



ENRON EMAIL PREDICTION

TCSS 531 CLOUD AND VIRTUALIZATION SYSTEMS
ENGINEERING

MEGHA CHAUHAN
MINH VU
SHANTHINI SIVARAMAN
SMRUTHI SRIDHAR

CONTENTS

INTRODUCTION.....	2
PROJECT REQUIREMENTS.....	2
ASSUMPTIONS	2
ANALYSIS AND ARCHITECTURE	2
TECHNOLOGY STACK	3
DESIGN AND DEVELOPMENT.....	6
CONCLUSION.....	11
REFERENCES	11

INTRODUCTION

The project 'Enron Email Prediction' aims to build an application where a recipient list of email addresses is predicted for any given user on inputting from and one or more To email address. This application serves as a prototype to any real time mail application system to show list of recipient suggestions to users.

PROJECT REQUIREMENTS

- Have two user input fields FROM and TO.
- Allow users to input FROM email address.
- Allow users to input one or more TO email address to start prediction process.
- Display a list of other recipient email address for the corresponding sender.
- Perform client side validation on email inputs.

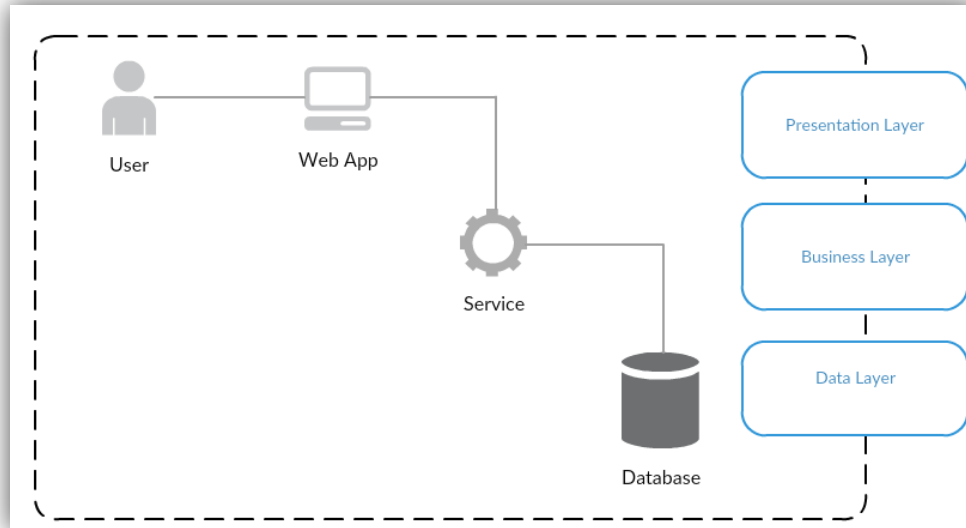
ASSUMPTIONS

- For more number of results, Top 10 recipient list is displayed.
- User cannot run the search if input fields are empty.

ANALYSIS AND ARCHITECTURE

Web Application Architecture

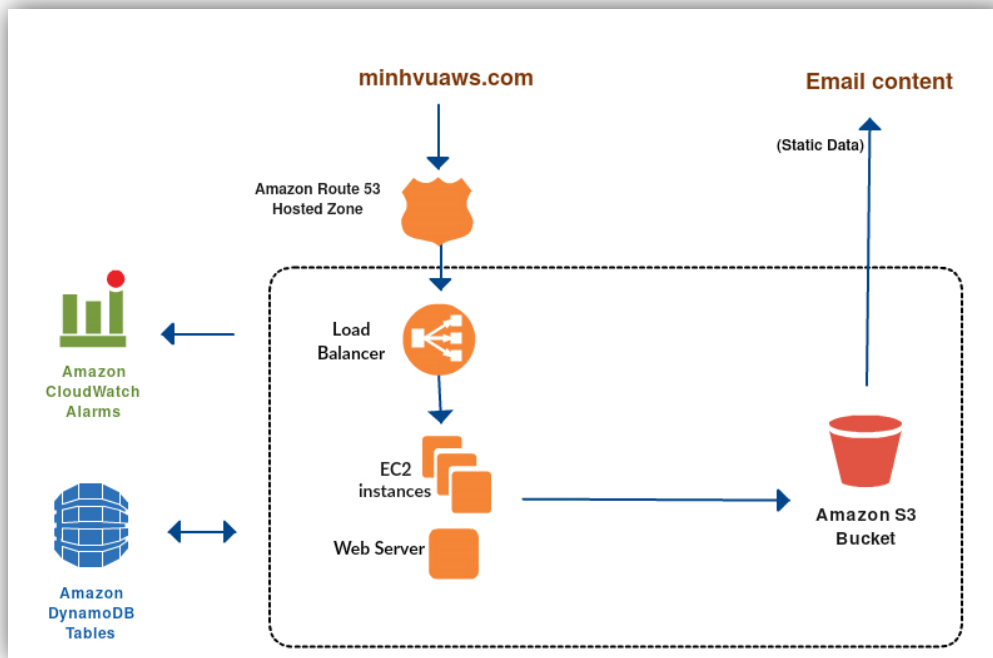
The application follows a simple **3 tier architecture** with Data logic, Business and Presentation logic as follows.



