



# SERVERLESS MESSENGER BOT WORKSHOP

Mikael Puittinen, Chief Technology Officer

Eetu Tuomala, Principal Cloud Specialist

10.11.2016

# SC5 BRIEFLY



CLOUD  
SOLUTIONS



BUSINESS  
APPLICATIONS



DIGITAL  
DESIGN

**10**  
YEARS

**60+**  
CUSTOMERS

**200+**  
PROJECTS

**85**  
HACKERS  
DESIGNERS

**HEL  
JKL**

**~7**  
MEUR  
2016



A-lehdet



**Energia**

**Gasum**

**HAPPYORNOT**



s a n o m a



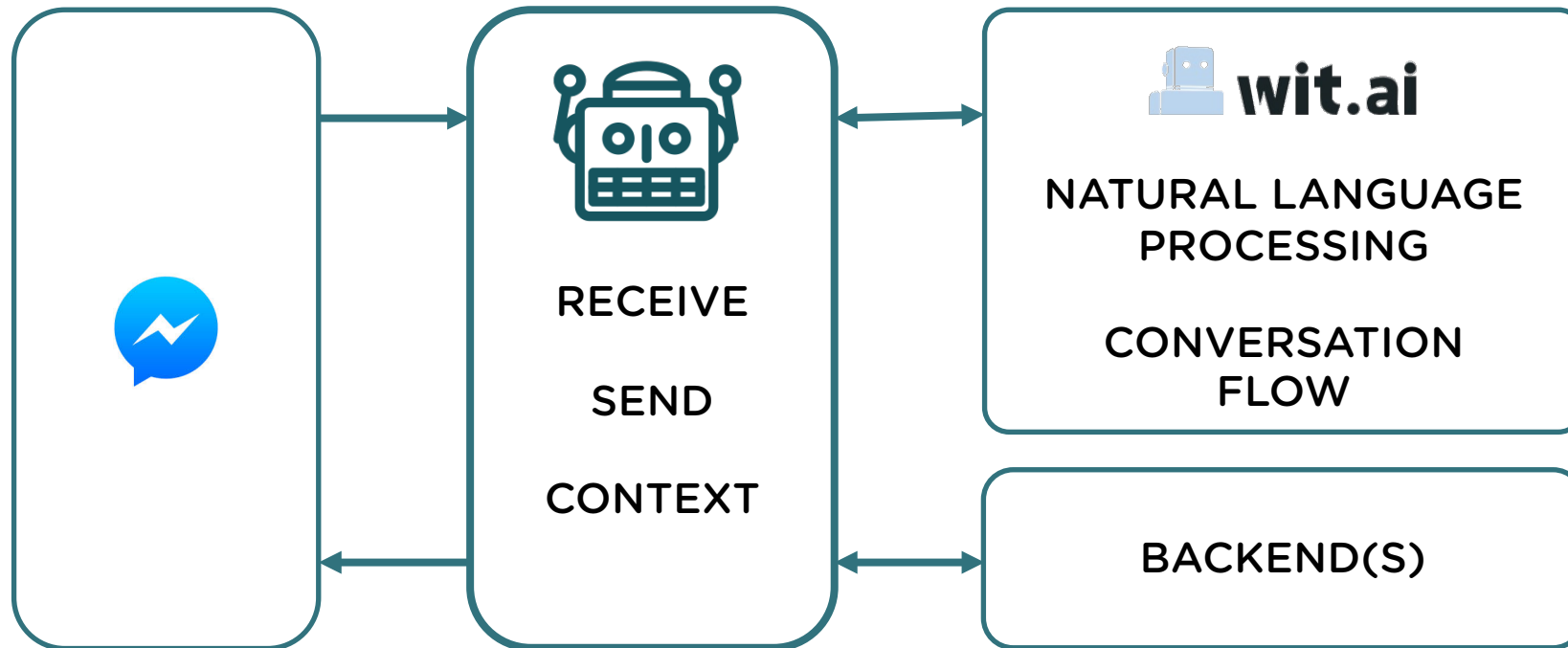
VISIT OUR WEB SITE FOR MORE INFO: [HTTPS://SC5.IO](https://sc5.io)

