JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Cross Account Infrastructure Replication

Badal Davda
Cloud Support Engineer
Amazon Web Services
Bangalore, India

Dr. M Uttara Kumari
Electronics and Communication
R V College of Engineering
Bangalore, India

Anurag Chandra
Electronics and Communication
R.V. College of Engineering
Bangalore, India

***Abstract*— Cloud computing has been identified as one of the most promising technology which provides highly secured, scalable and optimal Information Technology (IT) infrastructure solutions that too at very low costs. Due to usage of cloud at such a large scale, it gives rise to a wide range of use cases. One use case which has become quite popular in recent times is the migration of the cloud resources between two Amazon Web Services (AWS) accounts. To migrate the infrastructure from one account to another account, the organizations using AWS cloud have either explicit teams, trained to take care of the cloud services or employees from other teams are responsible for the same-which are responsible for migration of the resources. The transfer is done manually and it is a cumbersome and time taking task for the teams. This paper proposes a method to automate the process of migration It gets the resources onboard to a level such that they can directly deliver the results in a very less time and less human effort and hence help in improving customers' experience.**

Keywords: Dashboarding, automation, forecasting

I. INTRODUCTION

In simple terms, cloud is a large network of connected computers [1]. In Fig 1, there are various kinds of connected devices which are collectively forming a cloud.

Major cloud service providers provide a huge range of distinct resources which help the users in meeting their needs. Now, considering the case of a service provider- providing services on web or dedicated application. These web-based and dedicated applications are hosted by resources available in cloud. When the application is bought by another party, the ownership of the applications should also be transferred to the buying side. Currently, transferring the ownership is a cumbersome task as all the resources are replicated or created manually in the destination account with the same configuration as in the source account. This project pioneers to automate this process of ownership transfer.

Fig 1. Cloud as a network of connected devices

II. WORK DONE

The automation has been a long-sought requirement in cloud services. This acts as a catalyst to motivate the users, as it helps them to concentrate more on their development rather than engaging in infrastructure management. This has resulted in a number of research works done around this field.

A container-based virtualization that uses Docker for container packaging and segregating them with Kubernetes for multi-host Docker container management has been the main focus in [2]. In the container-based environment, Kubernetes dynamically monitors the resource requirements and/or usage of the running applications. Based upon the results, adjustment of resource provisioning is managed. This paper further aims at developing a generic platform to facilitate dynamic resource-provisioning based on Kubernetes. Our platform contains the following three features.

Recently, there has been a shift in applications' architecture from standalone to microservices. In microservices style of architecture, small and loosely coupled modules are developed and deployed independently to compose an application. This

architectural style brings various benefits such as modular based easy maintainability and higher flexibility in scaling and aims at far lesser downtime in case of failure or upgrade. The literature [2] examines the availability achievable through Kubernetes under its default configuration. We have conducted a set of experiments which show that the service outage can be significantly higher than expected.

The organizations dependent on cloud faces problems like heterogeneous nature of pricing policies-not standard, yet govern by few companies which gives rise to lock-in problems and another problem is of the possibility to scale from 10s to 1000s of machines implies huge effort in IT management without smart automation systems. The paper [3] tries to explore some of the tools available that may help companies simplifying their application release process.

Users of the cloud determine their own rules which governs the provisioning or deprovisioning actions. These rules are mainly based on the monitoring of certain variables and their threshold set by the users. However, there is always a tradeoff between performance and cost of the resources. In [4], a novel analytical model is proposed that enable the study of application performance under different scaling rules. Based on these, algorithms are developed for performing scaling.

The literature [5] discusses about the problems associated with automated addition and removal of resources. This is called as autoscaling. The problem lies in meeting the agreed service level agreements. It proposes a machine learning algorithm which optimizes the requirements by a service.

