SC5

SERVERLESS CHATBOT WORKSHOP FULL STACK FEST 2017

Eetu Tuomala, *Cloud Application Architect*

Alexander Kozlov, *Senior Developer*

Alexandr Lukyanov, *Full Stack Developer*

September 6th, 2017

THIS IS SC5

WE CAN - WE CARE - WE SHARE

SC5

HEADQUARTERS IN HELSINKI

SC5

What you should know about us

DOMAINS + NUMBERS

BUSINESS APPLICATIONS

CLOUD SOLUTIONS

DIGITAL DESIGN

MACHINE LEARNING

10+ YEARS	100+ CUSTOMERS	400+ PROJECTS
85 HACKERS DESIGNERS	HEL JKL	7 MEUR 2016 (FC)

WHAT IS SERVERLESS

SC5

“ MANAGED SERVICES WHERE THE SERVICE PROVIDES SCALING / PATCHING / REPLACING / ETC AND YOU PROVIDE CODE / CONFIG. “

Ryan Scott Brown
Ansible / Red Hat

SERVERLESS COMPONENTS

- Function
- Event source
- Service that runs the code
- Networking environment

SERVERLESS PROS

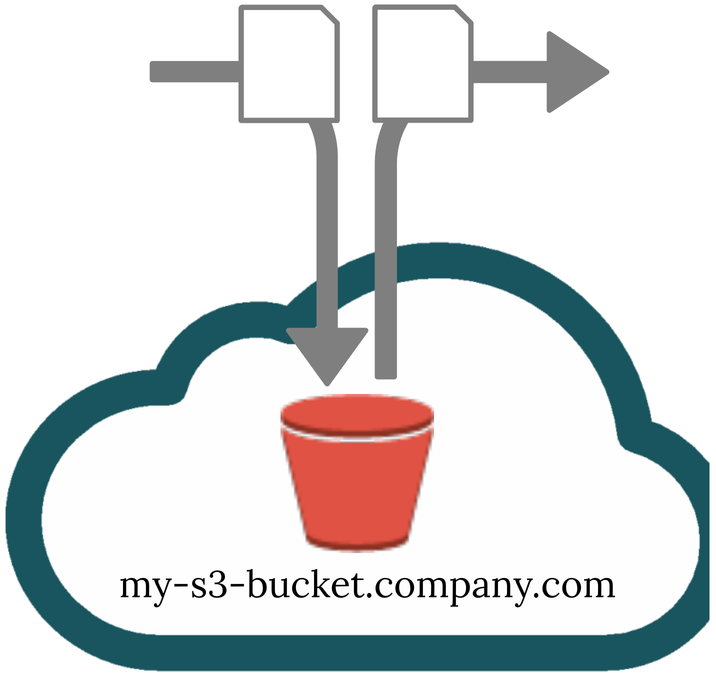
- Allows developers to focus on code
- Reduced time to market and quicker software release
- Lower operational and development costs
- A smaller cost to scale
- Requires less system administration

SERVERLESS CONS

- Not efficient for long-running applications
- Vendor lock-in
- Introduces additional overhead for function calls
- Cold start
- Local testing

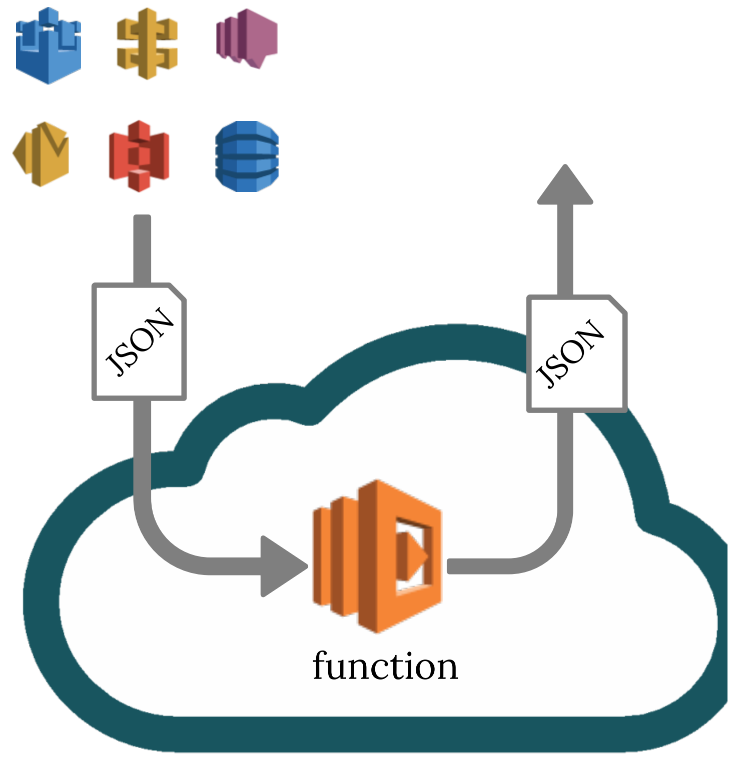
AMAZON SIMPLE STORAGE SERVICE (S3)

- (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



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)



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-hand side
5. Role: "Create new role from template"
6. Select policy "Simple Microservice"
7. Once created, test with some JSON input
8. See the logs in CloudWatch