# INTRODUCTION

- Introduction to AWS & Serverless Framework

# INTRODUCTION TO MESSENGER BOT COMPONENTS

# MESSENGER BOT ARCHITECTURE



# MESSENGER PLATFORM

Messenger Platform Beta Launched April 2016

Over 1Bn monthly users

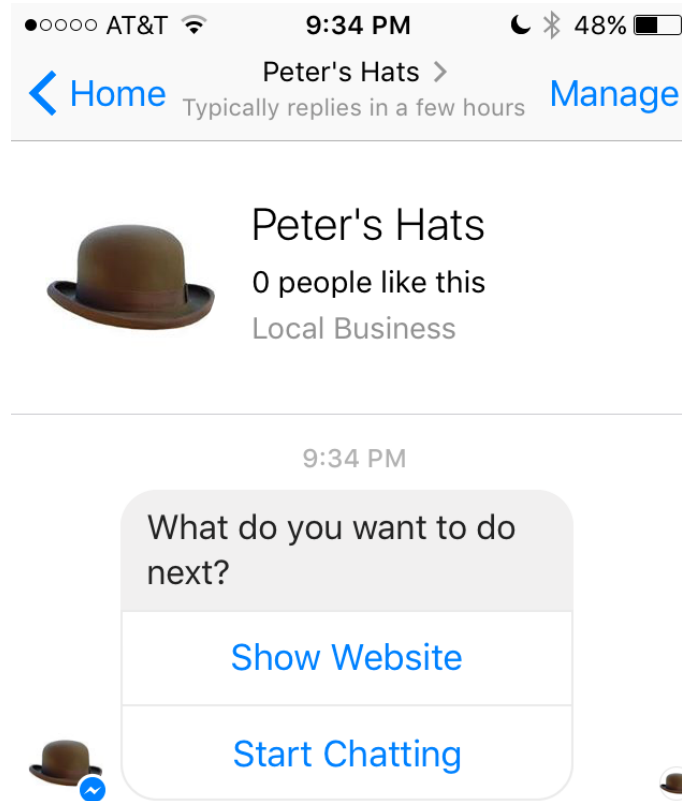
Over 33k bots

# RICH UI ELEMENTS: TEXT

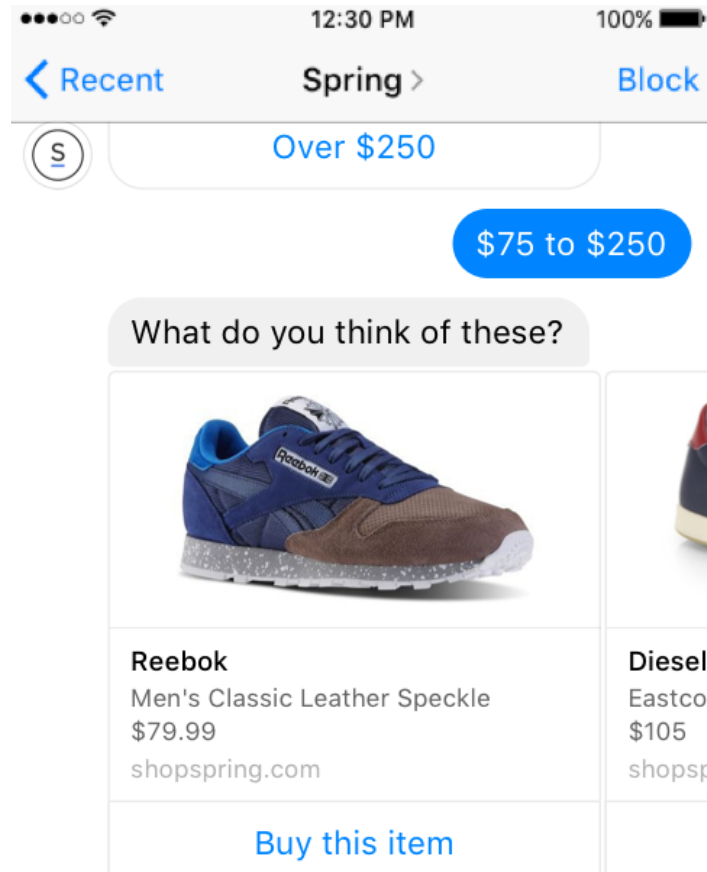




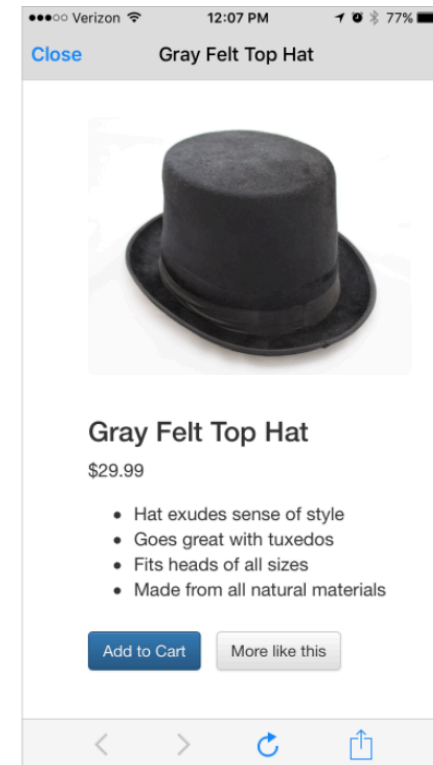
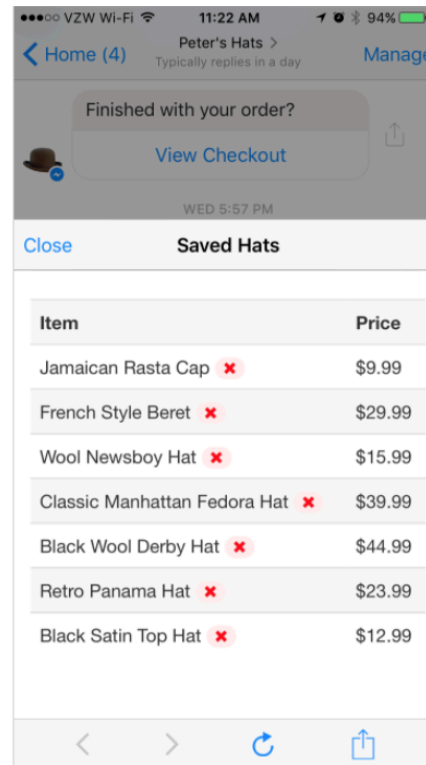
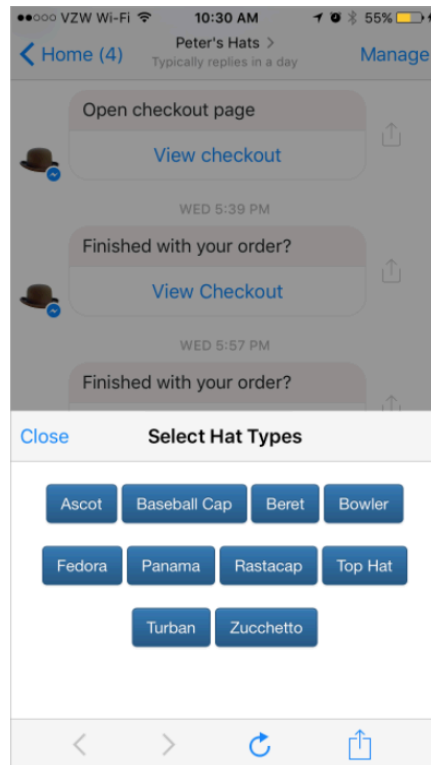
# RICH UI ELEMENTS: BUTTONS