III. MOTIVATION

There is long list of advantages which fairly vouch for the cloud services. Moreover, considering the market's response and cloud adaptation by the businesses further consolidates this fact. Some notable features of cloud services by the virtue of which, enterprises are shifting their entire infrastructure on cloud are-less cost, more security, less time for configurations and dependencies, automation, scalability, user friendly, maintenance-free, hassle-free migration are few among other benefits [6].

Though there has been a good development in automation of in cloud services. It offers various focal points for information and programming sharing, media sharing and accordingly making the administration of complex IT frameworks a lot easier [7]. But, with the increased trend of cloud computing-though, the businesses and organizations grew very faster-yet it is not sufficient as the customers have a wide variety of needs. Especially, the work done for the migration of resources from one platform to another or from one account to another is done manually.

It is a prime focus of this project to automate this task of migration of infrastructure between two different AWS account.

IV. SERVICES INVOLVED

Amazon Web Services provides approximately more than 150 distinct services currently. They are all categorized in a number of classes like-compute, storage, database, networking and content delivery, migration tools, mobile services, developer tools, management tools, business tools, analytics, machine learning are few among other categories. This project involves following fundamental services:

1. **Identity and Access Management (IAM):** Sharing of resources across different accounts is done using IAM. It uses Roles for which access is defined by Policies. The source account-where the resource belong, is called as trusting group, given the fact that this account trusts the other account which is accessing the resource from the first group. The other account which accesses the resource from the trusting account is called as trusted account.
2. **Virtual Private Cloud (VPC):** Amazon Virtual Private Cloud (Amazon VPC) provides an isolated area from the rest of the internet, to the users. It is a virtual network created and controlled by the users and can be used exactly as an on-premises private network. In simpler terms, it is an isolated region within cloud containing its own set of resources.
3. **Elastic Load Balancer (ELB):** Elastic Load Balancers automatically distributes internet traffic across multiple servers or systems. The targets handling the traffic can be of multiple types-Amazon Elastic Compute Cloud (EC2), Containers, IP addresses or Lambda functions. The traffic coming from the internet finds single node and this node is called as ELB. The ELB then divides the traffic to different nodes. The traffic distribution is usually controlled by based upon the policies defined in the ELB.
4. **Autoscaling Group (ASG):** ASG service keeps monitoring the users' applications and automatically adjusts the total capacity to maintain steady, seamless and predictable performance at the minimum possible charge. It ensures the maximum availability of the applications by defining some rules which has threshold for metrices like-bandwidth and CPU utilization and adds or removes the resources automatically based upon the rules defined. These rules are also called as Auto scaling policies.
5. **Elastic Compute Cloud (EC2):** The Elastic Compute Cloud (Amazon EC2) service by Amazon Web Services (AWS), provides scalable computing capacity in the AWS cloud. It provides a complete computing virtual environment and remote access facility-Secure Shell (SSH). This service is implemented using type-1 hypervisor on bare metal hardware and the virtual systems are called as Hardware Virtual Machine (HVM).
6. **Simple Storage Service (S3):** S3 is a storage service provided by Amazon Web Services (AWS). It is an object-based storage service that offers potentially scalable at corporate level, high data availability, security, and performance. This provides a very granular level of access control. This also means that the customers of all types and magnitude and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics.

V. DESIGN AND ARCHITECTURE

Fig 2 exhibits the complete architecture used in the project. The project uses AWS Software Development Kit (SDK), which is basically a python library and popularly known as Boto3. It allows Python developers to write software that makes use of Amazon services like S3 and EC2. Boto3 provides an easy to use, object-oriented API as well as low-level direct service access.

