ProjectSDCC – My serverless Application

"COVID 19 – VACCINE" application

Giulia Menichini
University of Rome
TorVergata

giulia.menichini@students.uniroma2.eu

ABSTRACT

This paper focuses on the definition of a serverless application implemented using the services provided by AWS. The name of the application is "COVID 19 – VACCINE" and allow the user to manage information relating to COVID 19 vaccination. In particular, the user can obtain data present in a table within the DynamoDB database, delete records from the table, insert new records in the table, obtain information on the vaccine by entering only the batch of the same and update where necessary the number of doses per vaccine of a batch already present in the table.

This application will show a python-based user interface through which the user can choose which operation to perform using the buttons on the home page. Through a button at the end of the home page it will be possible to open a new window populated with other functions for the final user of the application. This front-end interfaces on a back-end with a WEB RESTful service to send get requests and obtain the desired outputs.

The application architecture uses AWS Lambda, Amazon API Gateway and Amazon DynamoDB. AWS Lambda and API Gateway allows to send and receive data through a public backend. Finally, DynamoDB provides a persistent layer in which data can be stored.

KEYWORDS

Serverless, AWS Lambda, Amazon API gateway, AWS DynamoDB, IAM role, Policies, Tkinter

†Author addresses: Giulia Menichini, University of Rome "TorvVergata" faculty of Computer Science.

WOODSTOCK'18, June, 2018, El Paso, Texas USA

https://doi.org/10.1145/1234567890

ACM Reference format:

Giulia Menichini 2021. ProjectSDCC – My serverless application: "COVID 19 – VACCINE application". In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1 Serverless

The term 'Serverless' means the services, practices and strategy that allow you to create more agile applications with which to innovate and respond more quickly to change. With serverless computing, infrastructure management tasks such as provisioning and capacity patching are handled by AWS (more complete cloud platform used to lower cost, become more agile and innovate faster), so the programmer can focus on just writing the code to serve his costumers.

The main advantages that distinguish serverless computing from the other compute models are:

- The Serverless model requires no management and operation of infrastructure, giving developers more time to optimize code and develop innovative new features and functionality
- Serverless computing runs code on-demand only
- Serverless computing enables end users to pay only for resources being used, never paying for idle capacity.

Amazon Web Services introduces serverless in 2014 with AWS Lambda but today every cloud service provider offers a serverless platform including Azure, Google Cloud and IBM Cloud.

2 DynamoDB

Amazon DynamoDB is a database that supports document-type and key-value data models that offers performance of a few

milliseconds. It is a durable, fully managed, multi region database, backup and management offering built-in in memory caching security for internet applications. DynamoDB is serverless, meaning there are no servers to set up and manage and no software need to be installed or managed. Availability and fault tolerance are built-in features that eliminate the need to configure the application architecture for these features. DynanoDB offers allocated and on-demand capacity modes, allowing you to optimize costs. It is possible to specify capacity per workload or simply pay for the resources used.

2.1 Implementation of DynamoDB

A table has been created within the DynamoDB database to store the data for the application. This was possible thanks to the use of the Amazon DynamoDB console.

Amazon DynamoDB stores data in partitions that is an allocation of storage for a table automatically replicated across multiple Availability Zones within an AWS Region. Partition management is handled entirely by DynamoDb.

A table called "VaxTable" was created for the application with partition key "lotto(stringa)" through which DynamoDB stores and retrieves each item. To write an item to the table, the database uses the value of the partition key as input to an internal hash function. The output value from the hash function determines the partition in which the item will be stored. Otherwise, to read an item from the table, it is good to specify the partition key for the item. DynamoDB uses this value as input to its hash function, yielding the partition in which the item can be found.