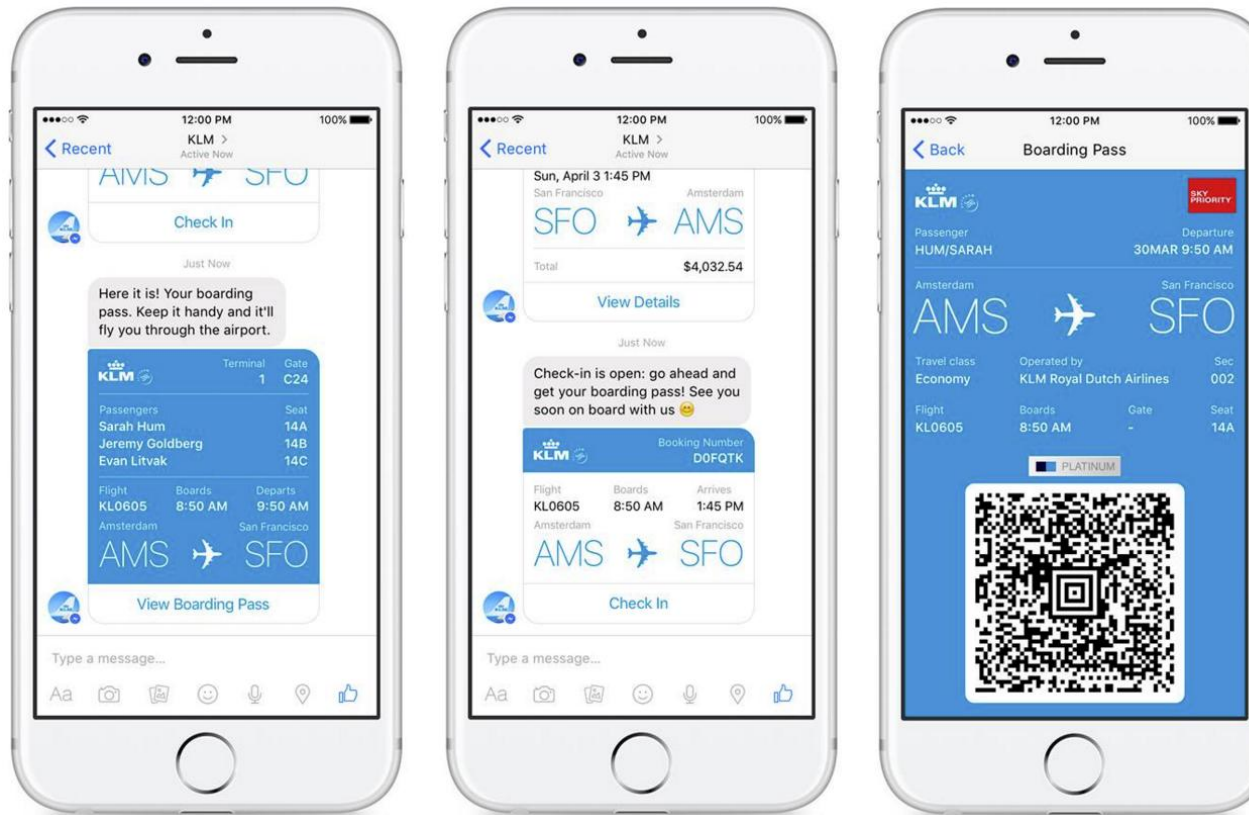
# RICH UI ELEMENTS: CAROUSEL



# RICH UI ELEMENTS: WEB VIEWS



# EXAMPLE: KLM MESSENGER



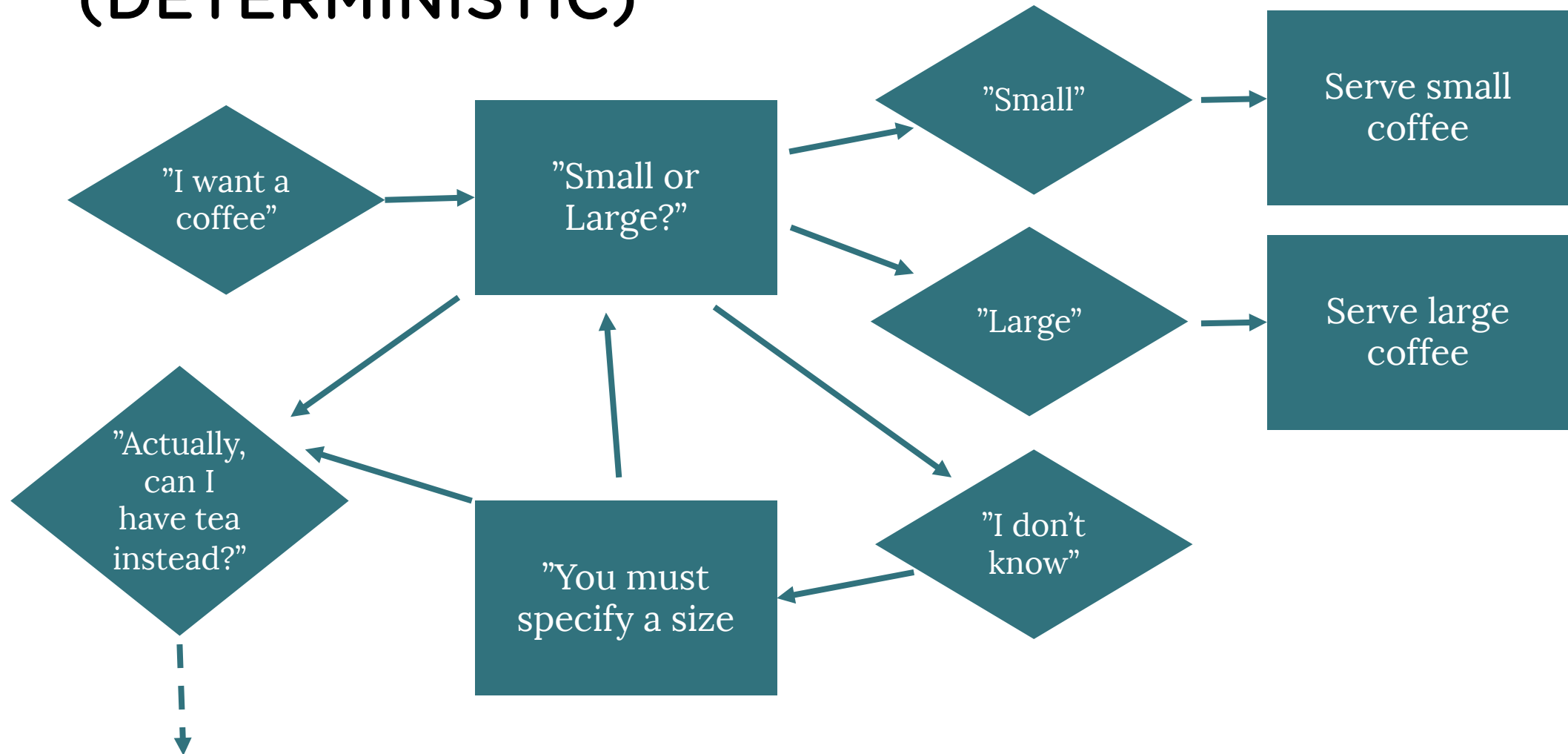
# MESSENGER BOT SETUP QUICK GUIDE

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

Messenger Platform API documentation available at:  
<https://developers.facebook.com/docs/messenger-platform/>

