Cloud Automation

Ayan Agrawal(aka398), Ashish Jain(aj2429), Surbhi Thole(sst390), Fiayaz Sundrani(fs1459)

Abstract-This project is developed to automate the cloud instance creation and make functionalities easily accessible to the user. The project uses AWS functionality-Lex for creating a chatbot which help's user in creating AWS EC2 and Openstack instances. The chatbot also provides assistance with other cloud functionalities such as listing user instances, checking the instance metrics and its health status. In addition to this, Openstack functionalities such as accessing the remote connection is also available to the user by simple interaction with the chatbot. The above functionalities can also be accessed through Human-voice interaction. Here we have used AWS services such as Cognito, EC2, RDS, API Gateway, Lambda, S3 and Cloudwatch. Openstack services which we have used are Nova Compute, KeyStone, Glance and Cinder.

I. Introduction

THIS report is a description of the various functionalities available to the user along with the cloud instance creation through chat interface. The objective chosen was to create an environment where user can create the instance by interacting with the chatbot without going through the configuration steps. Software Development Kit and APIs were developed for making the complete environment automatic. Several AWS services are used to perform different parts of the project which are explained further in the report.

The user needs to sign-up on our application to use all the functionalities. Once the user signs up and then logs into his account, he is authenticated via the AWS Cognito service. After the user is authenticated, he views his dashboard where different tabs like Table (to view his own instances), Chat (to get assistance and in creation of instances) are available to the user.

In order to make use of the chat feature and use several functionalities, the user needs to answer several questions like whether to create an Openstack or AWS instance, what image type and AMI ID is required for instance creation, what security and key-pair is required. Along with the above the user can also view the available instances, know its health status. The user can check whether the instance is running, paused or terminated. If the user creates an Openstack instance, he can also request for a remote connection via the chat bot.

We have used the maximum ability of the AWS Lex and its machine learning feature to train our utterances well. This makes the environment and our application user friendly and convenient to use. We have made use of the AWS Lambda to develop the state model and handled and validated the utterances well. Thus, directing and guiding the user to solve their purpose of using our application. The model incorporates the above functionalities, eventually leading to a user friendly environment.

The workflow of the services will be explained in detail further in the report. The AWS services used are AWS Cognito, IAM Role, EC2, RDS, S3, API Gateway, Lambda, and Lex. The Openstack services used are Horizon, Nova Compute, KeyStone, Glance and Cinder.

II. Architecture

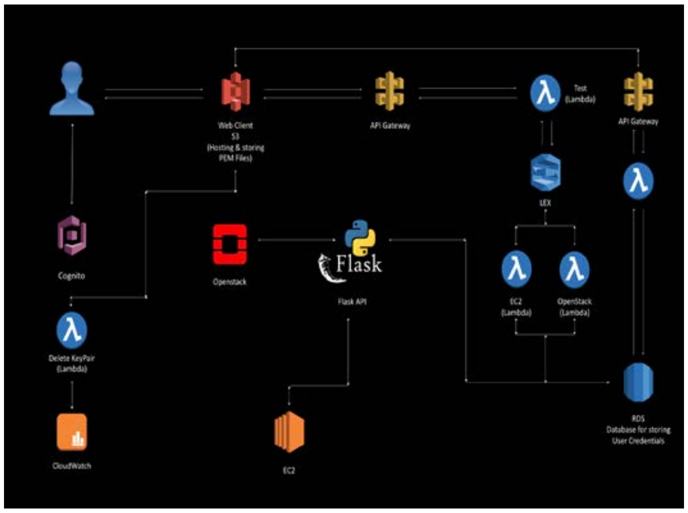


Figure 1 Architectural Block Diagram

We have created one EC2 instance that is treated as our data center. Over this all the user instance would be created either Openstack or EC2 as required by the user for its usage. Based on the input the user provides, the differentiation is done at the intent level in Lex so that corresponding questions will be asked to create the instance.

A. *EC2 Instance Creation*

The user has the facility to create the EC2 instance. The user can type in the required utterances and answer set of follow up questions and finally get his instance created. Thus the user have a whole lot of options as to which AMI ID to use whether Linux, Ubuntu or any other. For prototyping purpose we have provided five options for choosing the AMI ID. Next, the user gets the option for choosing which instance type and what configuration is required to solve the user's purpose. Later, the user gets the option to choose security group type and the key-value pair. Once, all the required parameters are obtained to create a new instance, the API is called which is developed in Flask environment. Thus, programmatically the instance gets created and user is not required to go through any configuration steps of making the instance.