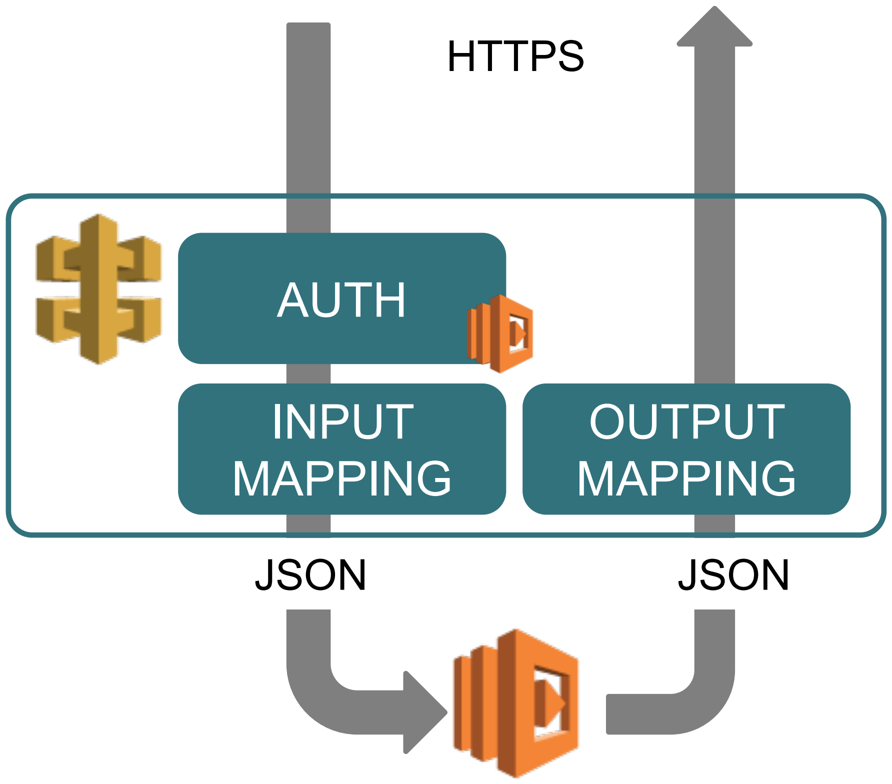
```
'use strict';

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

SC5

AMAZON 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

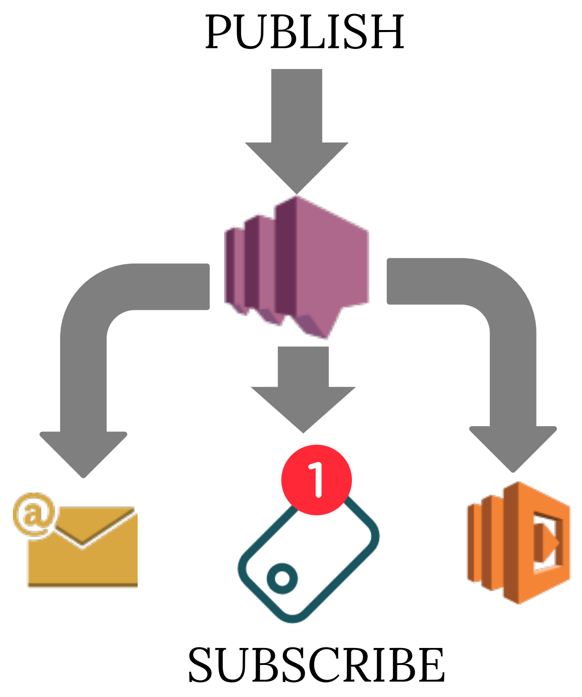


EXERCISE: API GATEWAY

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
   ```
   curl --data '{"foo":"bar2"}' https://nnnnnnnnn.execute-
   api.us-east-1.amazonaws.com/dev/echo
   ```

AMAZON 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



EXERCISE: AMAZON 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 message to the topic
5. Go to Services → CloudWatch
6. Open Logs for the Lambda function
7. Logs show the SNS message sent above (and the messages from earlier tests)

SERVERLESS FRAMEWORK

SC5

SERVERLESS FRAMEWORK

- Open-source application framework to easily build serverless architectures
- JAWS was first introduced in October 2015

WHY SERVERLESS FRAMEWORK

- Plug-in mechanism to enhance and customize the development experience
- Capability (via plugins) to run Lambda functions locally (run & test without deployment)
- Easier to debug (consume logs locally)
- Multi-cloud support (AWS / Azure / Google / Bluemix / Kubeless)

SC5 CONTRIBUTIONS

- Serverless Framework partner (1 of 4 worldwide) https://serverless.com/partners/
- Plugins: Mocha & Jest plugins for Test-Driven Development, KMS, many others
- Boilerplates: SC5 Serverless Boilerplate, Authentication Boilerplate
- Workshops: Blog workshop, ChatBot workshop
- Code contributions to the Serverless Framework
- Presence in GitHub and other channels

SERVERLESS FRAMEWORK DEMO

SC5

THE WORKSHOP

SC5

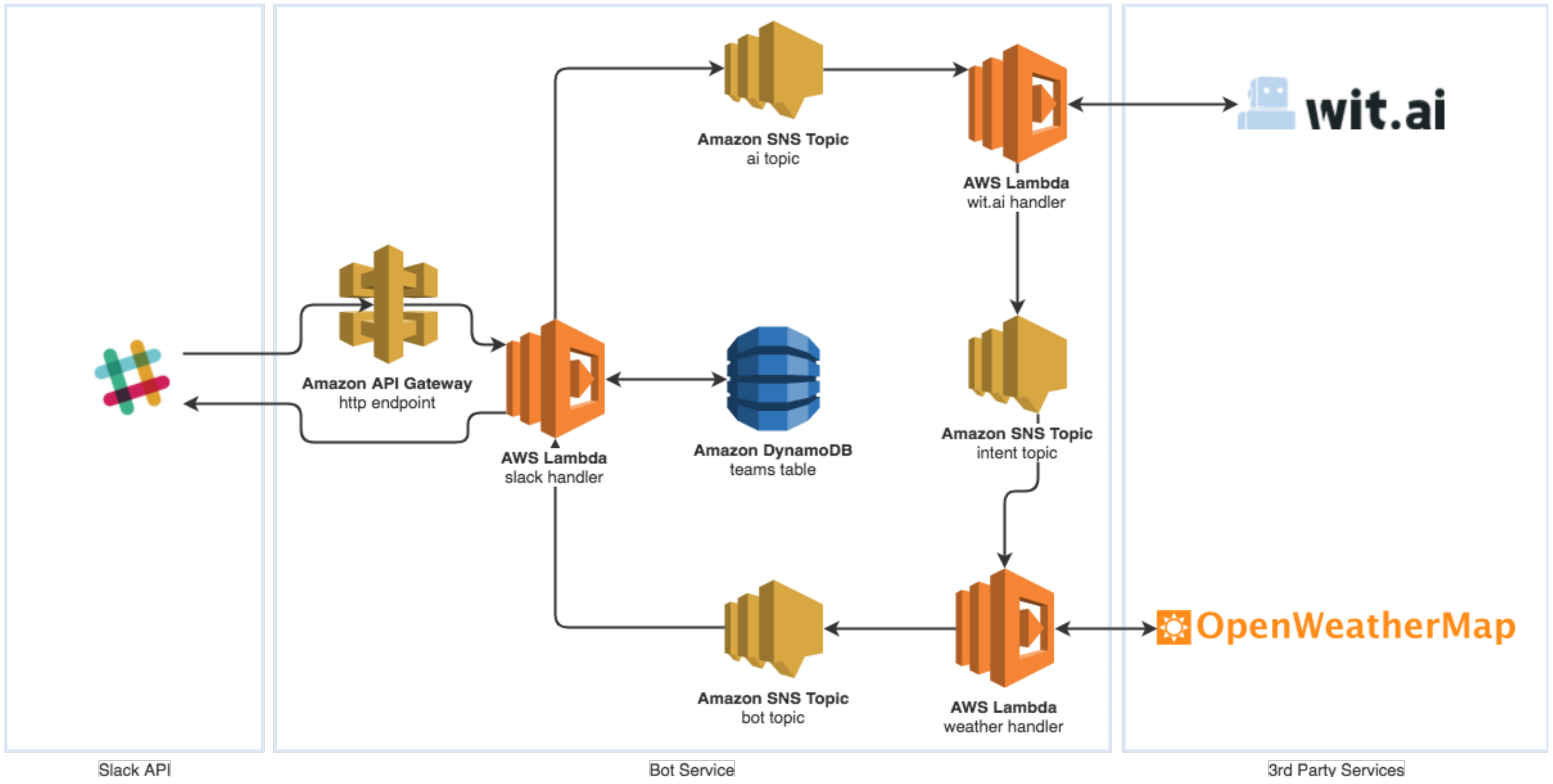
The Workshop

COMPONENTS

- Slack / Bot
- Wit.ai / Natural Language Processing
- OpenWeatherMap / Weather and forecast data
- AWS
  - AWS Lambda
  - Amazon SNS
  - Amazon API Gateway
  - Amazon DynamoDB

The Workshop

ARCHITECTURE



SC5

SET UP SERVERLESS PROJECT

- Install boilerplate:
  ```
  sls install \
  --url https://github.com/SC5/slack-chatbot-workshop \
  --name my-slack-weatherbot
  ```
- Change directory: `cd my-slack-weatherbot`
- Optional, set node version to match AWS Lambda environment: `nvm use`
- Install dependencies: `npm install`
- Test installation: `sls` (should have e.g. function create method)
- Rename example.env.yml: `mv example.env.yml .env.yml`

NOTICE: when copy-pasting code snippets, line breaks can be an issue

SLACK

SC5

ABOUT SLACK

- Slack is an instant messaging platform
- 8 million weekly active users

SET UP APPLICATION

- Set up Slack team https://slack.com/create (if you don't already have one)
- Select "Create new App" https://api.slack.com/apps
  - Add name and select account for the application
- On "Add features and functionality" select "Bots" and then "Add a Bot User"
  - Select default username
  - Set "Always Show My Bots as Online" to "On"

SLACK SECRETS

- Copy the app credentials from Settings → Basic Information and paste them to the .env.yml