WIT.AI

# CONVERSATION WORKFLOW (DETERMINISTIC)



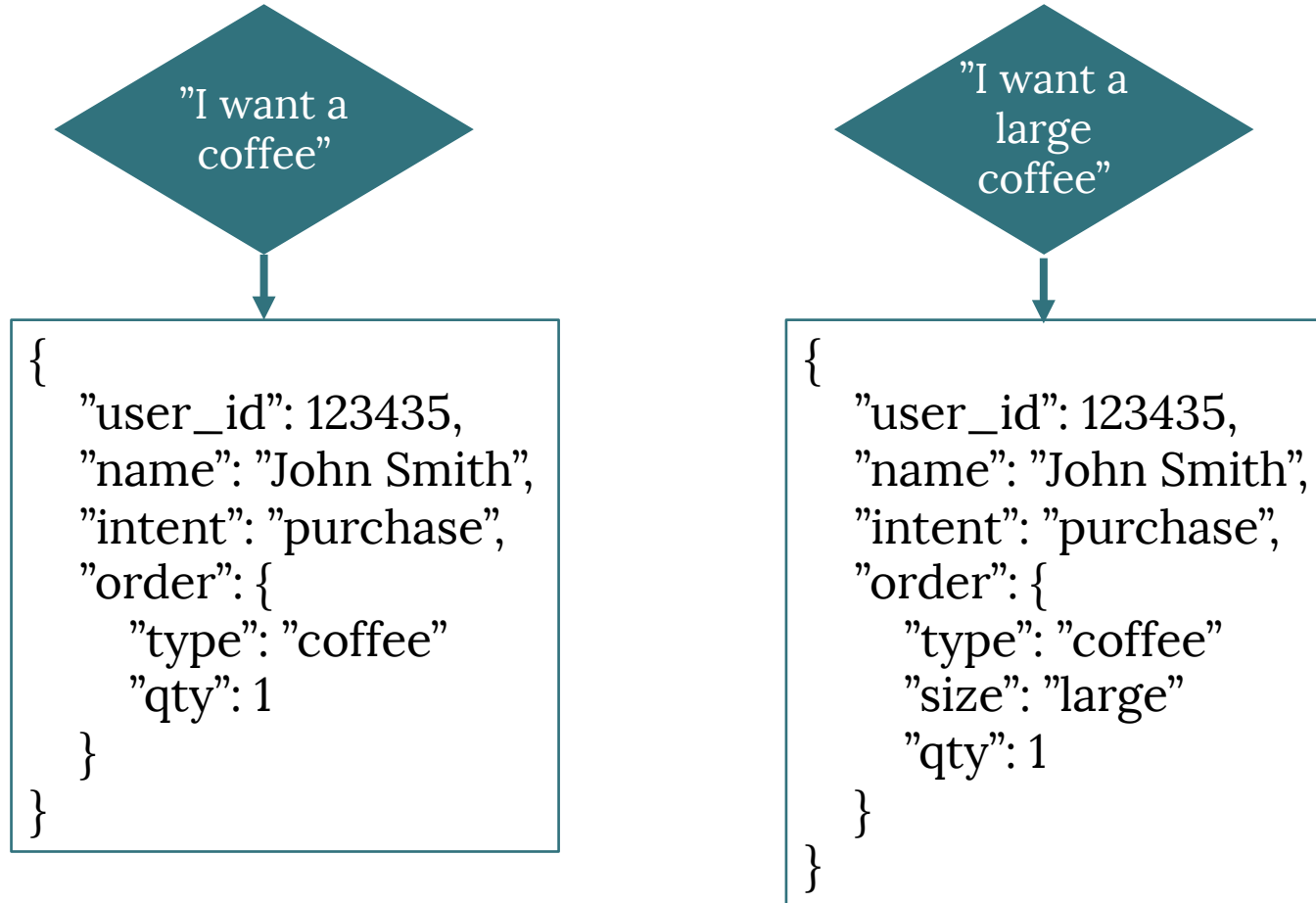


# WIT.AI APPROACH

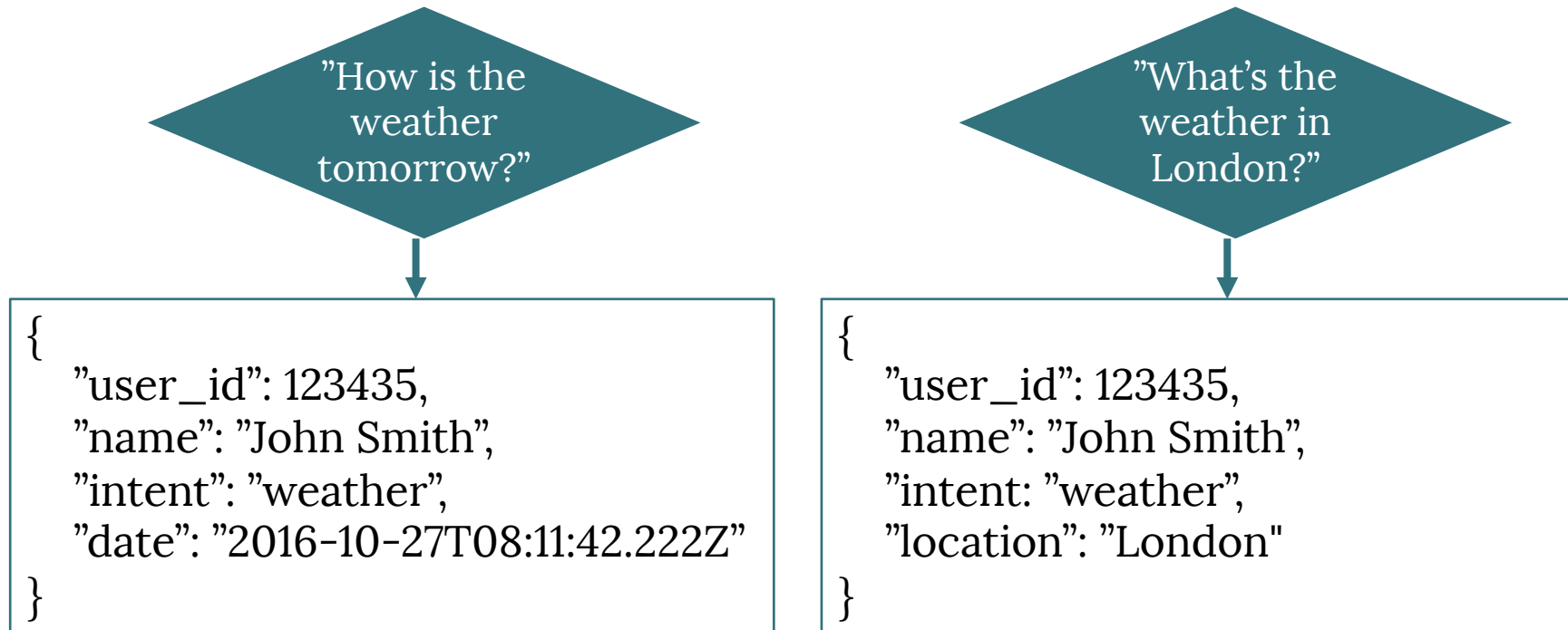
- Stories to define "rules"/"heuristics" for the conversation
- Context to define current state

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

# CONTEXTS



# NATURAL LANGUAGE PROCESSING



# 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>

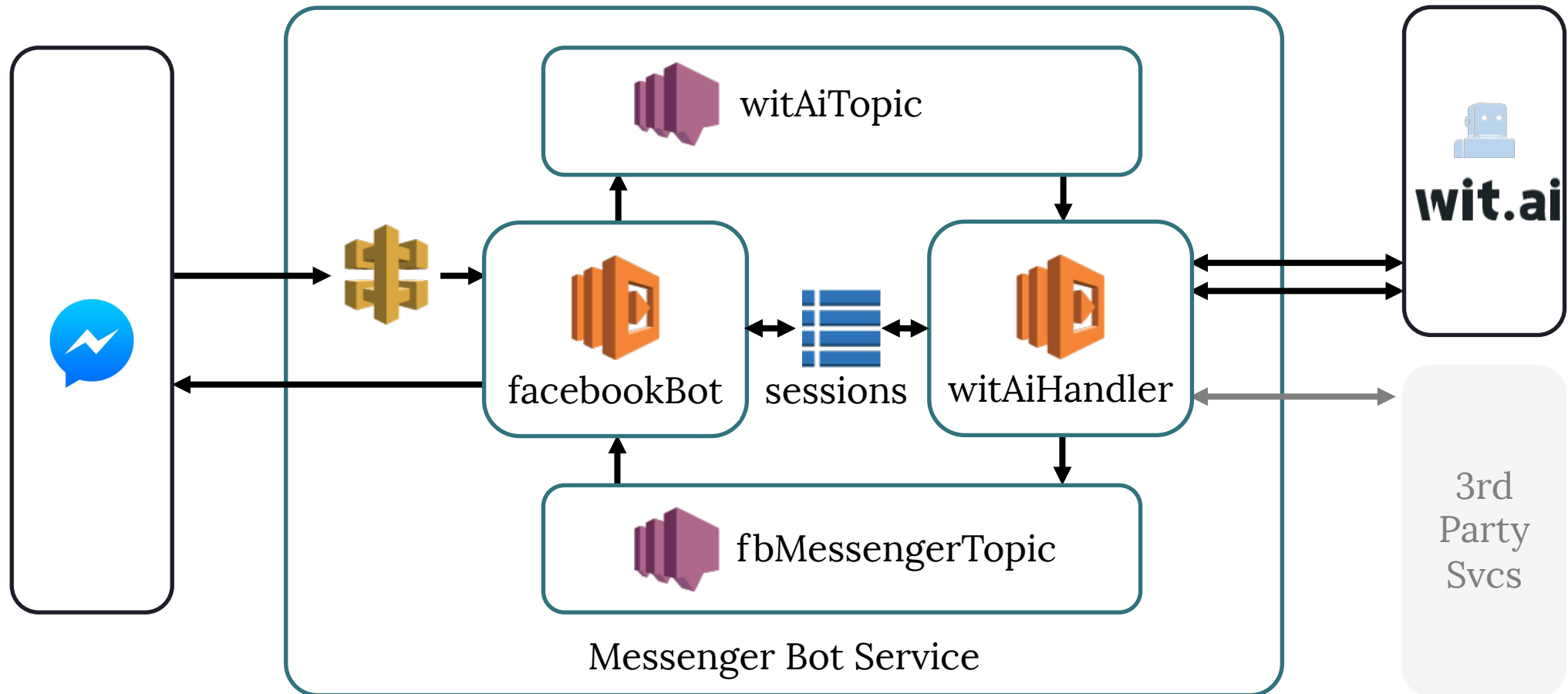


# OUR BOILERPLATE

serverless-messenger-boilerplate and its documentation  
available at

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

# BOILERPLATE ARCHITECTURE



# BOILERPLATE STRUCTURE

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

# .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 install node modules and copy *example.env* to *.env*

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

Generate a random verification token to *VERIFY\_TOKEN* in *.env*

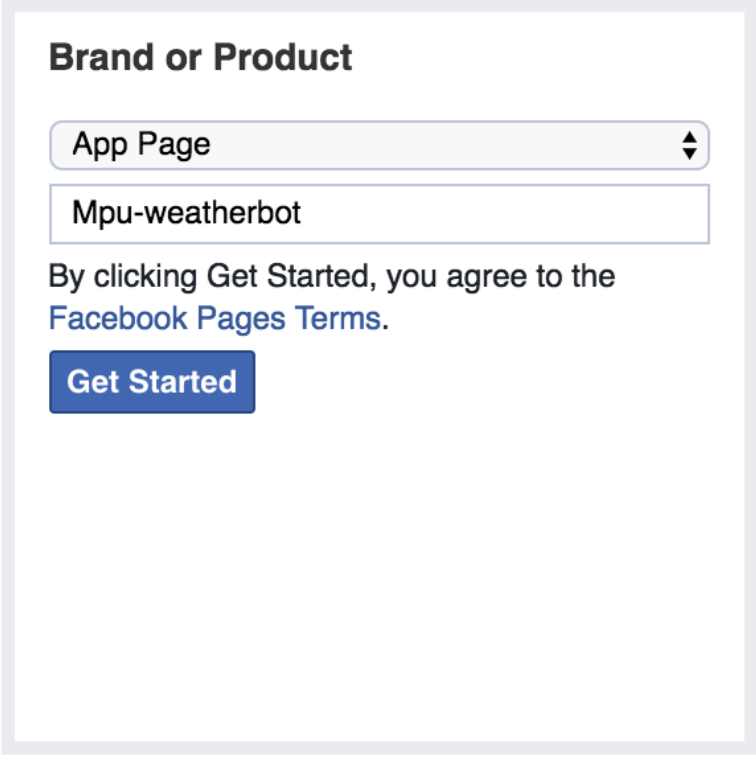
Deploy the service and memorize the URL

```
> sls deploy
```

# SETUP FACEBOOK PAGE AND APPLICATION

# CREATE A FACEBOOK PAGE

- Go to <http://facebook.com/pages/create>
- Create new page
- Skip all following steps

A screenshot of the Facebook page creation interface. It features a section titled "Brand or Product" with a dropdown menu showing "App Page" and a text input field containing "Mpu-weatherbot". Below these fields, there is a line of text stating "By clicking Get Started, you agree to the Facebook Pages Terms." followed by a blue "Get Started" button.