3 Tier Architecture

AWS Architecture

The application is developed on Cloud using Amazon Web Services. Highlighted below is the AWS Web application architecture.



AWS architecture

TECHNOLOGY STACK

Development Phase	Amazon Web Services / Technology	Functionalities
Code Development	Flask – Python web framework HTML and CSS	Presentation layer
	Boto3 - AWS SDK for python Flask	Business Layer
	AWS Public Data Set AWS Dynamo DB AWS Elastic Block Store Volume AWS SDK Java S3: static content	Database Layer
Provisioning	Elastic Load Balancers EC2 instance: Linux Route 53: Domain name registration	
Deployment	Elastic Beanstalk	
Monitoring	Cloud Watch	

Details of the AWS Tech Stack used:

AWS EC2 instance: We used Linux EC2 instance of type t2.micro.

Description	Status Checks	Monitoring	Tags
Instance ID	i-036360d31cca8361b		
Instance state	running		
Instance type	t2.micro		
Private DNS	ip-172-31-17-17.us-west-2.compute.internal		
Private IPs	172.31.17.17		
Secondary private IPs			
VPC ID	vpc-a20cb8c6		
Public DNS	ec2-52-40-186-196.us-west-2.compute.amazonaws.com		
Public IP	52.40.186.196		
Elastic IP	52.40.186.196		
Availability zone	us-west-2b		
Security groups	launch-wizard-1, view rules		
Scheduled events	No scheduled events		
AMI ID	ubuntu/images/hvm-ssd/ubuntu-trusty-14.04-amd64-server-20160114.5 (ami-9abea4fb)		

AWS Load balancer: We have a load balancer configured to split load across EC2 instances to increase scalability.

AWS Dynamo DB: We have used dynamo DB tables to store our data for our application.

AWS Route53: We configured Route 53 to reach our EC2 instance. We have updated the apache2 web server configuration file to point to our application.

Type	Value	Evaluate Target Health	Health Check ID
A	ALIAS linux.minhvuaws.com. (z38x2sebx8v0y0)	No	-
NS	ns-1806.awsdns-33.co.uk. ns-568.awsdns-07.net. ns-404.awsdns-50.com. ns-1419.awsdns-49.org.	-	-
SOA	ns-1806.awsdns-33.co.uk. awsdns-hostmaster.amaz	-	-
A	52.40.186.196	-	-

Edit Record Set

Name: minhvuaws.com.

Type: A - IPv4 address

Alias: ☒ Yes ☐ No

Alias Target: linux.minhvuaws.com.