```
SLACK_CLIENT_ID: nnn
SLACK_CLIENT_SECRET: nnn
SLACK_VERIFICATION_TOKEN: nnn
```

Slack

INSTALL ENDPOINT

- Create endpoint with:

```
sls create function \
-f slack-install \
--handler slack/install/index.handler
```

- Add http event under the handler in serverless.yml

```
events:
  - http:
      path: slack/install
      method: get
```

OAUTH REQUEST

- Create a function to handler that authenticates against Slack OAuth

```
const authenticate = (event) => {
  let code = null;
  if (event.queryStringParameters && event.queryStringParameters.code) {
    code = event.queryStringParameters.code;
  } else {
    throw new Error('No code available');
  }

  const params = {
    client_id: process.env.SLACK_CLIENT_ID,
    client_secret: process.env.SLACK_CLIENT_SECRET,
    code,
  };

  const url = `https://slack.com/api/oauth.access?${qs.stringify(params)}`;
  return fetch(url)
    .then(response => response.json())
    .then(log)
    .then((json) => {
      if (json.ok === true) {
        return json;
      }
      throw new Error('Slack connection error');
    });
};
```

OAUTH REQUEST

- When OAuth endpoint responses, Amazon API Gateway should redirect to callback URL

```
const response = (error) => ({
  statusCode: 302,
  headers: {
    Location: error
      ? `${process.env.INSTALL_CALLBACK_URL}#error`
      : `${process.env.INSTALL_CALLBACK_URL}#success`,
  },
});
```

Slack

OAUTH REQUEST

- Update the handler to save the team and make request to OAuth endpoint

```
module.exports.handler = (event, context, callback) =>
  authenticate(event)
    .then(saveTeam)
    .then(() => callback(null, response(null)))
    .catch(error =>
      log(error.toString())
        .then(() => callback(null, response(error))));
```

SC5

ADD REDIRECT URL TO SLACK APP

- Deploy the service with `sls deploy`
- Copy endpoint URL from Serverless CLI output
- Paste the endpoint to https://api.slack.com/apps → OAuth & Permissions → Redirect URLs → Add a new Redirect URL
- Click: Save URLs
- Copy SLACK_CLIENT_ID from .env.yml and
- Open http://slackbot-workshop.sandbox.sc5.io
- Paste client id to text field and click Save
- Click Add to Slack button and Authorize the Slack application
- Now your application should be installed to your slack team

EVENTS HANDLER

- Create events handler:
  ```
  sls create function -f slack-
  events --handler
  slack/events/index.handler
  ```
- Let’s start with logging the input event in events handler

```
'use strict';

const log = require('../../shared/log');

module.exports.handler =
  (event, context, callback) => {
    log(event);
    callback(null, 'ok');
  };
```

Slack

EVENTS ENDPOINT

- Add http event to events handler and then deploy with `sls deploy`
- Copy the events endpoint URL from the Serverless CLI output
- Open https://api.slack.com/apps/ select your application and go to Event Subscriptions
- Enable Event Subscriptions and paste events endpoint URL to request URL field
- Check logs with: `sls logs -f slack-events`

```
events:
  - http:
      path: slack/events
      method: post
```

SC5

RESPOND TO THE CHALLENGE

- Create a function to *slack/events/index.js* which verifies that the token is same that we have in out environmental variables

```
const verifyToken = (token) => {
  if (token === process.env.SLACK_VERIFICATION_TOKEN) {
    return Promise.resolve('ok');
  }
  return Promise.reject('Invalid token');
};
```

SC5

RESPOND TO THE CHALLENGE

- Create the response payload
- If the message type is `url_verification`, the challenge is send back to Slack Platform
- For any other calls empty 200 response is send