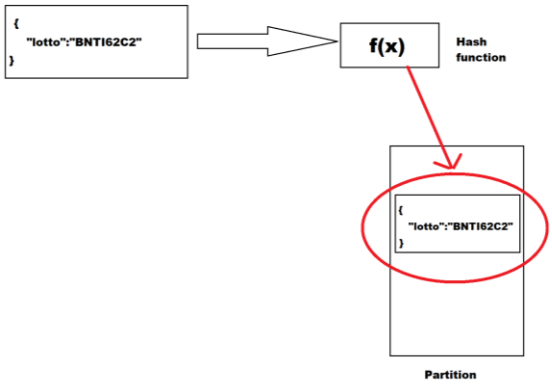

Figure 1: Representation of partitions and hash functions in DynamoDB

During creation was choose the "On Demand" setting, a simply pay-per-request pricing for read and write request. This choice made it possible to manage and balance costs and performance.

3 AWS Lambda

AWS Lambda is an event-driven, serverless computing platform, it runs code in response to events and automatically manages the computing resources required by that code.
It is possible to write Lambda functions in Node.js, Python, Go, Java and more.
An overwhelming benefit of those function is that you only pay for the resources needed to execute the function, making them an attractive option for organizations looking for a scalable and cost-effective way to add cloud functionality.
For the implementation of the application have been created five Lambda Function using Python Language:

- insertItem: function that allows the insertion of new records in the DynamoDb "VaxTable" table using putItem operation. Each record is made up of: "casa farmaceutica", "lotto", "nome vaccino", "scadenza"e "numero dosi per vaccino". The insertion of a new item requires the overwriting of an item already present in the table with same batch. To avoid this, we used a condition expression "attribute_not_exists(lotto)". This allow to proceed only if the batch of the item that the user want to insert is not already present in the table. In addition, two checks were places on the insertion of date of expiration and number of noses for vaccine. If the user enters the date in an incorrect format, he will be informed through an error message and it will therefore not be possible to continue with the insertion, if the entre of the number of doses for vaccine does not include an integer the user will be informed with a message printed in the output and only after the change will be able to insert the new record.

- deleteItem: function that allow the elimination of an item from the "VaxTable" table by the primary key (lotto) using deleteItem operation. To prevent the function from eliminating an item that is not present in the table, a condition expression "attribute_exists(lotto)" has been places on the batch. If the batch is not present in the table, an error message is thrown at the output.

- updateItem: function that allow to update "numero di dosi per vaccino" of an item already present in the table using UpdateItem operation. By default this operation edits an existing item's attribute, or adds a new item to the table if it does not already exists. To prevent this, a

condition expression "attribute_exists(lotto)" has been places on the batch. If the batch is not present in the table, an error message is thrown at the ouput.

- scanItem: function that allow to scan all the elements of the database using scanItem operation. The scan operation in Amazon DynamoDB reads every item in a table, return all of the data attributes for every item in the table. Scan always returns a result set. If no matching items are found, the result set is empty. The Scan operation performs eventually consistent reads, by default. This means that the Scan results might not reflect changes due to recently completed insertItem or updateItem operations. To avoid this, for this application was set the ConsistentRead parameter to true, this ensures that all the write operations that completed before the Scan began are included in the Scan response.

- getItem: this function allows to obtain information on an item already present in the table using the getItem operation. That operation returns a set of attributes for the item with the given primary key. An "if" has been places in the code to obtain a message ("lotto not found") that informs the user of the absence of the vaccine in the table for which he is looking for information. The getItem operation provides an eventually consistent read by default. In this application to geta strongly consistent read was set the parameter "consistentRead" to True so it always returns the last updated value.

To create, configure and manage AWS service has been used "Boto3" that is the Amazon Web Services Software Development kit (SDK) for Python which allows Python developers to write software that makes use of services like Amazon DynamoDB. To install the latest Boto3 release is used the pip command:

>>> pip install boto3

After installing, to use Boto3, is enough to import it and indicate which service or services the application is going to use.

```
import boto3
from boto3.dynamodb.conditions import Key

def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
```

Figure 2: Importing boto3

4 IAM Role and Policies

Every Lambda function has an IAM role associated with it. This role defines what other AWS services the function is allowed to interact with. For this application has been created an IAM role that grants Lambda function permission to access to write and read items to the DynamoDB table. It has been used the IAM console to create a new role selecting AWS Lambda for the role type.