After the call to the API, the user gets a prompt on the chat interface that the machine has been created and presents the user with the machine id. He can also view this machine id on the dashboard. Also, if he has created a new key-pair, the user gets the option to download the .pem key file. For security reasons, the key pair is available for download for 15 minutes. Also, by specifying proper security group id, the authorization is maintained which prevents from any other outside user to access the instance.

Since, the information is specific to the user, Lex uses the user information based on the user id provided at the very beginning. Thus, no other user is allowed to view or access any other users instance.

Similar to instance creation, APIs are also developed to list the instances, to get the health status of the instance, to obtain the running status of the instance. Thus, when the user provides the input for listing the instances, or checking the health status, the chatbot gives the input to the Lambda function. Lambda validates the input and the control is transferred to the EC2 instance where the API is created. After getting the appropriate response from the EC2, it is given back to Lex where the user can see the response from the application what he has asked for. This is how the EC2 instance can be created and different functionalities can be accessed.

B. *Openstack Instance Creation*

Openstack instance creation was the main focus in this Cloud Automation Project. After overcoming several hurdles we faced, we were able to figure out how the Openstack is deployed and how the services can be utilized in a meaningful way. Since, we had limited infrastructure, we developed the idea of Openstack Cloud Automation for prototyping and the ideas can be extended further. We treated the AWS EC2 as our datacenter but in real scenario there will be huge stacks piled up for the datacenter. Over this the Openstack would be downloaded and the services could be made available to the users through chatbot.

TABLE I
OPENSTACK SERVICES

Service	Where is it used?	Description
Keystone	Authentication	This provides identity and authentication for all Openstack services.
Nova Compute	Admin Instance	Handles all process related to instances. Responsible for building the disk image, launching it via the underlying virtualization driver, responding to calls to check its state, attaching persistent storage and terminating it.
Glance	Image Service	This provides the compute image repository. All compute instances launch from glance images.
Cinder	Block Storage	Its designed to present storage resources to end users that can be consumed by Nova compute.

Openstack instance creation was the main focus in this Cloud Automation Project. After overcoming several hurdles we faced, we were able to figure out how the Openstack is deployed and how the services can be utilized in a meaningful way. Since, we had limited infrastructure, we developed the idea of Openstack Cloud Automation for prototyping and the ideas can be extended further. We treated the AWS EC2 as our datacenter but in real scenario there will be huge stacks piled up for the datacenter. Over this the Openstack would be downloaded and the services could be made available to the users through chatbot.

Firstly, we installed Openstack server on an EC2 cloud instance (the main server). This was the biggest milestone we achieved in the project. Later, all the steps were same as used for the EC2 instance creation. Apart from all the functionality we provided for EC2, we wanted to go one step further by providing the remote connection of the instance. After all, there is no use just by creating the instance. The user should be able to access the instance so that some application could be built on that.

To perform any operation related to Openstack instance we had to create an authorization token. Once the token is acquired, we used different Openstack APIs to create instance and access different functionalities. For creating the authorization token we used the Keystone reference API. To get the remote connection of the instance we use the noVNC console to get the access via web.

III. AMAZON WEB SERVICES USED

Cognito AWS is used for the authentication purpose in our application. This protects the user information and hence all the instances created cannot be accessed by outside users. It also helps in authorizing and differentiating between the API call when different users are logged into the account. Thus by the CORS functionality provided by AWS, this can be achieved. This allows to retrieve an

TABLE II
AWS SERVICES USED

Service	Where is it used?	Description
Cognito	Authentication	Authentication of users, authorizing the API, provide user description by tokens
EC2	Admin Instance	Created the instance to treat this as a data center. This is used for saving all the user instances.
S3	Storage	Store app's front end code. Also, user key-value pair is stored for 15 minutes after they create instance using new key pair.
API Gateway	APIs	Contains API for different functionalities
Lambda	Functions	This is the crux where all the communication happens, validation is done and the response is returned to appropriate AWS service. Here the LEX, RDS, EC2 are integrated.
IAM	roles	To provide roles to API and lambda functions for access to AWS services
RDS	Database	List of user specific instances are stored in the RDS Database

access token which is further used to get the user information who has logged into the system and hence use the user's information to provide services such as instance list, health status, remote connection from the application. The authentication of the users is done by building up a user pool which will contain the information about the users who log into the system. They are authorized with the help of federated identity part of Cognito.

Next we have used AWS EC2 service for which we have created an Ubuntu AMI ID with t2.micro as the image type. Flask is used as a server and deployed on the EC2 instance. The programming is done in the python and the host runs on port 5001. Port 5001 is configured in the security group for that instance, so that the application can be accessed by the lambda function. Also the key-pair of that instance is secured store, as that is the data center on which instances are created.