**Brand or Product**

App Page

Mpu-weatherbot

By clicking Get Started, you agree to the [Facebook Pages Terms](#).

**Get Started**

# CREATE FACEBOOK APPLICATION

- Go to <https://developers.facebook.com/quickstarts/?platform=web>
- Click "Skip and Create App ID"
- Fill in info
- Create app ID

## Create a New App ID

Get started integrating Facebook into your app or website

Display Name

mpu-weatherbot

Contact Email

mikael.puittinen@sc5.io

Category

Apps for Pages ▾

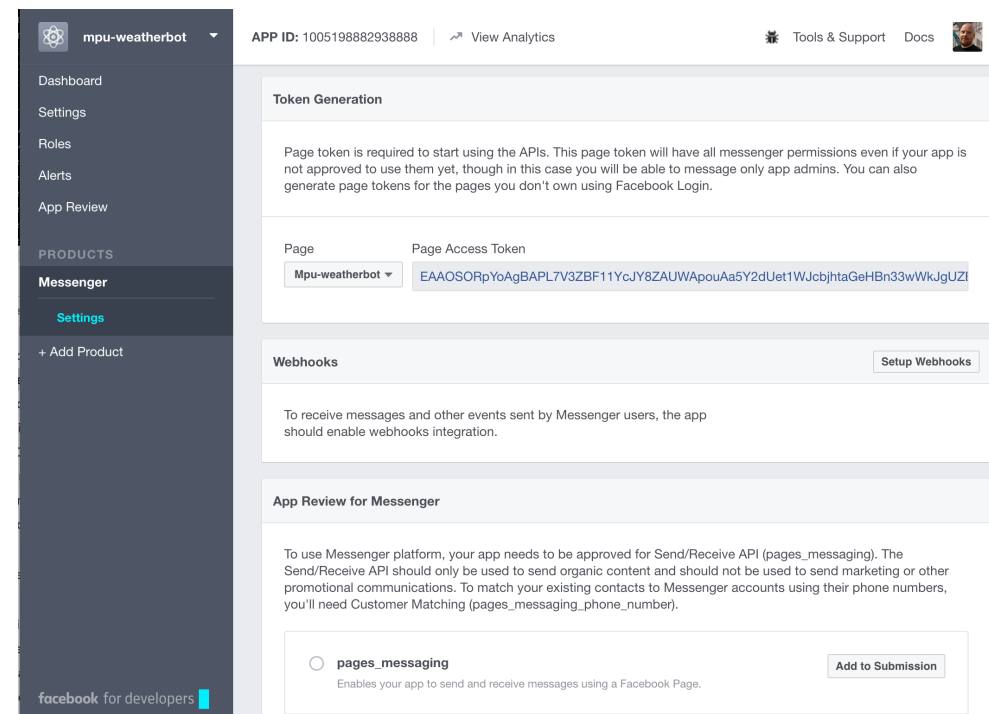
By proceeding, you agree to the [Facebook Platform Policies](#)

Cancel

Create App ID

# SETUP MESSENGER TOKEN

- "Get Started" for Messenger
- Generate a token for your page
- Copy token to `.env`  
(`FACEBOOK_BOT_PAGE_ACCESS_TOKEN`)



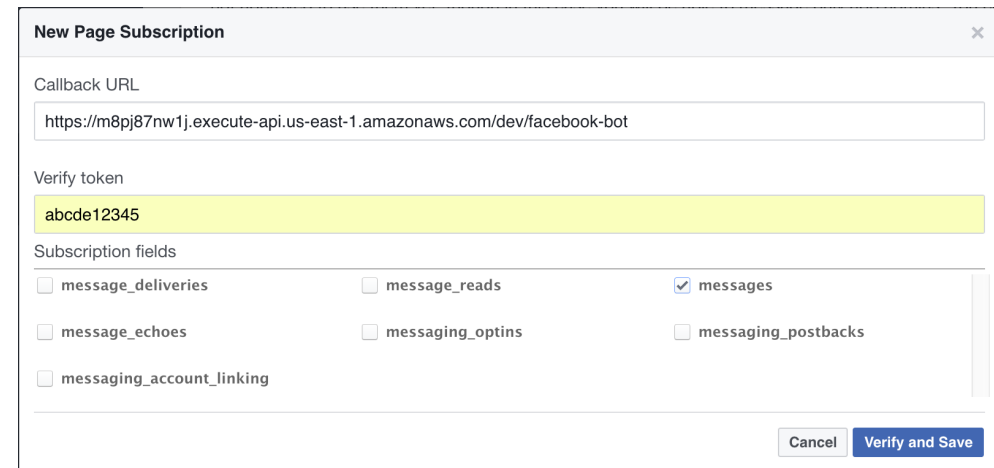


# SETUP MESSENGER WEBHOOK

- Initiate "Setup Webhooks"
- Enter the endpoint URL that you got during deployment
- Enter your verify token from *.env* (FACEBOOK\_VERIFY)
- Deploy

```
> sls deploy
```

- Verify and Save



New Page Subscription

Callback URL

`https://m8pj87nw1j.execute-api.us-east-1.amazonaws.com/dev/facebook-bot`

Verify token

`abcde12345`

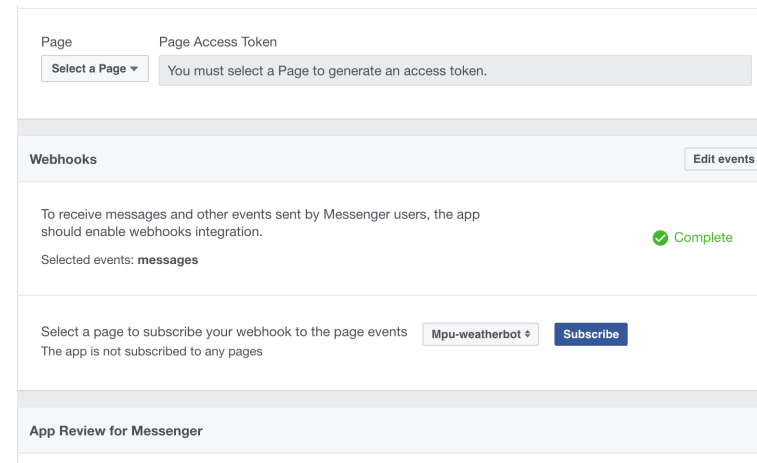
Subscription fields

<input type="checkbox"/> message_deliveries	<input type="checkbox"/> message_reads	<input checked="" type="checkbox"/> messages
<input type="checkbox"/> message_echoes	<input type="checkbox"/> messaging_optins	<input type="checkbox"/> messaging_postbacks
<input type="checkbox"/> messaging_account_linking		

