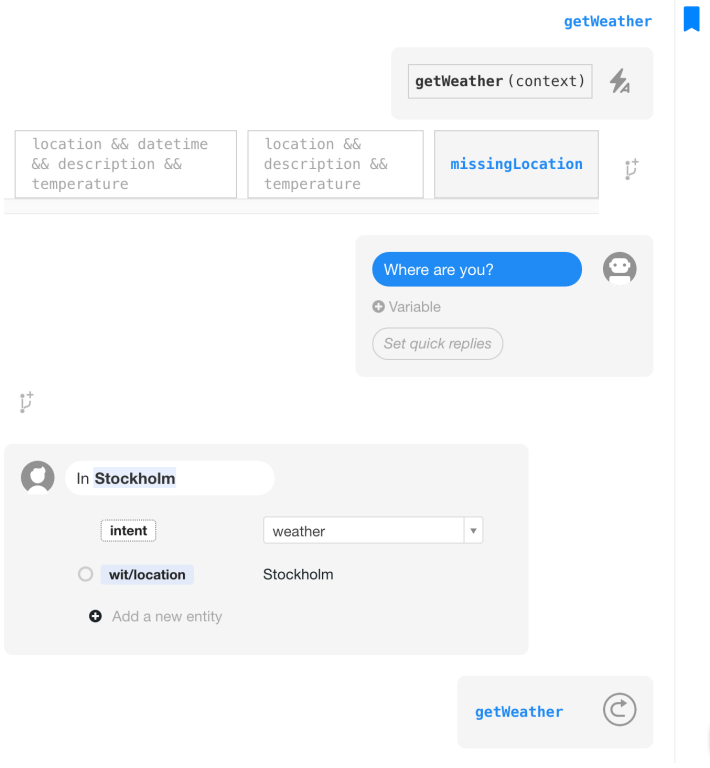
1. The current weather is now given if no date is provided
2. But if we do not provide a location, the bot should probably ask for it!



SC5

ASK ADDITIONAL QUESTIONS FROM THE USER

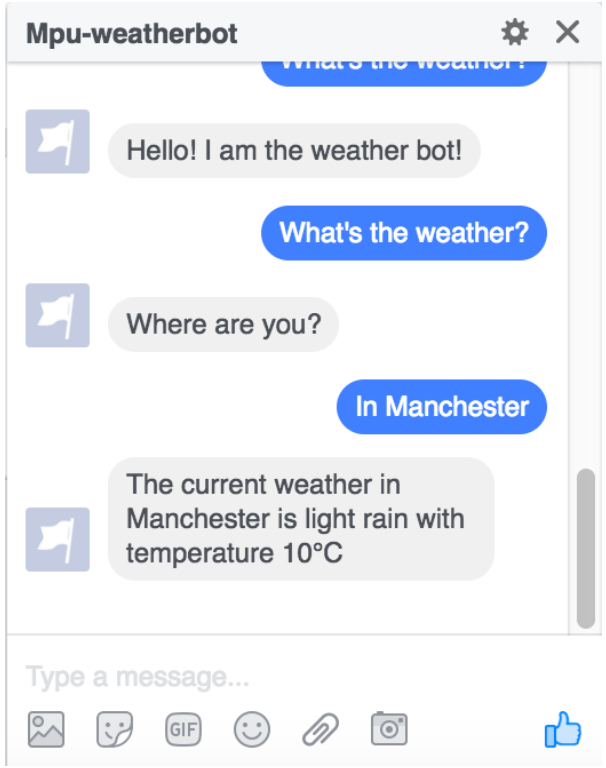
1. Click the bookmark icon next to the *getWeather* action and create bookmark *getWeather*
2. Create new fork with context key *missingLocation*
3. Add a bot response for asking the location
4. Add a user response and set *intent* + *wit/location* entities
5. Add Jump to *getWeather* bookmark created earlier



SC5

TRY IT!

IT WORKS!