```
const createResponse = (slack) => {
  const payload = {
    statusCode: 200,
  };
  if (slack.type && slack.type === 'url_verification') {
    Object.assign(payload, {
      headers: {
        'content-type': 'text/plain',
      },
      body: slack.challenge,
    });
  }
  return payload;
};
```

SC5

RESPOND TO THE CHALLENGE

- Create the promise chain for callback
- Verify that token is ok
- Send response as callback
- Keep the Slack Platform happy but sending 200 even if when failing
- Then deploy and retry the challenge

```
module.exports.handler = (event, context, callback) => {
  log(event);
  const slack = JSON.parse(event.body);
  return verifyToken(slack.token)
    .then(() => callback(null, createResponse(slack)))
    .catch(error =>
      log(error.toString())
        .then(() => callback(null, createResponse(slack))));
};
```

SC5

SUBSCRIBE TO BOT EVENTS

- Now that the challenge is ok, add event subscriptions
- Click Add Bot User Event and select *message.channels* and *message.im*
- Click **Save Changes**
- Open http://slackbot-workshop.sandbox.sc5.io/ and verify the changes by authorizing the application again
- Test if the Slack messages are logged with `sls logs -f slack-events -t`

SEND RESPONSE TO SLACK

- Create a function that sends response to Slack API

```
const sendResponse = (params) =>
  fetch(`https://slack.com/api/chat.postMessage?${qs.stringify(params)}`)
    .then(response => response.json())
    .then((response) => {
      if (response.ok !== true) {
        throw new Error('Slack connection error');
      }
      return true;
    });
```

SC5

CHECK BOT MENTION

- Check that the bot is mentioned in the message

```
const checkBotMention = (slack) => {
  const botnameRegExp = new RegExp(`<@${slack.team.bot.bot_user_id}>`);
  if (botnameRegExp.test(slack.event.text)) {
    return slack;
  }
  throw new Error('Bot not mentioned');
};

const removeBotMention = (slack) => {
  const botnameRegExp = new RegExp(`<@${slack.team.bot.bot_user_id}>\\s*`);
  const text = slack.event.text.replace(botnameRegExp, '');
  Object.assign(slack.event, { text });
  return slack;
};
```

PROCESS MESSAGE

- Process the incoming message

```
const processMessage = (slack) => {
  if (slack.type && slack.type !== 'url_verification') {
    return database.getTeam(slack.team_id)
      .then(team => Object.assign({}, slack, { team }))
      .then(slackWithTeam => checkBotMention(slackWithTeam))
      .then(slackWithTeam => removeBotMention(slackWithTeam))
      .then(log);
  }
  return true;
};
```

SEND RESPONSE TO SLACK

Update the handler to use processMessage and sendResponse functions

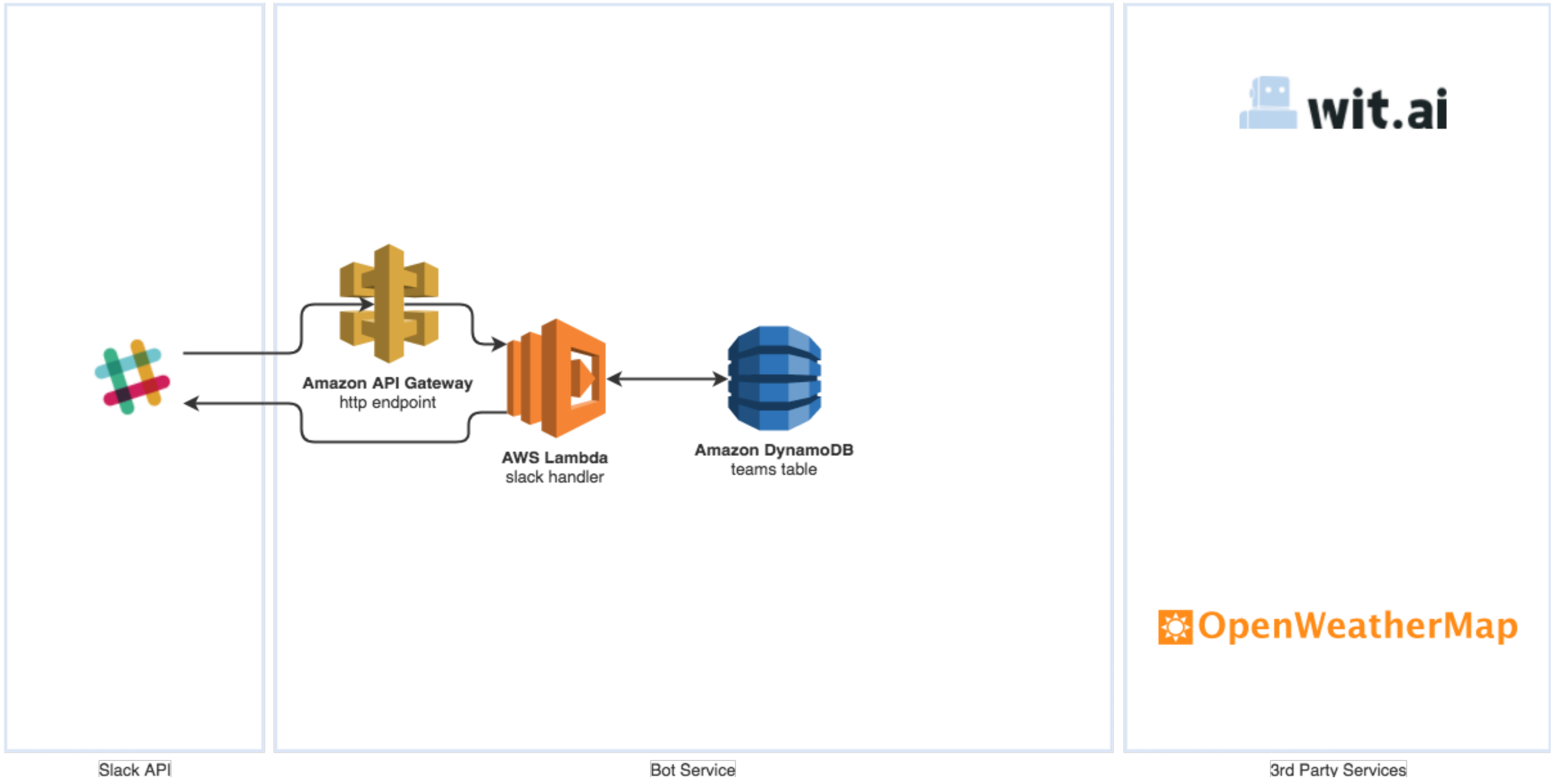
```
module.exports.handler = (event, context, callback) => {
  log(event);
  const slack = JSON.parse(event.body);
  return verifyToken(slack.token)
    .then(() => processMessage(slack))
    .then(message => sendResponse({
      token: message.team.bot.bot_access_token,
      channel: message.event.channel,
      text: 'Hello Barcelona!',
    }))
    .then(() => callback(null, createResponse(slack)))
    .catch(error =>
      log(error.toString())
        .then(() => callback(null, createResponse(slack))));
};
```

SEND RESPONSE TO SLACK

- Deploy with `sls deploy` and test what bot response when message is sent
- Note that you need to mention bot in your message e.g. *@mybotname Hello!*

Slack

ARCHITECTURE

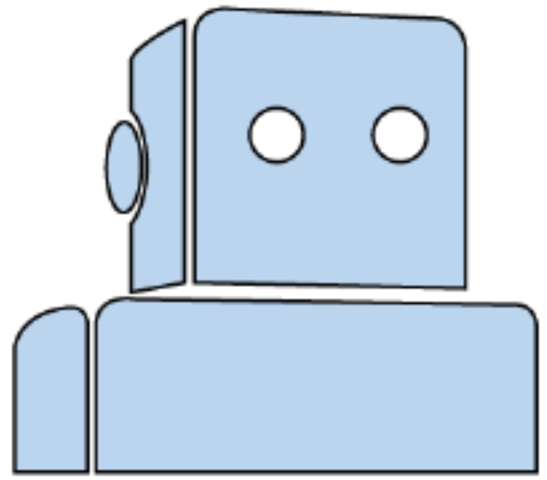


SC5

WIT.AI

ABOUT WIT.AI

- A natural language processing and speech recognition service
- Owned by Facebook
- Free to use, including commercial projects
- Supports 50 language



NLP/NLU - KEY CONCEPTS

Utterance

- Textual input from the user
- "Book me a ticket to Paris", "Booking", "Paris flight"

Intents

- Like verbs in sentences
- "BookFlight", intent for a travel application

Entities

- Like nouns in sentences
- "Paris", location entity

SET UP APPLICATION

- Login with Github or Facebook account
- Create new application by pressing +
    - Insert app name
    - Select language
    - Set application to open or private - your choice
- Press Create App

START TEACHING THE MODEL

- Under Test how your app understands a sentence type “What’s the weather in Barcelona?” to User says field
- Click “Add a new entity” and select “intent”
    - To the **Value** field type *weather* and press **Create new value “weather”**
- Double click **Barcelona** from sentence and **press Create an entity for “Barcelona”**
    - Select **wit/location**
- Click Validate
- Type “What’s the weather in Rome?” to User says field and check that if wit.ai understands it, if so, validate it
- Type ”Tell me the forecast for London, UK.” and validate if it’s OK

ADD DATE TIME ENTITY

- Type “How will be the weather in Barcelona tomorrow?”
  - Check that taught entities are ok
  - Double click “tomorrow”
  - Press Create an entity for “tomorrow” and select wit/datetime
  - Note timezone settings!
- Try “Tell me the Sunday's forecast for Berlin.” if it is ok then validate again

WIT.AI SECRETS

- Select settings and Server Access Token to .env.yml