Cancel Verify and Save

# SUBSCRIBE THE WEBHOOK TO THE FACEBOOK PAGE

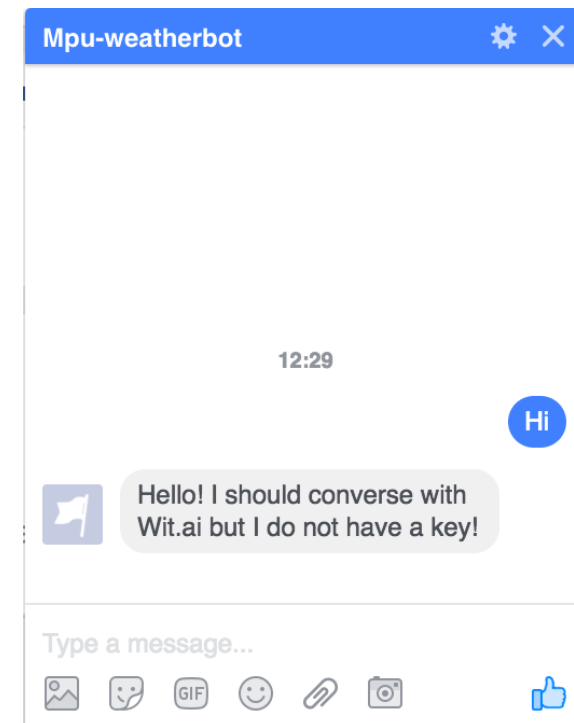
1. Under "Webhooks", select your page and subscribe



The screenshot shows the Facebook Developer console interface for configuring webhooks. At the top, there are two tabs: "Page" and "Page Access Token". Under the "Page" tab, there is a dropdown menu labeled "Select a Page" and a message: "You must select a Page to generate an access token." Below this, the "Webhooks" section is active, showing a status of "Complete" with a green checkmark. It indicates that the app should enable webhooks integration and that the selected events are "messages". At the bottom of the "Webhooks" section, there is a prompt to "Select a page to subscribe your webhook to the page events" with a dropdown menu showing "Mpu-weatherbot" and a "Subscribe" button. Below the "Webhooks" section, there is a section for "App Review for Messenger".

# TRY IT

- Send message to your Facebook page



# 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

# REGISTER TO OPENWEATHERMAP

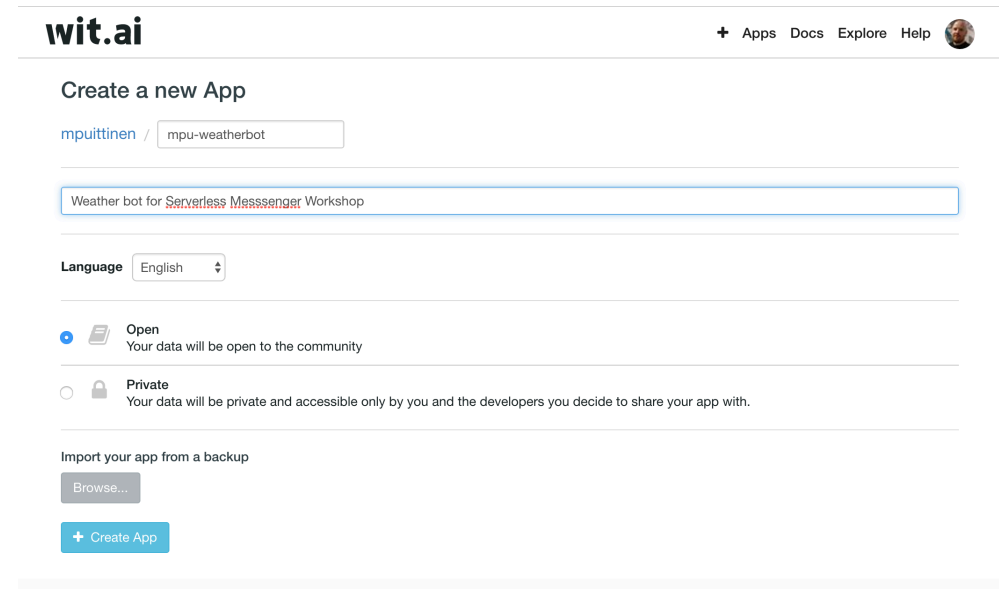
1. Go to <https://openweathermap.org/appid>
2. Sign up
3. Copy API key and set it to WEATHER\_API\_TOKEN in .env

Key	Name
83ffbd8ffb5bd3cded626ffddfbd67f6	Default

SETUP WIT.AI

# REGISTER + CREATE WIT.AI APP

- Go to <https://wit.ai>
- Register (if not already done)
- Create a new App



The screenshot shows the 'Create a new App' page on the wit.ai website. The header includes the 'wit.ai' logo and navigation links for '+ Apps', 'Docs', 'Explore', 'Help', and a user profile icon. The main form area is titled 'Create a new App' and contains the following elements:

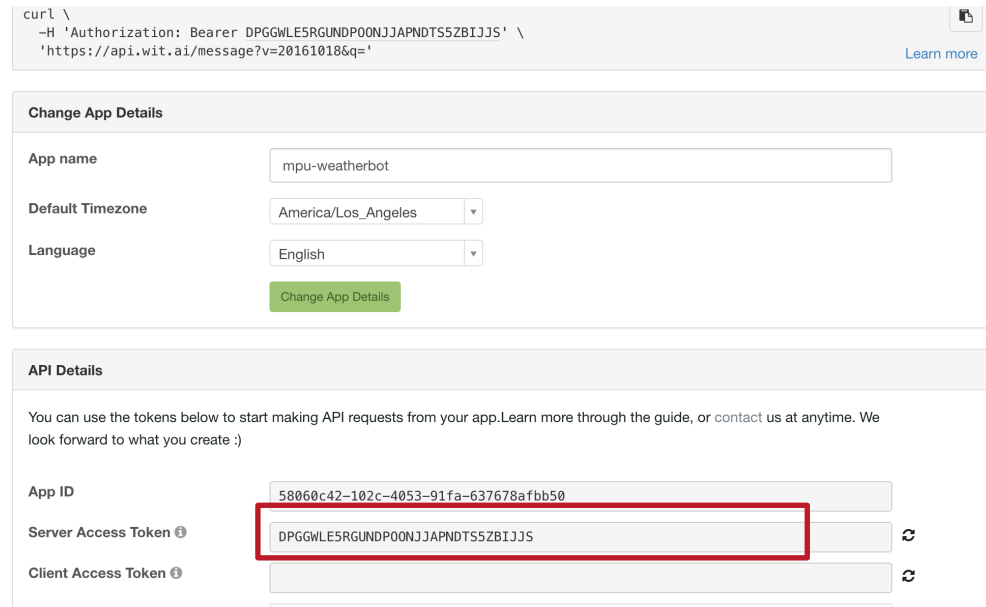
- A username field with 'mpuittinen' and a slash, followed by an app name input field containing 'mpu-weatherbot'.
- A description input field with the text 'Weather bot for ~~Serverless~~ Messenger Workshop'.
- A 'Language' dropdown menu set to 'English'.
- Two radio button options for data privacy:
  - Open** (selected): 'Your data will be open to the community'.
  - Private**: 'Your data will be private and accessible only by you and the developers you decide to share your app with.'
- An 'Import your app from a backup' section with a 'Browse...' button.
- A blue '+ Create App' button at the bottom.



# CONNECT THE ENDPOINT WITH WIT.AI

- Go to Settings
- Copy the Server Access Token ID to .env (WIT\_AI\_TOKEN)
- Deploy your service

```
> sls deploy
```



The screenshot displays the Wit.ai developer console interface. At the top, a terminal window shows a curl command for sending a message to the Wit.ai API. Below this, the 'Change App Details' section contains input fields for 'App name' (mpu-weatherbot), 'Default Timezone' (America/Los\_Angeles), and 'Language' (English), along with a 'Change App Details' button. The 'API Details' section provides instructions on using the tokens and lists three token types: 'App ID' (58060c42-102c-4053-91fa-637678afbb50), 'Server Access Token' (DPGGWLE5RGUNDPO0NJJAPNDTS5ZBIJJS, highlighted with a red box), and 'Client Access Token'.

```
curl \
-H 'Authorization: Bearer DPGGWLE5RGUNDPO0NJJAPNDTS5ZBIJJS' \
'https://api.wit.ai/message?v=20161018&q='
```

[Learn more](#)

### Change App Details

App name: mpu-weatherbot

Default Timezone: America/Los\_Angeles

Language: English

[Change App Details](#)

### API Details

You can use the tokens below to start making API requests from your app. Learn more through the guide, or contact us at anytime. We look forward to what you create :)