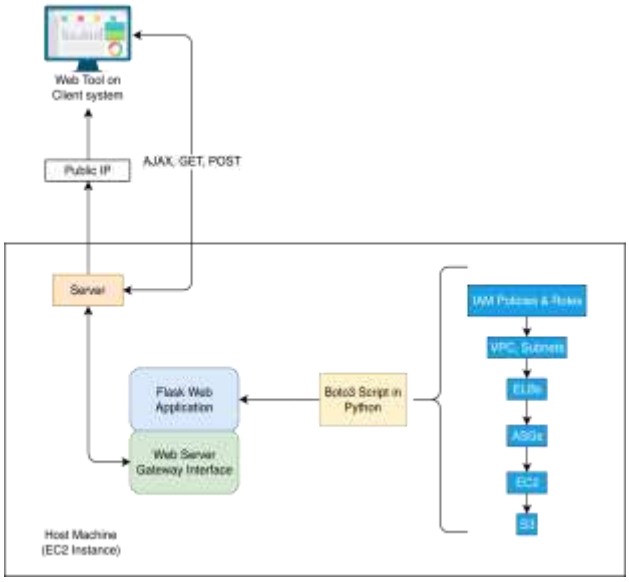

Fig 2. Architecture of the project

It uses Flask as a micro web framework for web applications in python. The codes for front end web page are mainly HTML and CSS files-written using Flask. These files communicate with the web server, which is written in python, using Web Server Gateway Interface (WSGI). As shown in Fig 3, WSGI-used as APIs-is essentially a tool-a set of rules-which implements functions like request, response and any other utility functions.

Fig 3. WSGI APIs used for communication

Flask uses jinja2 which is a popular templating engine for python. A templating engine is used to combine a web template with a certain data source to render a dynamic web page. In this project, DynamoDB is used. It gives the list of all the key-value pairs. This list is used by the ginger and it renders it to the front end of web page dynamically.

All the application files along with all the dependencies of frameworks are bundled-up as a python package. For bundling-up, a containerizing tool is used. The containerizing tool used in this project is Docker which is an open source project.

Fig 4. Containerizing Application using ECR and launching the containers

In Fig 4, the container thus obtained is stored in Amazon Elastic Container Registry (ECR). It can be directly pulled from anywhere over the internet using the repository's Universal Resource Locator (URL).

This image of the container can be used to launch as many containers as required at the same time. So, this makes the application highly scalable, greatly available, free from any dependency and operating system or any underlying hardware.

VI. IMPLEMENTATION

Cross Account Access Using IAM Roles

This is the beginning step of the project's implementation. Before the actual migration starts to happen, the source account needs the access of destination account. As it is evident from Fig 5, it involves three steps:

1. **Create a Role:** This is created in destination account-where resources are to be created. It specifies about who can access the destination account and defines the level of freedom that is available for the account assuming the role-source account in this case. Creation of role also involves attaching trusted policy. The trusted policy trusts the source account by including the source account's identity number.
2. **Grant Access to the Role:** As the name goes, this step is used to give permission to the role created in the first step. And as the role have trusted the source account, this step indirectly gives permissions to the source account. For security reasons, this step can only be done by using inline policies. Inline policies, though they are user managed, can be used only with one entity.
3. **Switching Role:** For switching the role in the destination account, there must be an IAM user in the source account which can be used to switch the role in the destination account.

Fig 5. Cross Account Access Procedure

Virtual Private Cloud (VPC) Migration

Most of the resources are contained within a VPC, Security Groups and further Subnets. So, these are prerequisites for the subsequent resources like-EC2, ELB and ASG. Every time a resource is created successfully, its Amazon Resource Name (ARN)/Id number/name is stored Fig 6 exhibits that the process of VPC migration starts with describing all the VPCs in the source account. All the configuration of the described resources is then used as key to get the mapping of corresponding resources in destination account. These values are used as parameters for creation of new resources. After creation of VPCs, Internet Gateways are extracted and replicated. Further, creation of subnets, creation and attachment of route tables to the VPCs is done.