```
WITAI_SERVER_ACCESS_TOKEN: nnn
```

CREATE HANDLER

Create handler function: `sls create function -f witai --handler witai/index.handler`

ADD EVENT TO HANDLER

Handler is triggered with SNS message. The name of the AI topic goes to environment. Add also bot topic name, that is the one slack handler will be listening
(notice: AI_TOPICNAME and BOT_TOPIC_NAME should be in to lines, PDF may corrupt the linebreaks)

```
AI_TOPIC_NAME: ${self:provider.environment.SERVERLESS_PROJECT}-
${self:provider.environment.SERVERLESS_STAGE}-ai
BOT_TOPIC_NAME: ${self:provider.environment.SERVERLESS_PROJECT}-
${self:provider.environment.SERVERLESS_STAGE}-bot
```

Add events and increate the timeout little bit, so that witai has some time to answer

```
timeout: 60
events:
  - sns: ${self:provider.environment.AI_TOPIC_NAME}
```

HANDLER FUNCTION

Create a function that requests wit.ai message endpoint

```
const witai = (event) => {
  if (event.text) {
    const client = new Wit({ accessToken: process.env.WITAI_SERVER_ACCESS_TOKEN });
    return client.message(event.text, { timezone: 'Etc/UTC' })
      .then(log)
      .then(data => ({ message: JSON.stringify(data) }));
  }
  return Promise.reject('no text');
};
```

HANDLER FUNCTION

Update the handler to remove bot username from message and send the payload to witai endpoint. getMessage and sendMessage functions can be found from shared/messaging module.

```
module.exports.handler = (event, context, callback) => {
  const message = getMessage(event);
  return witai(message.event)
    .then((response) => {
      Object.assign(message, { responseText: response.message });
      return sendMessage(process.env.BOT_TOPIC_NAME, { message });
    })
    .then(() => callback(null, 'ok'))
    .catch(error =>
      log(error.toString())
        .then(() => callback(null, error)));
};
```

SC5

Wit.ai

SLACK HANDLER

Add SNS event to slack-events handler in serverless.yml

```
- sns: ${self:provider.environment.BOT_TOPIC_NAME}
```

The slack-events handler should also send the SNS message that witai handler receives, add processMessage and sendMessage to the promise chain