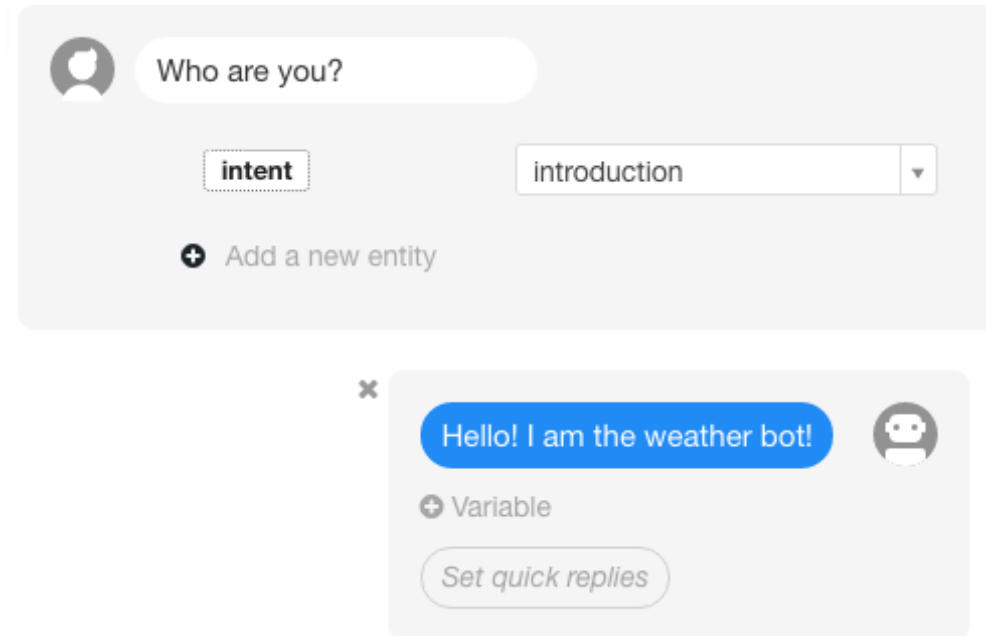
App ID: 58060c42-102c-4053-91fa-637678afbb50

Server Access Token ⓘ: DPGGWLE5RGUNDPO0NJJAPNDTS5ZBIJJS

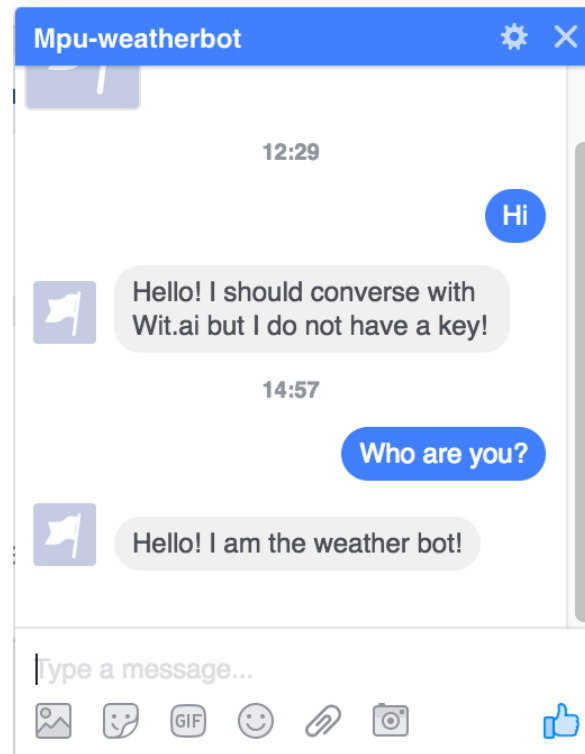
Client Access Token ⓘ:

# LET THE BOT INTRODUCE ITSELF

- Create a new story
- Add user input and intent
- Add a response with "Bot Sends"
- Remember to save the story



# TRY IT!

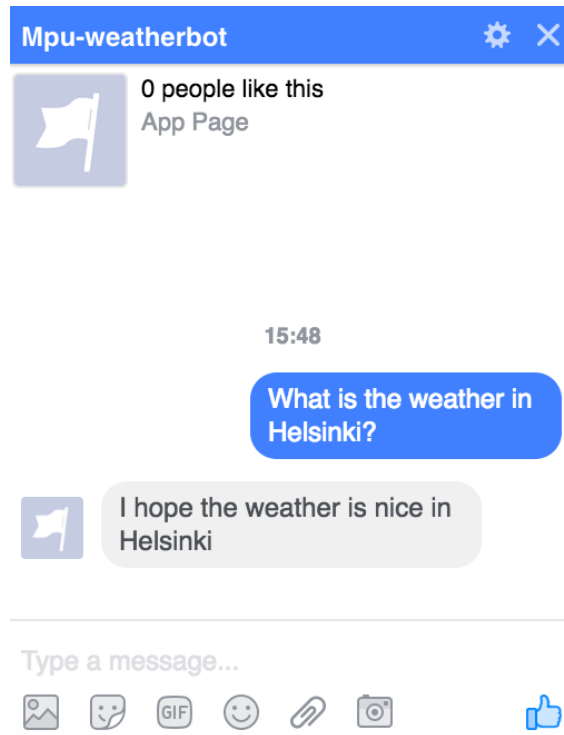


# WEATHER STORY (WITH FIXED RESPONSE)

- Add a new story
- Type in your user input
- Add entity *intent* with value *weather*
- Select the location and add new entity *wit/location*
- Select the date and add new entity *wit/datetime*
- Add action *debugContext* with "Bot Executes"
- Set context key *location*
- Add a response using "Bot sends"
- Save

The screenshot displays the Wit.ai interface for configuring a story. At the top, a user input box contains the text "What is the weather in Helsinki tomorrow?". Below this, two entities are defined: "wit/datetime" with the value "19/10/2016, 00:00:00" and "wit/location" with the value "Helsinki". A button labeled "Add a new entity" is visible. Below the entities, the "debugContext" action is configured with the context key "location". The response is set to "I hope the weather is nice in {location}". The interface also includes a "Set quick replies" button and a "Variable" section.

# 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
```

# ADD LOGIC TO THE STORY

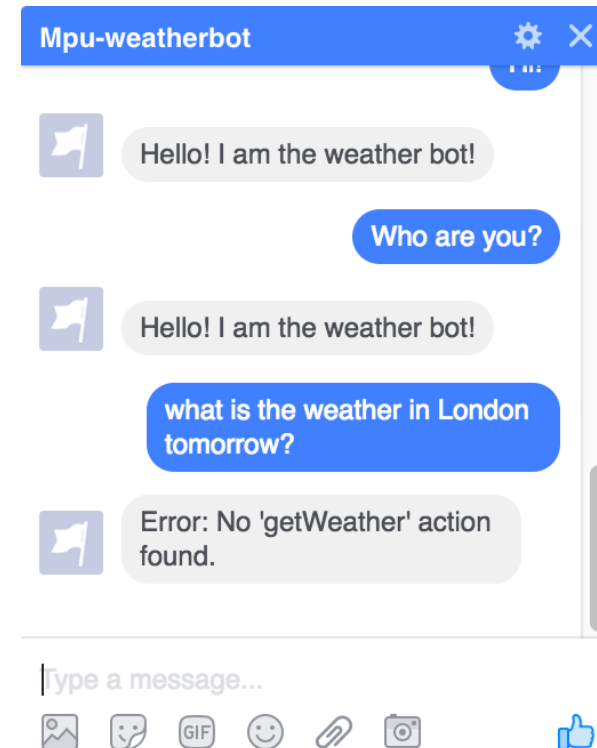
- Remove action *debugContext* and context key *location*
- Add action *getWeather*
- Add context key *location* && *datetime* && *description*
- Add response

The screenshot displays a chatbot story editor interface. At the top, a user input bubble contains the text "What is the weather in Helsinki tomorrow?". Below this, the editor shows two entities: "wit/datetime" with the value "19/10/2016, 00:00:00" and "wit/location" with the value "Helsinki". An "intent" dropdown is set to "weather". A button labeled "Add a new entity" is visible. Below the entities, the action "getWeather (context, entities)" is selected, indicated by a lightning bolt icon. Underneath the action, the context keys "location && datetime && description && temperature" are listed. At the bottom, a response bubble is shown with the text "The weather in {location} {datetime} is {description} with temperature of {temperature}°C". Below the response, there is a "Variable" section with a "Set quick replies" button.

# TRY IT!

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

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



# IMPLEMENT LOGIC FOR FETCHING WEATHER

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

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