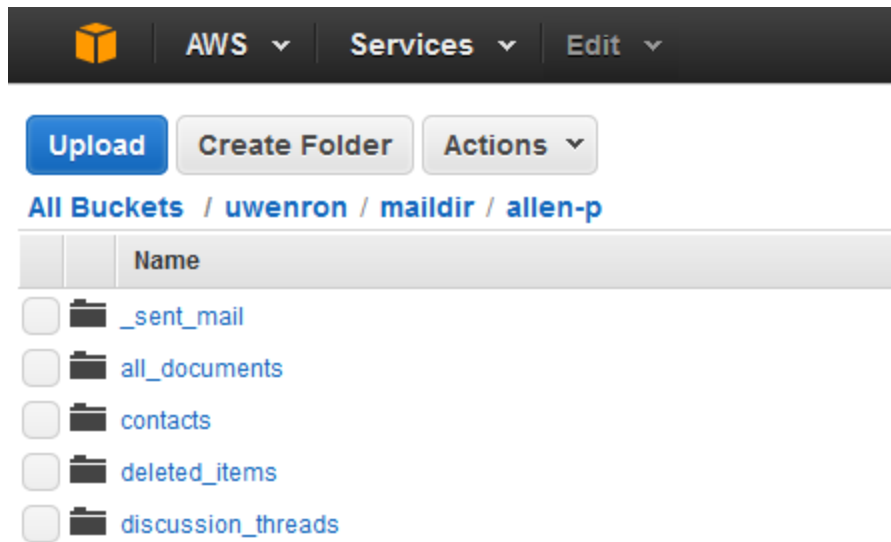
Alias Hosted Zone ID: Z38X2SELX8V0Y0

Routing Policy: Simple

Route 53 responds to queries based only on the values in this record set. [Learn More](#)

Evaluate Target Health: ☐ Yes ☒ No

AWS S3: We have used S3 to store static content, the actual email content. Dynamo DB table entries have a field which points to the S3 email content URL.



AWS EBS: We used elastic block store for loading data into the AWS cloud environment. We used AWS public dataset snapshot for loading the ENRON data. So we created EBS volume with this snapshot and attached to an EC2 instance and extracted the data needed.

AWS SDK for Python - Boto3: We used Boto3 in order to make calls to the AWS resources like S3 and Dynamo DB.

AWS SDK for Java: We used AWS SDK for java in order to clean up and extract the data needed for our application.

AWS Elastic Bean Stalk: To publish code in EC2 instance. We created virtual environment to install all the prerequisite packages needed for application.


We used **virtualenv** tool to create python environment and add all the required packages needed for our applications. This tool creates a folder which contains all the necessary executables to use the packages that a Python project would need. We then gathered the entire package needed for our application in 'requirements.txt' file.

We then deployed our code using elastic bean stalk in AWS. We zipped our code containing the project documents and requirements.txt file. So elastic bean stalk will install all the packages mentioned in this file while created EC2 instance.

In Elastic BeanStalk, we provided application name and the environment name chose EC2 instance of type t2.micro and provided service roles. Elastic bean stalk creates the EC2 instance and installs all required packages from requirement automatically.

Enron > enron-env (Environment ID: e-en6mhr29c, URL: enron-env.us-west-2.elasticbeanstalk.com) Actions ▾

Overview Refresh




Health
Ok
[Causes](#)

Running Version

First Release

[Upload and Deploy](#)




Configuration

64bit Amazon Linux 2016.03
v2.1.0 running Python 3.4

[Change](#)

enron-env.us-west-2.elasticbeanstalk.com Search

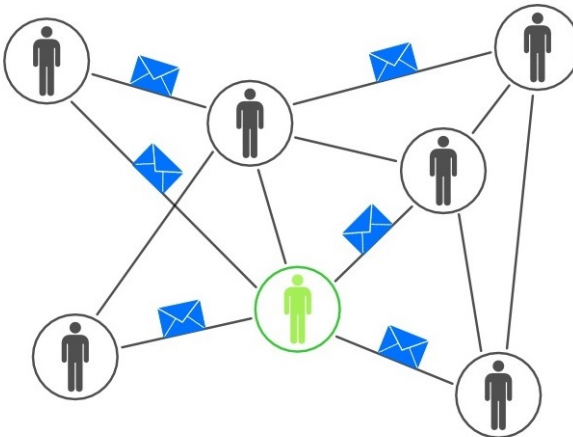


Email Prediction

Sender:

Recipients:

[Search](#)



Designed by- Megha Chauhan, Minh Vu, Smruthi, Shanthini

Cloud Watch: We used cloud watch to configure basic alarms in following resources:

EC2 instances – CPU Utilization, DiskReadBytes, DiskWriteBytes, CPU Credit Usage, CPU credit balance