```
if (event.httpMethod) {
  const slack = JSON.parse(event.body);
  return verifyToken(slack.token)
    .then(() => processMessage(slack))
    .then(message => sendMessage(process.env.AI_TOPIC_NAME, { message }))
    .then(() => callback(null, createResponse(slack)))
    .catch(error =>
      log(error.toString())
        .then(() => callback(null, createResponse(slack))));
}
return callback(null, 'Invalid event type');
```

SC5

Wit.ai

SLACK HANDLER

When the received event is SNS send message's responseText to the Slack channel

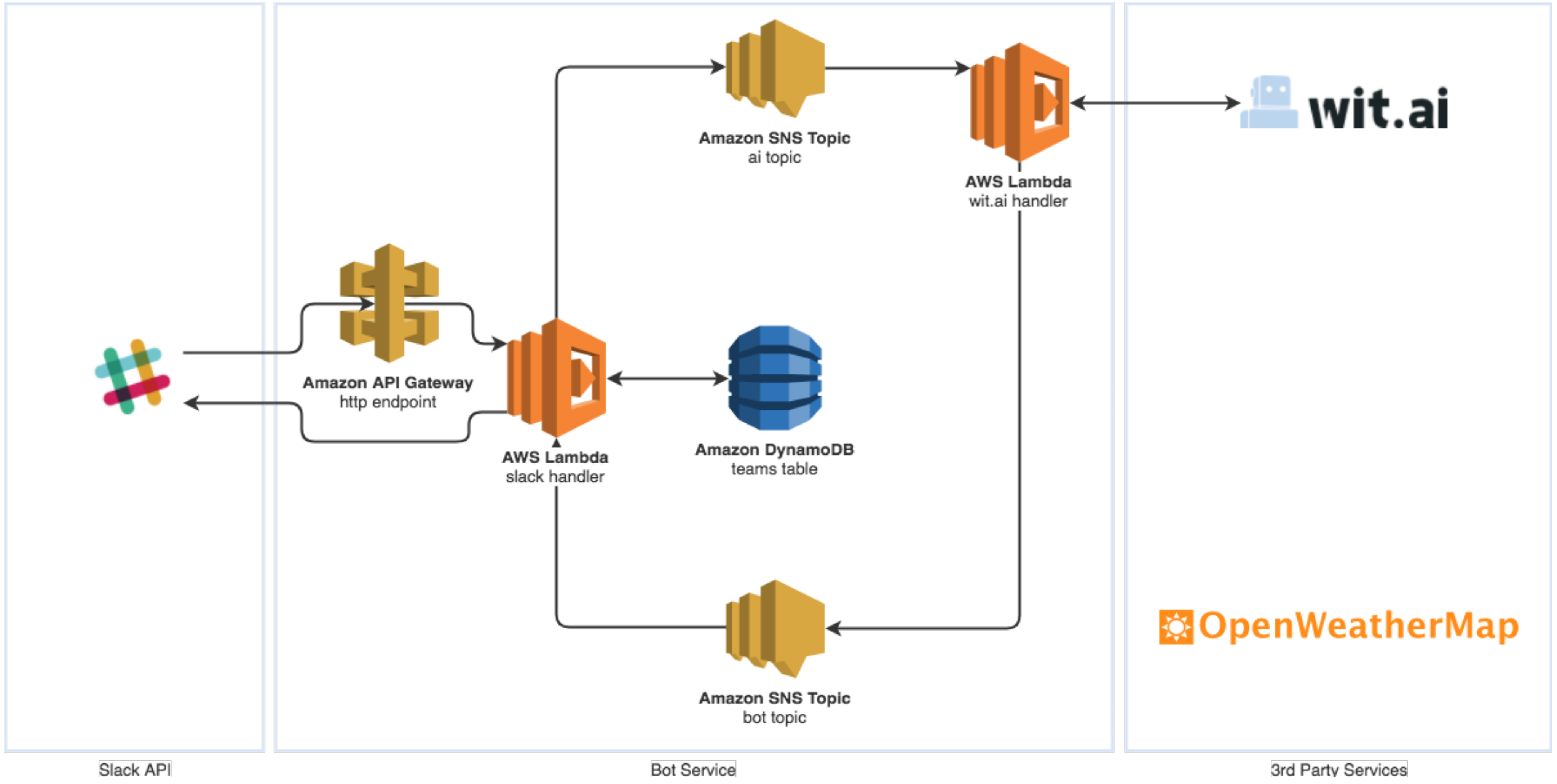
```
} else if (event.Records && event.Records[0].EventSource === 'aws:sns') {
  const message = getMessage(event);
  log(message);
  return sendResponse({
    token: message.team.bot.bot_access_token,
    channel: message.event.channel,
    text: message.responseText,
  })
    .then(callback(null, 'ok'))
    .catch((error) =>
      log(error.toString())
        .then(() => callback(error)));
}
```

SC5

DEPLOY AND TEST

- Deploy the project with sls deploy and test by sending a message from Slack

ARCHITECTURE



SC5

OPENWEATHERMAP

SC5

ABOUT OPENWEATHERMAP

- Current conditions and forecast for 200,000+ cities and any geo location
- Data and database are open and licened by Open Data Commons Open Database License (ODbL)

OpenWeatherMap

SET UP API ACCESS

- Create an account → https://home.openweathermap.org/users/sign_up
- From https://home.openweathermap.org/ select API keys and copy api key to .env.yml

```
OPENWEATHERMAP_API_KEY: “nnn”
```

OpenWeatherMap

CREATE HANDLER

- Create handler for weather: `sls create function -f weather --handler weather/index.handler`
- Create an SNS topic and subscription for the intent

```
INTENT_TOPIC_NAME:
${self:provider.environment.SERVERLESS_PROJECT
}-
${self:provider.environment.SERVERLESS_STAGE}-
intent
```

```
timeout: 30
events:
  - sns:
${self:provider.environment.INTENT_TOPIC_NAME}
```

SC5

OpenWeatherMap

FETCH WEATHER DATA

- Create a function to the weather/index.js that fetches current weather from OpenWeatherMap API by location name

```
const weatherByLocationName = (locationName) => {
  const params = {
    q: locationName,
    APPID: process.env.OPENWEATHERMAP_API_KEY,
  };

  return
fetch(`http://api.openweathermap.org/data/2.5/weather?${qs.stringify(params)
}`)
    .then(res => res.json())
    .then(mapWeatherData);
};
```

SC5

OpenWeatherMap

MAP WEATHER DATA

Set default values and create helper function for converting Kelvins to Celcius degrees

```
const defaultWeatherData = {
  temperature: 'unknown',
  description: 'unknown',
  location: 'unknown',
  icon: '10d',
};

const kelvinToCelsius = k => Math.round(k - 273.15);
```

SC5

OpenWeatherMap

MAP WEATHER DATA

