Claranet WorkShops

Serverless deployments on AWS using Zappa

17th September 2016

PyDay BCN

claranet
www.claranet.es

vmware PARTNER
PREMIER SERVICE PROVIDER

amazon web services | Partner Network
ADVANCED CONSULTING PARTNER
MANAGED SERVICE PROVIDER

Microsoft Partner
Gold Cloud Platform
Gold Hosting
Gold Small and Midmarket Cloud Solutions
Gold Midmarket Solution Provider

PCI DSS CERTIFIED

DNV·GL ISO/IEC 27001

UKAS MANAGEMENT SYSTEMS

Preamble

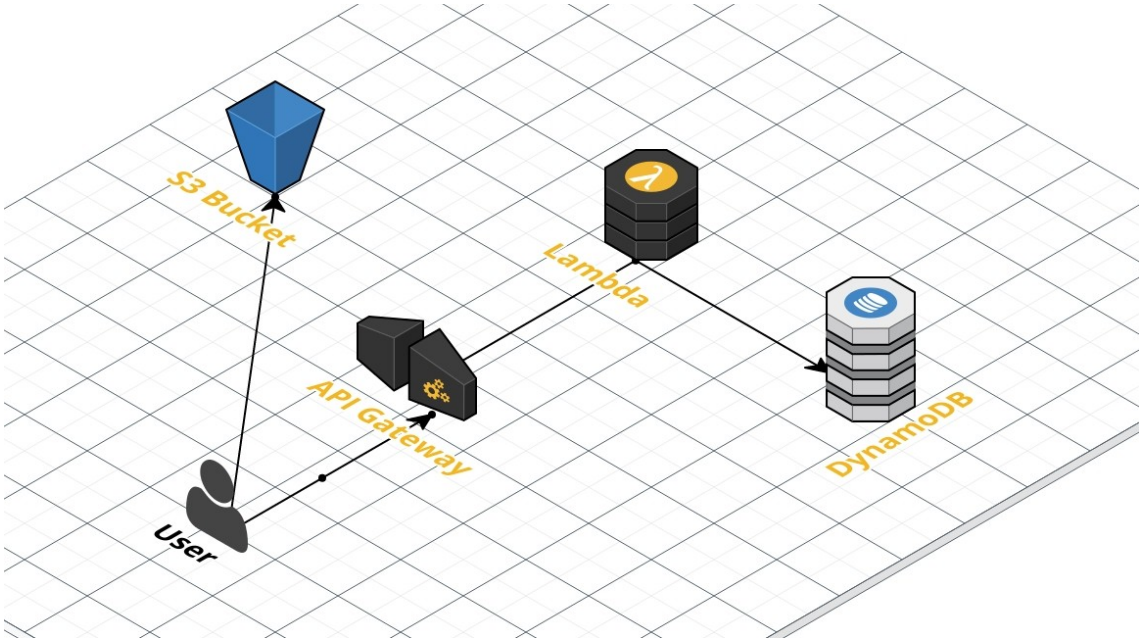
Welcome to the Serverless Deployments on AWS workshop. The goal of this session is to give you a quick vision and demonstration around how to quickly build a serverless architecture on AWS.

The Dashboard design was inspired by this blog post:
https://anmolkoul.wordpress.com/2015/06/05/interactive-data-visualization-using-d3-js-dc-js-nodejs-and-mongodb
For the rendered we will use d3.js, dc.js and crossfilter.js –yes, we know it's PyDay, but for the rendered we need some js stuff!

We have prepared an environment by creating a bucket and define a public policy on it (http://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html#example-bucket-policies-use-case-2). We have also populated a DynamoDB table with fake deployment data. The script to create them is available on the Github repository. The rest it is only permissions access for each user.

Remember that the final goal is to can easily analyze our deployments logs(informations about each deployment(status/duration/environment/…).

Below it's the architecture of the WorkShop with the use of 4 AWS services, an S3 Bucket to store and deliver the static content, API Gateway to easily create a Restfull API, Lambda to execute our Python code in response to requests performed against API Gaetway and DynamoDB to store our deployment logs.