```
1  const moment = require('moment');
2  const weather = require('./weather');
3
4  const actions = {
5    getWeather: (data) => new Promise((resolve, reject) => {
6      const context = data.context;
7      const entities = data.entities;
8
9      const missingLocation = entities.location === undefined;
10     const location = entities.location ? entities.location[0].value : null;
11     const datetime = entities.datetime ? entities.datetime[0].value : null;
12
13     if (missingLocation) {
14       const contextData = Object.assign({}, context, { missingLocation });
15       resolve(contextData);
16     } else if (datetime) {
17       weather.forecastByLocationName(location, datetime)
18         .then((weatherData) => {
19           const contextData = Object.assign({}, context, weatherData);
20           if (datetime) {
21             Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
22           }
23           resolve(contextData);
24         })
25         .catch(reject);
26     } else {
27       weather.weatherByLocationName(location)
28         .then((weatherData) => {
29           const contextData = Object.assign({}, context, weatherData);
30           if (datetime) {
31             Object.assign(contextData, { datetime: moment(datetime).calendar().toLowerCase() });
32           }
33           resolve(contextData);
34         })
35         .catch(reject);
36     }
37   });
38 }
```



# TRY IT!

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

Let's fix that!



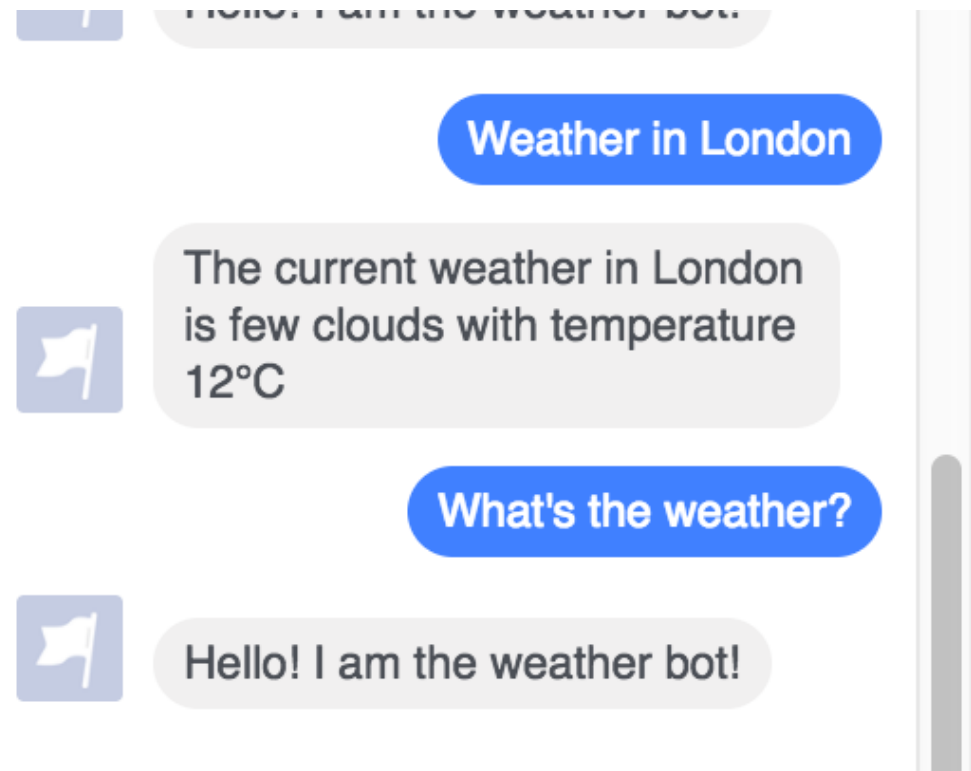
# FORK THE STORY WITHOUT A DATE

- Click the fork icon next to the context keys
- Create new context key *location && description && temperature (without datetime)*
- Create new response
- Save

The screenshot shows a story editor interface. At the top, there is a box containing the text `getWeather (context, entities)` and a lightning bolt icon. Below this, there are two boxes side-by-side. The left box contains the text `location && datetime && description && temperature`. The right box contains the text `location && description && temperature` in blue, with a fork icon to its right. Below these boxes, there is a large blue box containing the text `The current weather in {location} is {description} with temperature {temperature}°C`. To the right of this box is a robot icon. Below the blue box, there is a small box containing the text `+ Variable`. At the bottom, there is a button labeled `Set quick replies`.

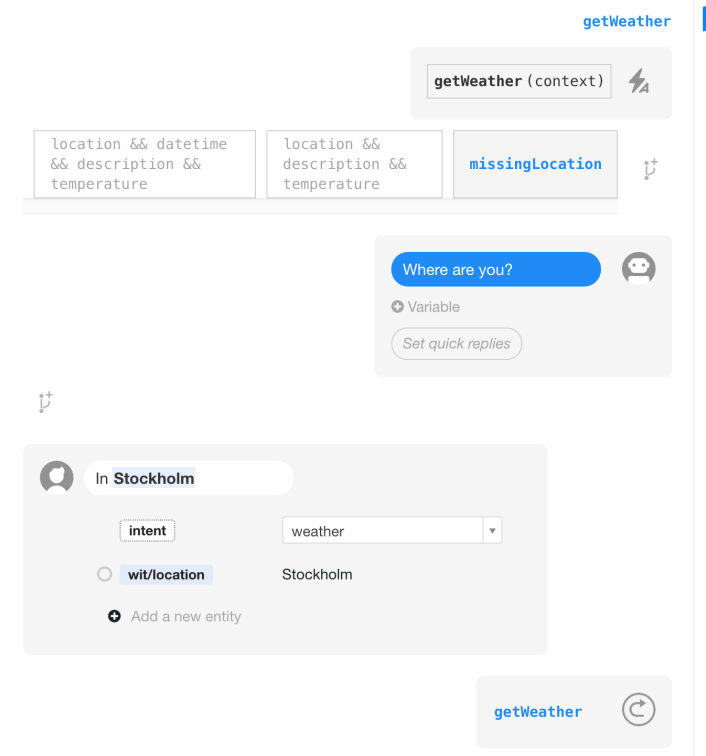
# TRY IT!

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



# ASK ADDITIONAL QUESTIONS FROM THE USER

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



TRY IT!

IT WORKS!



# 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

The screenshot shows the Wit.ai Expressions interface. At the top, there are tabs for 'Expressions', 'Voice', and 'Stories'. The 'Expressions' tab is selected. Below the tabs, there is a text input field containing 'Who are you=' and a dropdown menu showing 'introduction'. Below the input field, there is a button labeled 'intent' and a button labeled 'Add a new entity'. Below these buttons, there is a green button labeled 'Validate'. To the right of the 'Validate' button, there is a button labeled 'Preview API Output'. Below the 'Validate' button, there is a large text area displaying the JSON output of the API call:

```
[
  {
    "_text": "Who are you=",
    "confidence": null,
    "intent": "default_intent",
    "entities": {
      "intent": [
        {
          "confidence": 0.9827075899547016,
          "value": "introduction"
        }
      ]
    }
  }
]
```

# WIT.AI INBOX

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

## Test how your bot understands a sentence

You can train your bot by adding more examples


What is the temperature in **Frankfurt on Sunday**? ✕

☐ **wit/datetime** 23/10/2016, 00:00:00

☐ **wit/location** Frankfurt

## Your app uses 3 entities

Name	Search Strategy ⓘ	Values
------	-------------------	--------

Press  to chat with your bot



SC5

THANKS A LOT!

*[mikael.puittinen@sc5.io](mailto:mikael.puittinen@sc5.io)*