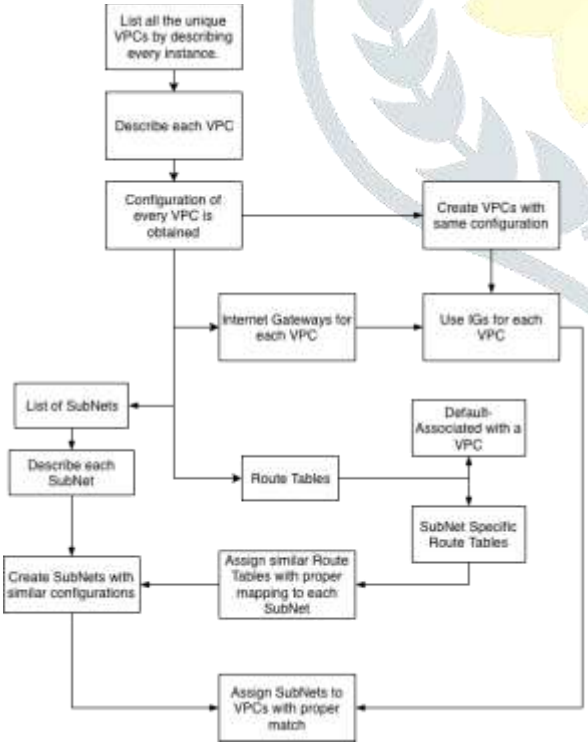

Fig 6. Virtual Private Cloud Migration Flow

Elastic Load Balancer (ELB) Migration

ELBs migration, as shown in the logic diagram of Fig 7, starts with describing all the ELBs in the source account. In the next step, the program finds out the list of all the listeners-a subprocess which continuously checks for connection requests incoming on a configured port used in front end. These listeners are then created in the destination account. Afterwards, target

groups-group of servers handling a particular type of requests-are described in the source account and created in the destination account.

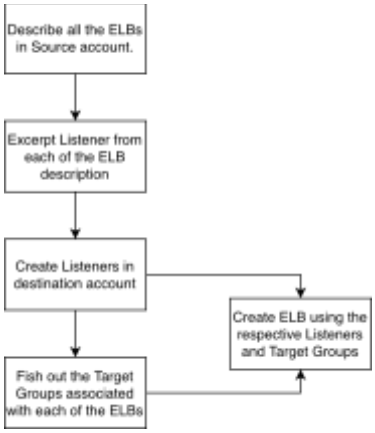

Fig 7. Elastic Load Balancer Migration Flow

After these two perquisites are ready, the program can start with creation of ELBs in the destination account.

Autoscaling Group (ASG) Migration

As portrayed in Fig 8, the migration of ASG starts with creating Autoscaling client in source as well as destination account. The clients are created from source and destination sessions.

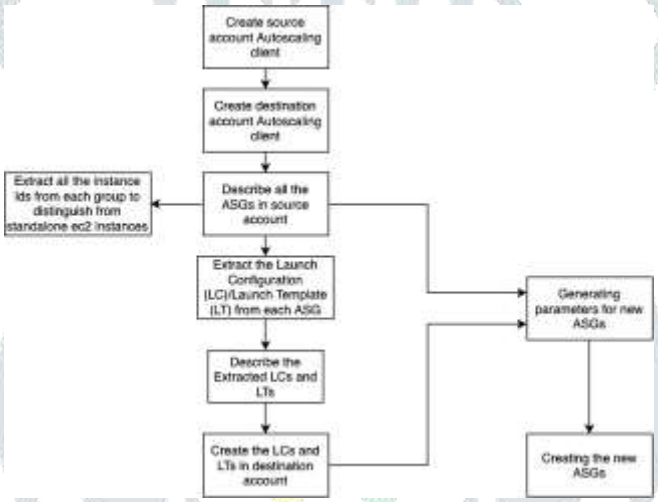

Fig 8. Autoscaling Group Migration Flow

Source as well as destination sessions are created as a result of switching of Identity Access Management (IAM) role in source account by an IAM user in the destination account. After successful creation of clients, source client is used in the source account to describe all the existing ASGs in the source account.