During the creation in the IAM console it was necessary to attach specifically created policies to allow the function to write, read, scan and delete items on the DynamoDB database.

For the application was created manually the "VaccinePolicy" policy by defining a .json file during creation. Json file composed as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListAndDescribe",
            "Effect": "Allow",
            "Action": [
                "dynamodb:List*",
                "dynamodb:DescribeReservedCapacity*",
                "dynamodb:DescribeLimits",
                "dynamodb:DescribeTimeToLive"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SpecificTable",
            "Effect": "Allow",
            "Action": [
                "dynamodb:BatchGet*",
                "dynamodb:DescribeStream",
                "dynamodb:DescribeTable",
                "dynamodb:Get*",
                "dynamodb:Query",
                "dynamodb:Scan",
                "dynamodb:BatchWrite*",
                "dynamodb:CreateTable",
                "dynamodb:Delete*",
                "dynamodb:Update*",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:910909993862:table/VaxTable"
        }
    ]
}
```

Figure 3: File .json for the "VaccinePolicy" policy

For the "resource" property in the json file, during the creation phase was set the ARN (Amazon Resource Name) of the previously created "VaxTable" table. The ARN uniquely identifies the AWS resources.

5 API Gateway

Amazon API Gateway is a fully managed service used by developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for application to access data, business logic, or functionality from backend services.

For the application, the "ApiVaccine" API has been created, using the AWS console. For each Lambda function a resource was then created identified by the same name of the Lambda function. Every resource contains a "GET" method. After creating an API, it is necessary to deploy it to make it callable by the users of the "COVID 19 – VACCINE" application. To deploy an API, you create an API deployment and associate it with a stage. A stage is a logical reference to a lifecycle state of the API. (in "COVID 19 – VACCINE" application has been chosen "prod1"). API stages are identified by the API ID and stage name. They're included in the URL used to invoke the API. Each stage is a named reference to a deployment of the API and is made available for client applications to call. As the API evolves, it is possible to deploy it to different stages as different version of the API.

Once the API has been distributed and through the same console, it was suitably testes by entering the test query in the URL or in the AWS query label.

6 Front-end

To increase the usability of the application, it was decided to create a graphical user interface (GUI) in Python language. The frameworks for building GUI in Python can be divided into two categories:

- Coss-platform: totally portable, they always five the same look to all applications, regardless of the operating system that runs them.
- Platform-specific: They use visual elements provided by the particular operating system that runs the interface.

A cross-platform framework was used for the application: Tkinter

6.1 Tkinter

Tkinter is a library built into the standard library and is therefore immediately available in every Python installation. Tkinter offers all the basic elements of a GUI, windows, menus, buttons, Checkbutton, entry, label, listbox, toplevel, photoimage and field. They are called "widgets" in Tkinter terminology.

For the positioning of the widgets within the GUI windows, a layout manager is used. Tkinter provides three of them:

- Pack(): allow you to list in sequence the objects to be arranged in our GUI. The display order is identical to the order in which they are registered through the pack() instruction.

- Grid(): allows you to place objects within a 2-D grid. The objects are positioned through the use of the grid () instruction, indicating the row and column number.