Another service which is used in the implementation of the web application is S3. Amazon S3 is a storage service to help users keep different types of data on cloud. Apart from the UI code, users key-pair for the instance is also stored in the S3. The key-pair file name is stored appending with the user id. Thus there will be no conflict of names if different users want to specify the same key-pair file name. This key-pair is available to the user for only 15 minutes after the creation time and user needs to download the file in that time period only. If the user does not download, the file will deleted. This is done for security of the user instances. In our web application, S3 is used to store multiple types of data which includes the whole front end of the application. This includes all the HTML, CSS and JavaScript files used in the process.

API Gateway is an AWS service which is used to perform the functionalities of calling through multiple APIs. Thus appropriate Lambda function can be called via the API Gateway. These APIs are further used to perform different responsibilities like providing the Lex chatbot as a user interface, interacting with the RDS Database, storing the files in S3 as the object file.

Lambda service is the core of all the functionalities providing responses to all the other services being used. We have written functions in Python 3.6 in Lambda which are performing the functions such as API call, Lex validation, final prompt to Lex, storing the key-pair, accessing the S3, providing data to the frontend through RDS. These lambda functions are given access to many different services of AWS so that they can perform the functionalities efficiently and with ease.

IAM is another AWS service used by the lambda functions to gain access to the AWS services for performing the necessary operations. In order to perform the operations by the lambda functions, they need to have access to the service before making the changes in that service. All the logs are visible in the CloudWatch service of AWS. Hence IAM roles steps in to provide these lambda functions the necessary access to the services to resume the functionality smoothly.

IV. WORKFLOW

First the user logs into the system with the help of a URL which is available on the S3 bucket. The URL is available after the Cognito federated users are made. We have converted the long URL into the short URL. After reaching the home page of the application, the user sees the Dashboard so that user can access various Cloud Functionalities. Currently, Chat Interface and Table of Instances are available. Additionally, user could be provided with the billing status, access the services API, log functionality, network configuration, snapshot creation, subnet configuration and much more.

The user can start interacting with the chatbot. As and when the user gives the input, Lex returns the appropriate response. Let's say user types "I want remote connection". This input will first hit the API gateway which will be passed to the Lambda function where the SDK of Lex is used. Based on the SDK and the configuration of the intents and slots done in the Lex, it takes the user input. The input is validated against the Lambda function and then based on the intent and slot value, the response is returned back to the user on the chat UI.

The response is generated after making the API call that is deployed on the EC2. Also necessary updating is done in the RDS database if that particular functionality demands. For example, after the user hits "I want remote connection", the user gets the response from the RDS database as to for which particular instance the user wants to take the instance of. Once the user provides the input, the above flow is repeated and the instance-id as the parameter is given to the API to get the URL of the remote of that particular instance.

For all the issues in the above flow, it was managed and handled by the logs getting generated in the CloudWatch AWS service. By using the log service, the flow execution became simple and all type of error validation was handled.

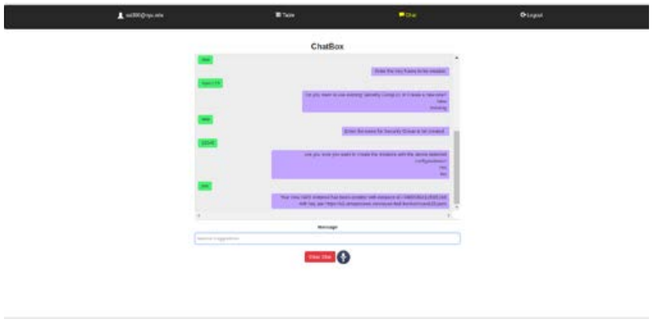


Figure 2 Create EC2 Instance

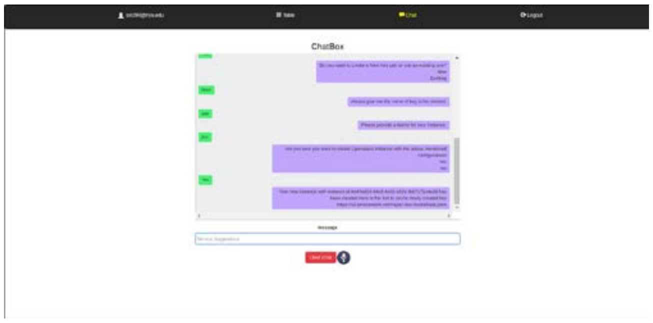
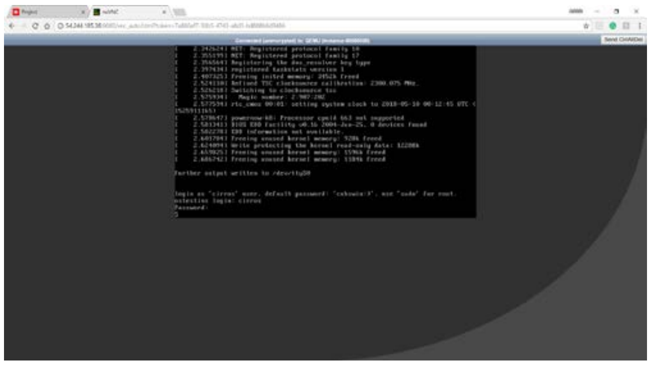