```
const mapWeatherData = (data) => {
  const description = data.weather
    ? data.weather[0].description
    : defaultWeatherData.description;

  const icon = data.weather
    ? data.weather[0].icon
    : defaultWeatherData.icon;

  const temperature = data.main && data.main.temp
    ? kelvinToCelsius(data.main.temp)
    : defaultWeatherData.temperature;

  return {
    temperature,
    description,
    icon,
    location: data.name,
    date: moment(data.dt * 1000).format(),
  };
};
```

OpenWeatherMap

UPDATE HANDLER

- Update handler to get weather data from OpenWeatherMap API

```
module.exports.handler = (event, context, callback) => {
  const message = getMessage(event);
  const meaning = JSON.parse(getMessage(event).responseText);
  return weatherByLocationName(meaning.entities.location[0].value)
    .then(data => Object.assign({}, message, { responseText:
JSON.stringify(data) }))
    .then(result => sendMessage(process.env.BOT_TOPIC_NAME, { message:
result }))
    .then(() => callback(null, 'ok'));
};
```

SC5

OpenWeatherMap

UPDATE WITAI HANDLER

- Update witai handler to send the SNS message to intent topic

```
return sendMessage(process.env.INTENT_TOPIC_NAME, { message });
```

- Deploy the project with `sls deploy` and test with Slack
- The response in Slack should be raw, stringified JSON.

SC5

OpenWeatherMap

FETCH FORECAST DATA

```
const forecastByLocationName = (locationName, datetime) => {
  const time = moment(datetime);
  const timestampInSecondsStart = time.valueOf() / 1000;
  const timestampInSecondsEnd = time.add(3, 'hours').valueOf() / 1000;
  const params = {
    q: locationName,
    APPID: process.env.OPENWEATHERMAP_API_KEY,
  };

  return fetch(`http://api.openweathermap.org/data/2.5/forecast?${qs.stringify(params)}`)
    .then(result => result.json())
    .then((data) => {
      const forecastsInDatetime = data.list.slice().filter(({ dt }) =>
        (dt >= timestampInSecondsStart && dt < timestampInSecondsEnd));
      log({ forecastsInDatetime });
      const weatherData = forecastsInDatetime.length > 0 ? forecastsInDatetime[0] : {};
      return mapWeatherData(weatherData);
    });
};
```

OpenWeatherMap

MAP ICONS

- Use emonjies as weather icons
- XXd is daytime icon
- XXn is nighttime icon

```
const mapIcon = (icon) => {
  switch (icon) {
    case '01d':
      return ':sunny:';
    case '01n':
      return ':sunny:';
    case '02d':
      return ':sun_small_cloud:';
    case '02n':
      return ':sun_small_cloud:';
    case '03d':
      return ':sun_behind_cloud:';
    case '03n':
      return ':sun_behind_cloud:';
    case '04d':
      return ':cloud:';
    case '04n':
      return ':cloud:';
    case '09d':
      return ':rain_cloud:';
    case '10d':
      return ':partly_sunny_rain:';
    case '11d':
      return ':thunder_cloud_and_rain:';
    case '13d':
      return ':snow_cloud:';
    case '50d':
      return ':fog:';
    default:
      return ':partly_sunny_rain:';
  }
};
```

SC5

OpenWeatherMap

RESPONSE TEMPLATE

- Create a response template
  (notice the linebreaks)

```
const createResponse = (data) =>
  `${moment(data.datetime).calendar()} in
${data.locationName}:\n${mapIcon(data.icon)} ${data.description} and
${data.temperature}°C.`;
```

- And add helper for datetime

```
const getDatetime = (entities) => {
  if (entities.datetime) {
    if (entities.datetime[0].type === 'interval') {
      return entities.datetime[0].from.value;
    }
    return entities.datetime[0].value;
  }
  return Date.now();
};
```

OpenWeatherMap

UPDATE HANDLER

Update handler to return either current weather or forecast

(notice the linebreaks)