- Place(): allow you to indicate the absolute/relative position of objects withing the GUI.

6.2 The Homepage

In the main application window, there are two buttons. The first button "Get Database Informations" through the pyrhon "requests" library allows you to execute the "scan ()" function which returns to the user the total scan of the database with the information relating to it.

In the second part of the page there are five insertion labels that allow the user to enter the values to be assigned to the new vaccine that you want to insert. The operation is confirmed through the "ADD VACCINE" button which allows you to execute the "insertVax()" function. The user is informed of the successful insertion by a message.

In the same page we find two other buttons: the first "OTHER INFORMATIONS" which allows the user to reach a second page, depending on the first, containing other buttons useful to the user for the execution of other functions. The second button is "EXIT" which allows the user to exit the application thus closing both open windows.

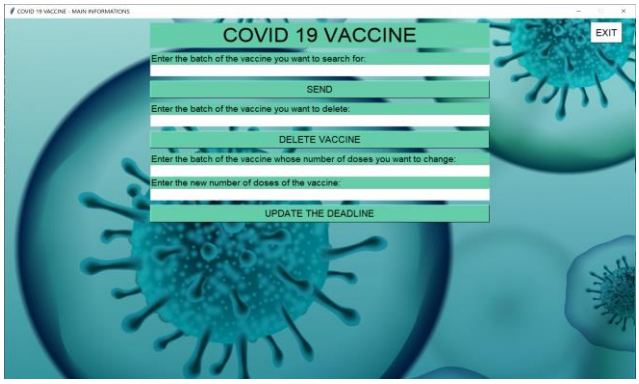

Figure 5: Home Page of the application

On the second page there are four buttons and four insertion labels. The first button allows the user to perform the getItem operation by entering the "lotto" of the vaccine to be searched for in the insertion label. In output, the user receives the vaccine information which has been parsed through the "parsing_text()" function for a clearer reading of the information. If the item is not present in the Database, the user receives in output a message "VACCINE NOT FOUND".

The second button on the page allows you to perform the deleteItem operation by entering the "lotto" of the vaccine you want to delete. If the record is not present in the "VaxTable" table, the user will be informed via an output message. Vice versa, if the elimination is successful, the user receives in output the message "VACCINE CORRECTLY ELIMINATED"

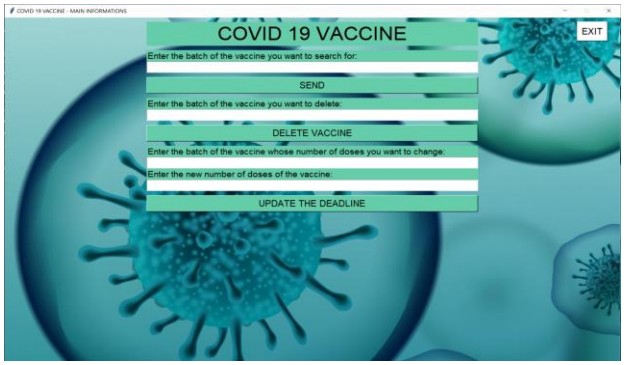

Figure 6: Second Window of the application

The third button in the window allows you to perform the updateItem operation which allows the user to update the number of doses per vaccine left in the table. The user can then enter the "lotto" of the vaccine he wishes to update and the value of the number of doses he wishes to modify. There is also a check on the insertion of this number: if the entry does not involve an integer the user will be informed with a message "number of doses for vaccine must be an integer" and will not be able to continue with the update before this change is made. Otherwise, if the update was successful the user receives the message "lotto correctly updated"

As in the main window, also in the second there is the "EXIT" button which allows the user to close both windows previously opened.

6.3 Other Python-libraries

Other Python Libraries were used for the implementation, they are:

- Python requests library that is the de facto standard for making HTTP requests in Python. It abstracts the complexities of making requests behind a simple API.
- PIL (python image library) that adds image processing capabilities to your python interpreter. Provides extensive file format support and powerful image processing capabilities. It has been used to manage the background image of the two windows.
- Json a built-in package which can be used to work with JSON data
- DateTime that supplies classes for manipulating dates and times. In the application war used to check the expiry date entered through the putItem() operation.

REFERENCES

[1] AWS Lambda, 2021. https://aws.amazon.com/lambda/.

[2] AWS API Gateway 2021 https://aws.amazon.com/it/api-gateway/

[3] AWS DynamoDB 2021 https://aws.amazon.com/it/dynamodb/

[4] AWS IAM 2021 https://aws.amazon.com/it/iam/

[5] AWS Serverless 2021 https://aws.amazon.com/it/serverless/

[6] Tkinter documentation https://docs.python.org/3/library/tk.html

[7] Pillow documentation https://pillow.readthedocs.io/en/stable/