Figure 3 Create Openstack Instance



```
[    2.342624] NET: Registered protocol family 10
[    2.355199] NET: Registered protocol family 17
[    2.356564] Registering the dns_resolver key type
[    2.397434] registered taskstats version 1
[    2.407325] Freeing initrd memory: 2452k freed
[    2.524110] Refined TSC clocksource calibration: 2300.075 MHz.
[    2.526218] Switching to clocksource tsc
[    2.575934]   Magic number: 2:987:282
[    2.577594] rtc_cmos 00:01: setting system clock to 2018-05-10 00:12:45 UTC (
1525911165)
[    2.578647] powernow-k8: Processor cpuid 663 not supported
[    2.581341] BIOS EDD facility v0.16 2004-Jun-25, 0 devices found
[    2.582278] EDD information not available.
[    2.601794] Freeing unused kernel memory: 928k freed
[    2.624094] Write protecting the kernel read-only data: 12288k
[    2.653025] Freeing unused kernel memory: 1596k freed
[    2.686742] Freeing unused kernel memory: 1384k freed

Further output written to /dev/ttyS0


login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
cirros login: cirros
Password:
$
```

Figure 4 Remote Connection

V. Future Scope

For now we have given functionalities such as Create EC2 instance, List Instances, terminate Instance, Get Instance metrics and Check the health status of Instances. There are a range of other functionalities which can be implemented on EC2 instance using a chatbot without having to login and manually perform those activities. They are as follows : a) Actions to be performed on EC2 instance such as Run, Start, Stop the instance. These functionalities can be provided to the user via chatbot with reference to Instance Id and the action to be performed. There is a scope of user getting Details for its instances such as its IP Address, AMI ID, Network Interfaces etc.

b) There is a possibility of user able to reset Instance Attributes via a chatbot. We can implement a functionality where user will be able to take snapshots of its instance using a chatbot. There is a possibility of user able to connect its instance via chatbot remotely.

There are range of other services such as Get Console Output, Get Console Screenshot, Get Password Data, ModifyInstanceCreditSpecification, MonitorInstances, DisassociateIamInstanceProfile etc. All these range of EC2 functionalities can be implemented via chatbot where the user wont be required to login and perform all the actions manually.

For Openstack, the major Future scope can be a possibility of deploying the openstack instance on EC2 automatically using bash script but with the help of a chatbot and when the chatbot installs the openstack instance on an EC2 instance, we can perform different functionalities in an Openstack Instance which are already mentioned in the report above. Other than that, we can give the option of creating a snapshot for Openstack instances of the user via chatbot. There are also range of other actions such as Pause and Unpause the Instance of Openstack.

There is a possibility of implementing a functionality where the user can get the complete details about their openstack instances such as their Public IP Address, Private IP Address, KeyPair which may or may not be used etc. All these functionalities and range of other services can be implemented for Openstack Instances and EC2 instances using ChatBot functionalities.

References

Cognito Developer Guide
https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html
S3 Developer Guide
https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html
API Gateway Developer Guide
https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html
Lambda Developer Guide
https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
Lex Developer Guide
https://aws.amazon.com/lex/
Boto3 Python Documentation
https://boto3.readthedocs.io/en/latest/
IAM Developer Guide
https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html
Deploying Flask on EC2
https://www.matthealy.com.au/blog/post/deploying-flask-to-amazon-web-services-ec2/
Developing Flask API
https://pythonspot.com/flask-web-app-with-python/
Creating the authorization token - keystone reference api
https://docs.openstack.org/keystone/latest/api_curl_examples.html
Openstack Developer Guide for Compute API
https://developer.openstack.org/api-ref/compute/
Install Openstack on EC2
https://www.techrepublic.com/article/how-to-install-openstack-on-a-single-ubuntu-server-virtual-machine/

Python Development
https://docs.openstack.org/mitaka/user-guide/common/cli_install_openstack_command_line_clients.html
Actions on Openstack Compute Instance
https://developer.openstack.org/api-ref/compute/
Openstack Server Api Guide
https://developer.openstack.org/api-guide/compute/server_concepts.html
Openstack Compute Reference material
https://developer.openstack.org/api-ref/compute/