Dynamo DB – Consumer Read/Write Capacity Units, Successful Request Latency

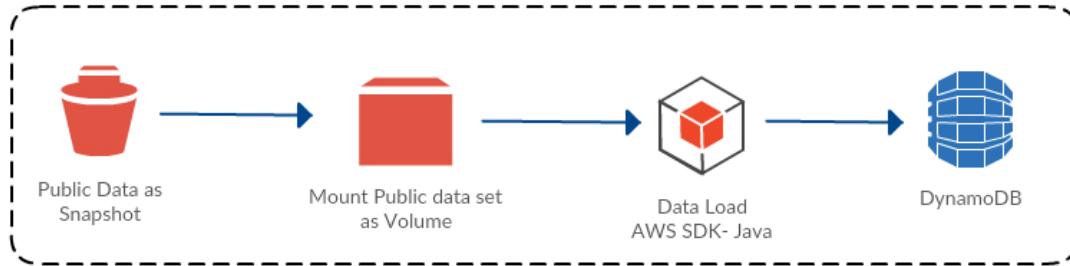
Elastic Load balancer – healthy host count, backend connection errors.

DESIGN AND DEVELOPMENT

Data Layer

AWS offers public data set as part of their service and Enron is one public data set available in the form of Elastic Block Storage. Public data sets are stored as 'Snapshots' and these snapshots are attached as EBS Volumes to EC2 instances. Once the volume is attached, it is mounted as a directory structure and made available for use in

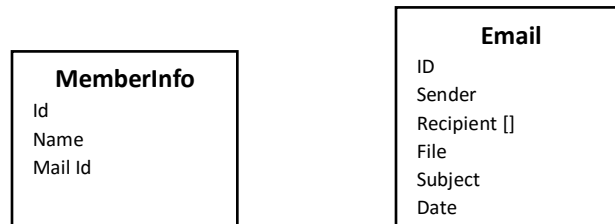
the EC2 instance. The original data set is available as a collection of email files categorized according to mail system and grouped by employee name, while the public data set AWS offers is available in the form of XMLs. The data loading follows the traditional process of **Extract, Transform and Load**. Each employee is represented as an xml file and the email transmissions are listed as contents of XMLs. The below illustrations detail more on the process.



AWS data loading

Data Design

DynamoDB has been used in the development of the application. Given below is the schema of the tables used.



Database Schema

Member

- Maintains all the enron employee email Ids and Name mappings.
- Helps to search even by name

Email

- Contains details of each email
- Maintains the email network. Sender to recipient (m: n mapping)
- Message- Points to the path containing the actual email .xml file in S3 storage (provided as part of public data set).
- Date- Date of email. Can be used to retrieve 'latest' recipient list

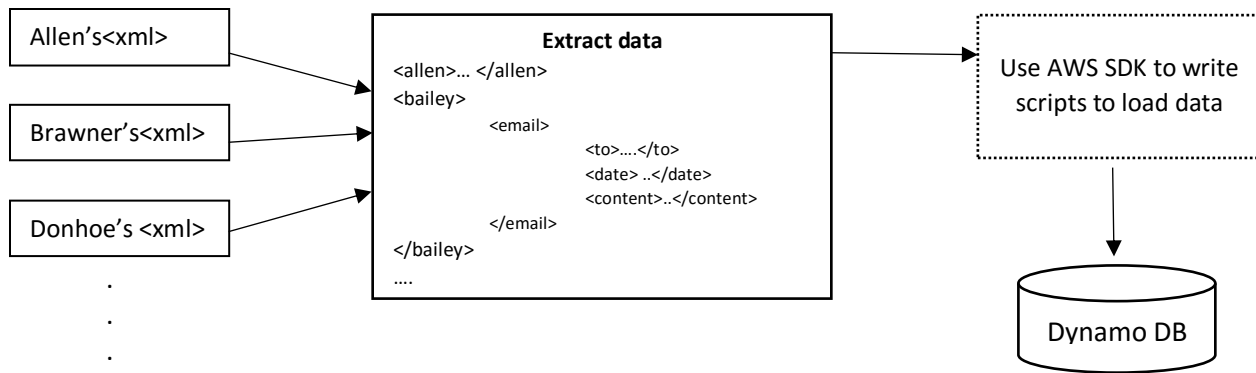
Data Loading

Extract: Each employee has a **single XML** file that contains details of emails sent and received. (total 151 xmls). The necessary details can be extracted from these XMLs (from and to email) to get a single xml. Since the data required for our application is only the network of email addresses, we parse the XML and extract the necessary sender and receiver information from the XMLs.

Transform: The extracted content is transformed to a DB understandable format using the AWS SDK libraries.