From ASG description, there are two things which are extracted-Launch Configuration/Template list and EC2 instance id list. Instance id list is used to avoid duplication of any EC2 instance which are standalone, i.e., not associated to any ASG. Launch Configuration list is used to describe all the Launch configurations. After descriptions, the program gets all the parameters onboard before calling the APIs for creation. The client is then switched to destination client and all the Launch Configurations are created first. With having launch configuration ready, the program uses these launch configurations in the destination account to create ASGs in the destination account.

Elastic Compute Cloud (EC2) Migration

This section talks about the migration of EC2 resources. Fig 9 explains the logic flow implemented to migrate EC2 resources. The migration starts with generation of source and destination clients. After this, all the instances-including the ones associated with ASGs-are described. From the description response, the instance ids are extracted. But as the ASG module is already creating EC2 instances associated with ASG, they must not be created again. To ensure that, the program removes the instances from them instance list which are part of the ASG using the instance list sent by ASG module earlier. After retaining, only the standalone instances, the system image ids of those instances are fished out from the instances' description. The system ids thus fetched are copied to the destination account. From the copied images, EC2 instances are launched with ASG, they must not be created again. To ensure that, the program removes the instances from them instance list which are part of the ASG using the instance list sent by ASG module earlier. After retaining only, the standalone instances, the system image ids of those instances are fished out from the instances' description. The system ids thus fetched are copied to the destination account. From the copied images, EC2 instances are launched.

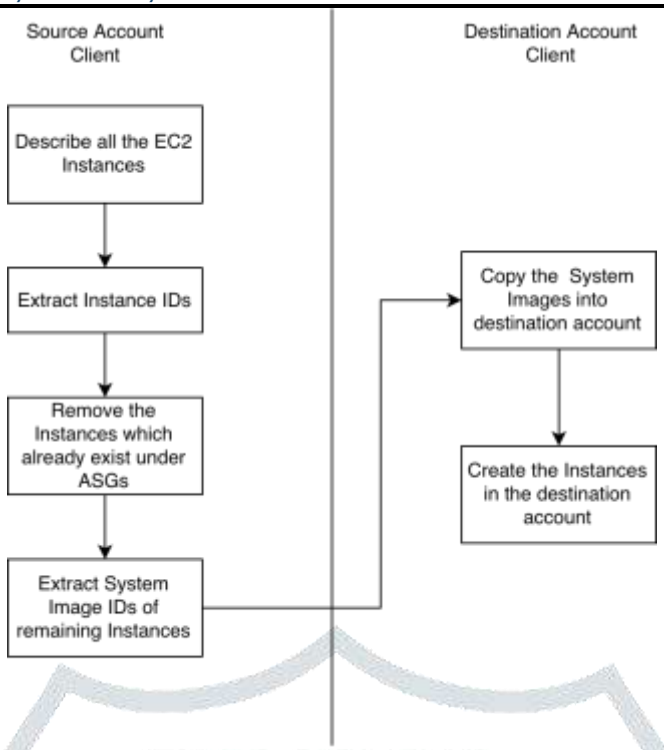

Fig 9. Elastic Compute Cloud Migration Flow

Simple Storage Service (S3) Migration

The Fig 10 explains the flow logic of the S3 resource migration. It is a region-specific service. So, the web tool first asks the users about their preferred regions. Then all the buckets are listed which are present in the source account.