Index

1. Environment Configuration

In this part we will configure all the requierement to do the WorkShop

1.1 Download and Configure AWS Cli

AWS Cli is an Amazon tool to interact with AWS' Services. We need to set it up because Zappa will use it to retrieve the access credentials (you can directly define the credentials in Zappa but it's not a "clean way" to do it, because if you store your Zappa project on Github, for instance, you could forget to remove your credentials from the Zappa configuration file, or forget to add this file in your .gitignore).

We also need AWS Cli to upload the static content in an S3 Bucket because currently Zappa does not support this part.

1.1.1 AWS Cli Installation & Configuration

The installation process is really easy, you just need to execute this command:

>> pip install awscli

Now you need to configure it.

For users who are already using the AWS Cli you can configure a profile with the command:
>> aws configure --profile XXXX

If you do that you will need to add in the zappa_settings.json (generate during the next part (1.2)) this line:

"profile_name": "your-profile-name"

>> aws configure

Access Key: The access key received on the Workshop Credentials request page
Secret Key: The secret key received on the Workshop Credentials request page
Default Region: eu-west-1
Output format: json

Validate the configuration of the AWS Cli by taping:

>> aws s3 ls s3://pydayworkshopbcn/

And normally you will receive:

2016-09-16 09:50:30 2462 test_users.sh

1.2 Environment configuration

In this part we will configure the environment for the project.

>> mkdir pydayworkshop && cd pydayworkshop
(If you haven't virtualenv installed: >> pip install virtualenv)
>> virtualenv workshop
>> source workshop/bin/activate
>> pip install zappa flask

Normally in the step you need to execute the "zappa init" command to create the default zappa configuration. But for this workshop you will use a pre configured zappa settings files to avoid conflicts with the others Workshopers!

So copy in this directory the file "zappa_settings.json" present in the archive. And inside, replace only **devXX** (Where XX is your_claranet_user id. Example: dev16 if your user is zappa-test-user-16 in the downloaded credentials file)

If you have configured a custom profile with aws cli, you need to add, in your zappa_settings.json :

```
"profile_name": "your-profile-name"
```

Good, we are now ready to have fun!

2. Exercise 1: Hello World Like

In this first exercise we will create a simple Hello World Like. Why "like"? Because we found the "Hello World" message so boring.

Our environment is ready, so we can start by creating a simple Flask application.

1 .1. First deployment

2.1.1 Create the app file

Create a new file named **dashboard.py** add the content below and save it:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
        return “Our democracy has been hacked!”
```

2.2.2 Deploy the code

We will deploy this first version on AWS. To do that:

>> zappa deploy **devXX**

Zappa will create all the requirements to have a working version of your code. When the deployment is done, click on the link after “Your updated Zappa deployment is live!:”

You can display detailed information about the environment deployed with the command:

>> zappa status

2.2.3 Visualize the Lambda Logs

An interesting feature of Zappa is the possibility to view the Lambda logs without opening an AWS console (it could take some seconds to actually retrieve the newest logs).

>> zappa tail

2.2.4 Update & rollback the application

Now we will go through a quick overview of how to update an application deployed with Zappa.

Update the message (“Our democracy has been hacked!”) in the **dashboard.py** by a new one.
Update the application with the command:

>> zappa update **devXX**

Another interesting command is rollback. For example you can rollback to the previous version with:

>> zappa rollback **devXX** -n 1

And now you will see the first message (“Our democracy has been hacked!”) in your web browser.

3. Simple index.html

In this exercise we will create an index.html file with a few static content files.

3.1 Create new directories and files

>> mkdir templates && mkdir your_claranet_user (ex: zappa-test-user-19)

In the archive downloaded, add the content in your_claranet_user folder. Normally, at this step, your structure should be:

```
| dashboard.py
        | templates
        | your_claranet_user
                | static
                    | lib
                    | js
                    | css
```

Create the content below for the new file **index.html** in the templates folder. **Replace your_claranet_user variables** and save it.