Load: AWS SDK for Java has been used to 'put' the huge count of 'transformed' items into DynamoDB.

From AWS public data set



Tables in Dynamo DB:

Email Table:

Email [Close](#)

Overview **Items** Metrics Alarms Capacity Indexes

[Create item](#) [Actions](#) ▼

Scan: [Table] Email: Email, Name ▼

<input type="checkbox"/>	Email	Name
<input type="checkbox"/>	david.w.delainey@enron.com	David W Delainey
<input type="checkbox"/>	mark.guzman@enron.com	Mark Guzman
<input type="checkbox"/>	ryan.slinger@enron.com	Ryan Slinger
<input type="checkbox"/>	benjamin.rogers@enron.com	Benjamin Rogers
<input type="checkbox"/>	andy.zipper@enron.com	Andy Zipper
<input type="checkbox"/>	stephanie.panus@enron.com	Stephanie Panus
<input type="checkbox"/>	jason.williams@enron.com	Jason Williams
<input type="checkbox"/>	andrew.h.lewis@enron.com	Andrew H Lewis
<input type="checkbox"/>	john.hodge@enron.com	John Hodge
<input type="checkbox"/>	carol.st.clair@enron.com	Carol St Clair

Enron Table:

Enron Close

Overview Items Metrics Alarms Capacity Indexes Triggers Access control

Create item Actions

Scan: [Table] Enron: RowId, From Viewing 1 to 10

	RowId	From	DateSent	File	Subject	To
<input type="checkbox"/>	N43752	pete.davis@...	Thu, 18 Oct ...	maildir/dean-...	Start Date: 1...	bert.meyers@enron.com, bill.williams@enron...
<input type="checkbox"/>	N16281	aleonard@ca...	Tue, 6 Jun 20...	maildir/badee...	CAISO Notice...	marketparticipants@caiso.com
<input type="checkbox"/>	N48192	david.delaine...	Tue, 28 Nov ...	maildir/delain...	DeAcero	David W Delainey/HOU/ECT@ECT, Janet R Dl...
<input type="checkbox"/>	N32478	louis.soldano...	Fri, 10 Dec 1...	maildir/camp...	PCB Deconta...	Louis Soldano/ET&S/Enron@ENRON
<input type="checkbox"/>	N3910	phillip.allen@...	Fri, 2 Feb 20...	maildir/allen-...		info@geoswan.com
<input type="checkbox"/>	N34314	johnnie.white...	Fri, 3 Aug 20...	maildir/camp...	Eott Crude oil...	larry.campbell@enron.com
<input type="checkbox"/>	N8715	john.arnold@...	Fri, 2 Mar 20...	maildir/arnold...	silverman	jarnold@enron.com
<input type="checkbox"/>	N44857	george.hople...	Wed, 13 Dec ...	maildir/dean-...	Weekly Monit...	Kevin.M.Presto@enron.com, George.Hopley@...
<input type="checkbox"/>	N26533	daniel.musch...	Mon, 12 Nov ...	maildir/baugh...	24 hr 1-800 p...	don.baughman@enron.com, jae.black@enron...

Business Layer

The business logic is implemented using Python.

AWS SDK for python Boto3 → we have used the AWS SDK for python – Boto3 interface for making calls to the Dynamo DB. The results obtained by this call will be in JSON format.

Flask → We use Flask which is a python web framework which provides tools and libraries to build web application. Flask projects contain a structure where it contains folders like template, statics, models, pages.

The pages contain the code for the actual business logic where the logic for the email ID's is implemented.

The template contains the user view of the application in which user input is obtained and the results are displayed. It is a HTML page with a form.

The static folder contains the styles needed by the web application.

The model contains the class definition of the model. In our project it is the definition of the user and the email content.

Business Logic:

The user input is captured and we run a scan on the table "Enron" to match the user input. Scan the dynamo DB table with the user input. Match the FROM email Id and TO email ID and extract all the rows which contain these values and list the other recipients to whom the mail will be sent.

The scan query to the dynamo DB is done using boto3. To run a scan on a table on a particular key or attribute value we need to import the boto3.dynamodb.conditions.Key and boto3.dynamodb.conditions.Attr classes.

Dynamo DB results are captured in the response attribute in boto3 and the render_template method is used for routing the app to the html page to display the results. POST and GET method are used to perform these operations. POST and GET are http methods that tell what the server wants to do. GET method tells server to get the requested information. POST method helps to capture what the user entered in the input field.