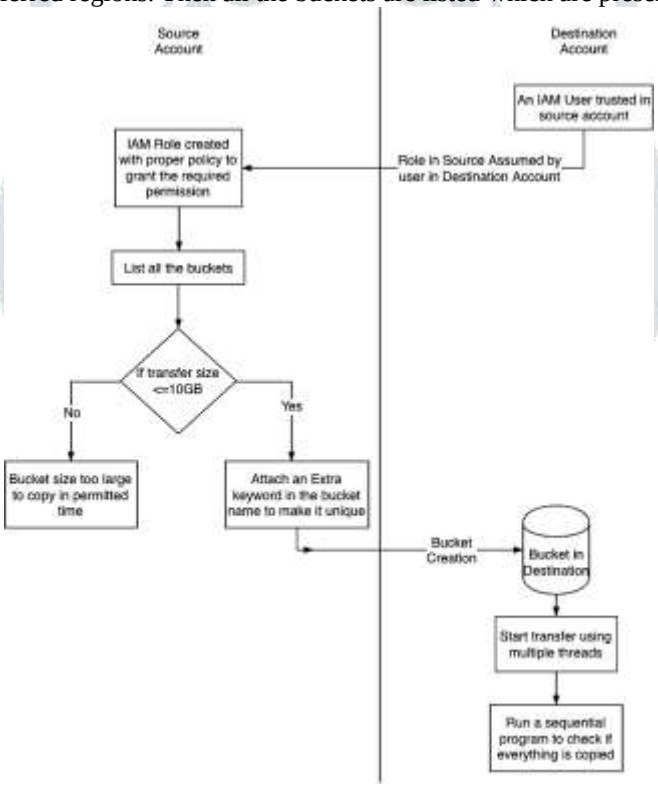

Fig 10. Simple Storage Service Migration Flow

Then all the listed buckets are described in a loop. After description, extraction of relevant configurations information is performed. The command logic then checks if the total size of a bucket is less than 10GB. Migration is allowed only if the total bucket size is less than 10GB-due to time constraints.

A bucket is created in the destination account with a name having a random number as a suffix, added to the bucket name in the source account. This makes sure that the bucket name complies with fact that S3 bucket names have universal namespace.

VII. RESULTS

This section gives details about the results so achieved after finishing the project. It successfully automates the process of automation of the fundamental AWS resources-IAM policies, roles and users, ELBs, ASGs, EC2 instances and S3 storage.

Web Interface

As the EC2 instance with user data is launched, it produces a public IP address. As this EC2 instance has a flask which runs over it, and flask in-turn hosts the web pages. So, if that public IP address receives an HTTP request-using a web browser

in this case-the web page is seen on the browser. In Fig 11, it can be seen that Firefox is the web browser and the IP address along with port number is-127.0.0.1:5000. The web page in Fig 11 is the first web page which has a short explanation about the tool, instruction to use the tool and tab boxes for different services.

Fig 11. First Web Page

Terminal Output

The terminal output starts after the execution of flask running command, as shown in Fig12. It is done after entering the credential details on the credentials page on the web. The flask the starts executing of the python script. As the Python script runs, there are outputs which are printed on the terminal. This is done to understand what happens as the code runs. All the python modules are employed with exception handling. Even then, in case, if there is an exception which occurs while the program runs, the terminal output makes it easier to trace the source of error.

Finally, the Fig 13 shows a progress web page. This page is mainly intended to keep the process more interactive while the program runs. This web page has a separate section for each different service. The sections have different colour signifying different state of the services.

When a service is in queue and its module has not even started execution, then its section is in red colour. When the module is in execution state, its colour is yellow. Finally if the module has finished its execution, it turns to green colour with a Successful message.