```
<!DOCTYPE html>
<html>
<head>
  <title>Dashboard</title>
            <link rel="stylesheet" href="https://s3-eu-west-1.amazonaws.com/pydayworkshopbcn/your_claranet_user/static/lib/css/bootstrap.min.css">
                <link rel="stylesheet" href="https://s3-eu-west-1.amazonaws.com/pydayworkshopbcn/your_claranet_user/static/lib/css/keen-dashboards.css">
                <link rel="stylesheet" href="https://s3-eu-west-1.amazonaws.com/pydayworkshopbcn/your_claranet_user/static/lib/css/dc.css"
>
```

```
                <link rel="stylesheet" href="https://s3-eu-west-1.amazonaws.com/pydayworkshopbcn/your_claranet_user/static/css/custom.css">


</head>
<body class="application">

  <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
    <div class="container-fluid">
      <div class="navbar-header">
        <a class="navbar-brand" href="./">Claranet Dashboard</a>
      </div>
    </div>
  </div>

</body>
</html>
```

3.2 Update the dashboard.py content

To use the **index.html** file we need to update the d**ashboard.py** file with:

```
from flask import Flask
from flask import render_template
from flask import Response


app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")
```

3.3 Upload the static content

We need to upload the static content in an S3 Bucket to load it. You cannot deliver static content directly through the Flask application.

To upload it, we will use the AWS Cli(because currently Zappa doesn't support this part):

>> aws s3 sync you_claranet_user s3://pydayworkshopbcn/your_claranet_user/ --region eu-west-1 (add --profile XXXX if you have define a profile in aws cli)

To verify, open in your web browser and verify that you can visualize:

3.4 Update the Zappa project

Before we update our project we will exclude the static content files to avoid to add them to the Lambda function (the impact is only on the archive size used by the Lambda function).

Update the zappa_settings.json file and add:

“exclude”: [“your_claranet_user”]

Save it and now we can update the project:

>> zappa update **devXX**

Finally, you can refresh your web page to view the modifications with a simple navigation bar on the top.

4. Extraction of the DynamoDB Datas

In this part we will retrieve the deployment documents stored in DynamoDB.

4.1 Retrieve DynamoDB content

First thing, we need to add a simple function to convert every Decimal variable in float. DynamoDB does not accept float values, so every time you want to add a DynamoDB document with a float value, you will need to convert it into Decimal. That is why when you retrieve a DynamoDB document you need to convert every Decimal value in float.

In our dashboard.py add the required libraries:

```
from decimal import Decimal
import json
import boto3
```

And then add the method to open the connection on our DynamoDB table:

```
dynamodb = boto3.resource('dynamodb', region_name='eu-west-1')
table = dynamodb.Table('deployment_tracking')
```

And after these lines add the function to convert every DynamoDB items values in float.

```
def dyno_decode(datas):
    """ Convert each decimal value in a dict to float

        :params datas dict: object to convert
        :return dict
    """
    if isinstance(datas, Decimal):
        return float(datas)
    elif isinstance(datas, list):
        return [float(i) if isinstance(i, Decimal) else i for i in datas]
    elif isinstance(datas, dict):
        return dict([(k, dyno_decode(v)) for k, v in datas.items()])
    else:
        return datas
```

4.2 Add a new Flask route

To add a new route (method), add these lines after the default route (‘/’)

```
@app.route("/get_deployments")
def get_deployments():
    """ Return deployments informations in Json

    """
    deployments = [dyno_decode(i) for i in table.scan()['Items']]
    return Response(json.dumps(deployments))
```

Here we are using the scan method to retrieve all the documents in DynamoDB, with convert every Decimal value in float, and finally it returns a list of dictionaries in Json.

4.3 One more update

Now we will update the project one more time. Still the same command:

>> zappa update **devXX**

And, normally, if you open the URL in your browser and add “/get_deployments” you will see a json will the deployments information.

5. Put everything together

Final step, we will put the glue to have our final dashboard. What we need is to modify the content of the **index.html** file and update a js file to target the URI to retrieve Json.

5.1 Update the index.html

Last steps replace the **index.html** file by the one in the archive downloaded with your credentials.

You will need also to update the file:

your_calranet_user/static/js/graph.js

And change the **XX** in the line .defer(d3.json, "/dev**XX**/get_deployments") by your claranet user id.

Update the static content with:

>> aws s3 sync you_claranet_user s3://pydayworkshopbcn/your_claranet_user/ --region eu-west-1 (more –profile XXX if needed)

And update your Zappa project

>> zappa update **devXX**

Go in your web browser, refresh the page and normally you have a beautiful(more or less) dashboard with interactive visualization.

Congratulation you've reach the end of this workshop. We hope that we like it, it's a quick overview of the possibility of Zappa and serverless but we want to give you an easy way to have an idea of the possibilities.