The app route then passes the results of the query using render template to the html page in the presentation layer.

Functionalities used in Flask:

App route: Routing uses URL to route the application to the desired function in the code. We use this to redirect the html link in Flask.

Request Object: This is a HTTP method helping us handle GET and POST request from the server.

Response Object: Server response list of JSON Object contained in our Dynamo db.

Render Template: will redirect the app to the template folder where the HTML code exists.

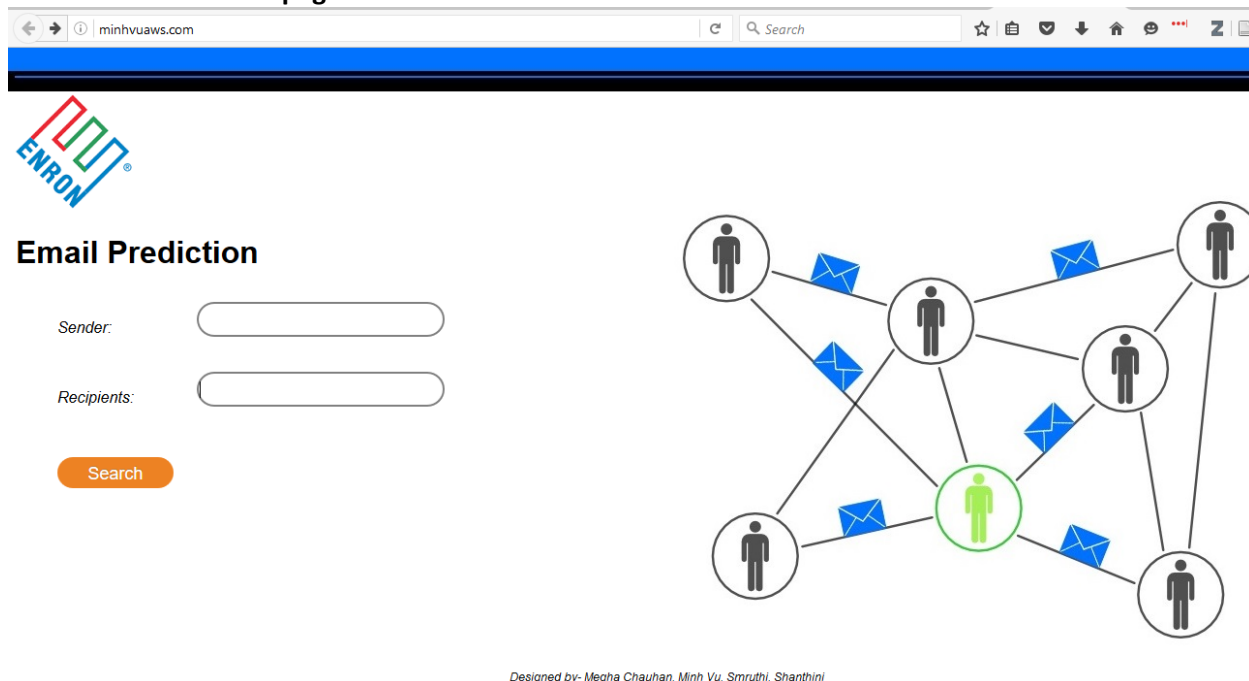
JSONIFY: it helps to send JSON response to the browser.

Presentation Layer

The html page in the template folder acts as the UI layer. The results are displayed which displays the corresponding list of email recipients for the FROM and TO email ID entered by the user.

Output will display the other recipients in the email conversation from the user entered FROM and TO email ID and also list the actual email content.

We have then added the actual email content of the conversation in S3 and user can look into the content by clicking on more details.

Preview of Enron web page:

CONCLUSION

By designing and implementing this application we have learnt

- How to use the different resources from AWS stack to build a web app.
- How we can host an entire web application in cloud infrastructure.
- How to integrate different technology components together in cloud.
- Scaling and configuration of application.

REFERENCES

1. <https://aws.amazon.com/datasets/enron-email-data/>
2. <http://docs.aws.amazon.com/amazondynamodb/latest/developerguide>
3. <http://flask.pocoo.org/docs/0.11/>
4. <https://boto3.readthedocs.io/en/latest/guide/dynamodb.html>