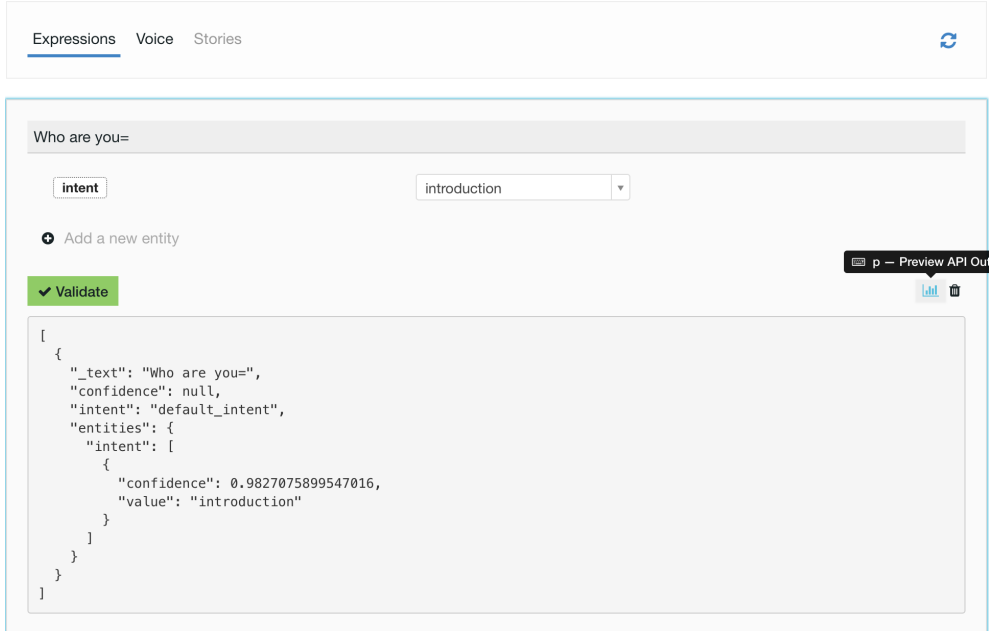


SC5

TRAINING THE BOT

WIT.AI INBOX

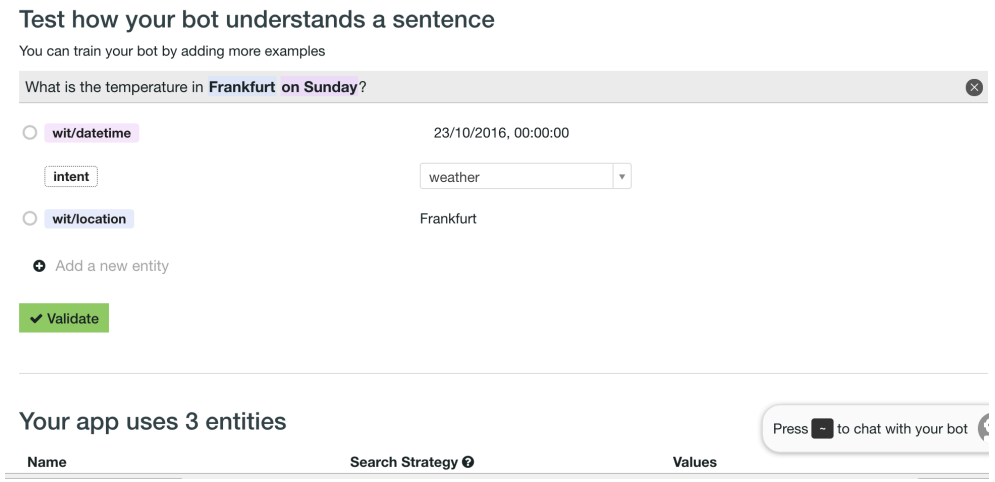
- Use Wit.ai inbox to train model based on incoming messages
- Validate expressions that have been entered by users
- Preview API output for expressions



SC5

WIT.AI INBOX

- Use Wit.ai understanding to train model
- Enter expressions, set entities and validate to improve accuracy



SC5

ADDITIONAL CHALLENGES

ADDITIONAL CHALLENGES (FOR EXAMPLE)

1. Create new story in a different language and hook your bot to it
2. Add SQS to the messageQueue to add robustness (remember to submit a PR to the boilerplate ☺)
3. Implement flow for ”What is the weather tomorrow?”, i.e. remember datetime when asking for location
4. Hook voice commands to bot
5. Or anything else / extend bot with something non-weather related

THANKS A LOT!