```
module.exports.handler = (event, context, callback) => {
  const message = getMessage(event);
  const meaning = JSON.parse(getMessage(event).responseText);
  const locationName = meaning.entities.location[0].value;
  const datetime = getDatetime(meaning.entities);
  return (meaning.entities.datetime
    ? forecastByLocationName(locationName, datetime)
    : weatherByLocationName(locationName))
      .then(data =>
        Object.assign({},
          message,
          { responseText: createResponse(Object.assign({ datetime, locationName
}, data)) }))
      .then(result => sendMessage(process.env.BOT_TOPIC_NAME, { message: result
}))
      .then(() => callback(null, 'ok'));
};
```

SC5

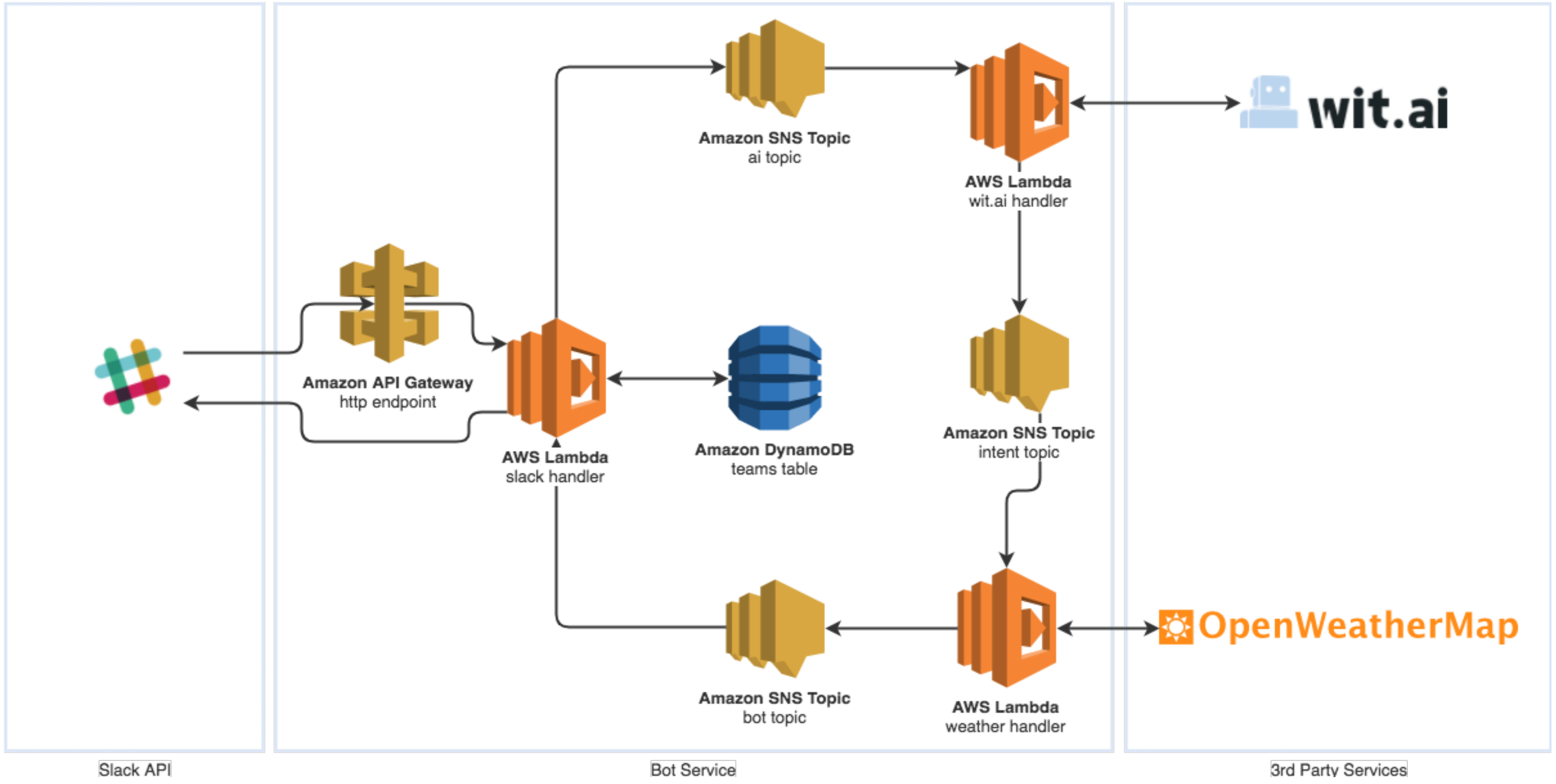
OpenWeatherMap

DEPLOY AND TEST

- Deploy the project with sls deploy and test by sending a message from Slack

OpenWeatherMap

ARCHITECTURE



SC5

ENHANCE THE BOT

SC5

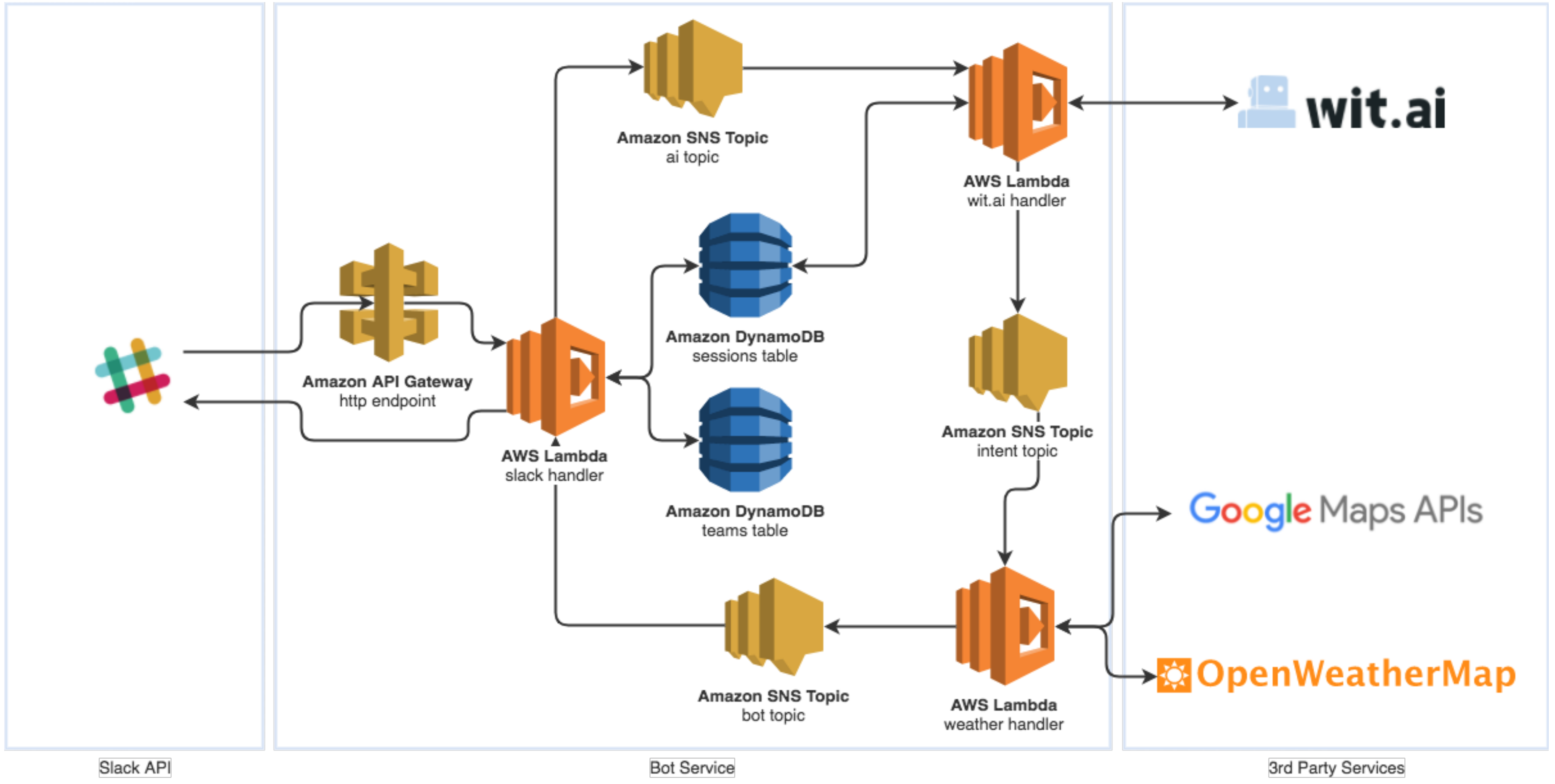
HOW TO ENHANCE THE BOT

1. Timezone & accurate position e.g. with Google maps/timezone api
2. Session or state of conversation.
   What't the weather in Barcelona? Saves "Barcelona" as last place then if "What's the weather" is sent it remembers "Barcelona" and fetches weather of Barcelona

SC5

The Workshop

ARCHITECTURE



SC5

SC5

THANK YOU!