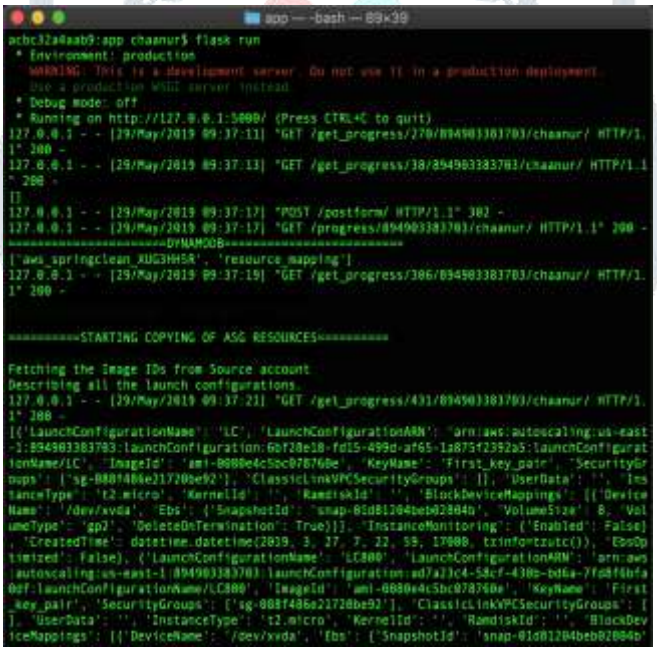

Fig 12. Terminal Output Showing the Progress

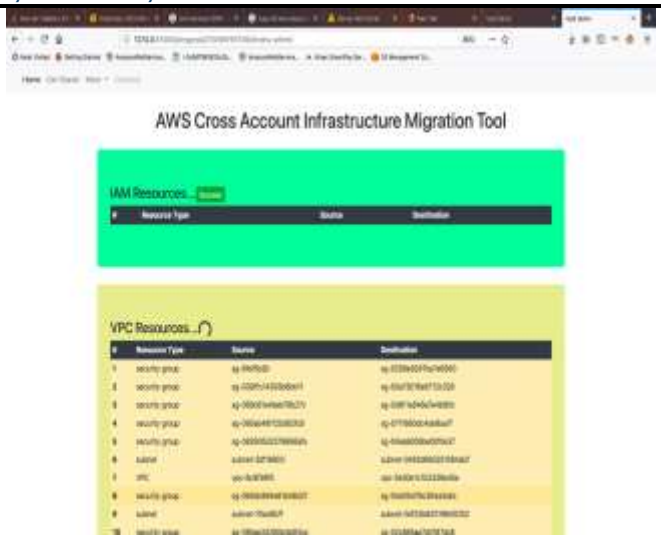

Fig 13. Progress of Web Page

VIII. CONCLUSION

The cross-account infrastructure migration tool automates most of the processes in the course of migrating the resources between two different AWS accounts. It mainly involves six fundamental services-Identity and Access Management (IAM) policies and roles, Virtual Private Cloud (VPC), Load Balancers, Autoscaling Groups (ASG), Simple Storage Service (S3) and Elastic Compute Cloud (EC2).

So, this tool helps the users by minimizing their effort and further it helps in decreasing the human errors and that too in a very less span of time. Hence, these benefits improve the customer experience and helps in consolidation of the company's position in the market.

REFERENCES

[1] Chang, Chia-Chen, Shun-Ren Yang, En-Hau Yeh, Phone Lin, and Jeu-Yih Jeng. "A kubernetes-based monitoring platform for dynamic cloud resource provisioning." In the proceedings of GLOBECOM 2017-2017 IEEE Global Communications Conference, pp. 1-6. IEEE, 2017.

[2] Vayghan, Leila Abdollahi, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. "Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned." In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 970-973. IEEE, 2018.

[3] Miglierina, Marco. "Application deployment and management in the cloud." In 2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 422-428. IEEE, 2014.

[4] Suleiman, Basem, and Srikumar Venugopal. "Modeling performance of elasticity rules for cloud-based applications." In 2013 17th IEEE International Enterprise Distributed Object Computing Conference, pp. 201-206. IEEE, 2013.

[5] Arabnejad, Hamid, Pooyan Jamshidi, Giovani Estrada, Nabil El Ioini, and Claus Pahl. "An auto-scaling cloud controller using fuzzy q-learning-implementation in openstack." In European Conference on Service-Oriented and Cloud Computing, pp. 152-167. Springer, Cham, 2016.

[6] Al-Haidari, Fahd, M. Sqalli, and Khaled Salah. "Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources." In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, vol. 2, pp. 256-261. IEEE, 2013.

[7] Ahmed E, Gani A, Khan MK, Buyya R, Khan SU. Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. Journal of Network and Computer Applications. 2015 Jun 1:52:15