

Automation of Infrastructure using AWS

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering

By

Name: BODDU MOHAN HARI

KRISHNA (**Reg. no.** 37110101)

&

Name: BOLISETTY RAJ KRISHNA

(**Reg. no.** 37110104)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119, TAMILNADU

March 2021



SATHYABAMA



INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabamauniversity.ac.in

DEPARTEMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **B. MOHAN HARI KRISHNA (37110101)** and **B. RAJ KRISHNA (37110104)** who carried out the project entitled "**Automation of Infrastructure using AWS**" under my supervision from November 2020 to March 2021.

Internal Guide

Name: Mrs. A. C. SHANTHA SHEELA, M.E., (Ph.D).,

HEAD OF DEPARTEMENT

Dr. S. VIGNESHWARI, M.E., Ph.D.

Dr. L. Lakshmanan, M.E., Ph.D.

Submitted for Viva Voice Examination held on _____

Internal Examiner

External Examiner

DECLARATION

BODDU MOHAN HARI KRISHNA (Reg.No.37110101) and BOLISETTY RAJ KRISHNA (Reg. No.37110104) hereby declare that the Project Report entitled “**Cloud Computing**” done by me under the guidance of **Mrs. A. C. SHANTHA SHEELA, M.E., (Ph.D)., Dr.S. VIGNESHWARI, M.E., Ph.D.,** and **Dr. L. Lakshmanan, M.E., Ph.D.** at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

NAME OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing , **Dr.S.Vigneshwari M.E., Ph.D., and Dr.L.Lakshmanan M.E., Ph.D.**, Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Mrs. A. C. SHANTHA SHEELA, M.E., (Ph.D).**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE



MOHAN HARI KRISHNA BODDU

has successfully completed the AWS Certification
requirements and has achieved their:

AWS Certified Solutions Architect - Associate

Issue Date
Mar 05, 2021

Expiration Date
Mar 05, 2024

A handwritten signature in black ink, appearing to read "Maureen Lonergan".

Maureen Lonergan
Director, Training and Certification

Validation Number QYFDVJ2D2JV1QYCM
Validate at: <http://aws.amazon.com/verification>



NPTEL Online Certification

(Funded by the Ministry of HRD, Govt. of India)



This certificate is awarded to

RAJ KRISHNA

for successfully completing the course

Programming, Data Structures and Algorithms Using Python

with a consolidated score of **52** %

Online Assignments	19.88/25	Programming Exam	12.5/25	Proctored Exam	20/50
--------------------	----------	------------------	---------	----------------	-------

Total number of candidates certified in this course: **2779**

Devendra Jalihal

Prof. Devendra Jalihal
Chairman
Centre for Continuing Education, IITM

Jul-Sep 2019
(8 week course)

Prof. Andrew Thangaraj

Prof. Andrew Thangaraj
NPTEL Coordinator
IIT Madras



Indian Institute of Technology Madras



Roll No: NPTEL19CS40S21183298

To validate and check scores: <https://nptel.ac.in/noc>

ABSTRACT

Today in this present era where technological advances are at its vertex, everything is getting automated. In the process of Application Development there exists a need for provisioning of compute, storage and networking resources for employees of a company for their daily responsibilities. For such scenarios employees acquire compute and storage resources from a third person who handles such issues, might be a System Admin, Devops Engineer or a Cloud Engineer. But it involves for checking the policies of resources, taking care of governance, complying with the rules to meet the industry standard, If a new employee takes incharge of such positions it's difficult to cope with all these things quickly. So we came up with an idea to automate all these Infrastructure Deployment(IaaS) through WhatsApp using Amazon Web Services(AWS). This automation eliminates the manual and complex tasks involved in the deployment activities with far more easier and user friendly manner which brings more agility in terms of productivity in Technology.

TABLE OF CONTENTS:

Chapter No.	Title.	Page No.
	Abstract	v
	List of Figures	i
	List of Abbreviations	x
		x
		i
1.	INTRODUCTION	1
	1.1. Existing System	1
	1.2. Purpose of the Project	2
	1.3. Proposed System	2
2.	LITERATURE SURVEY	3
3.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	
	3.1. Aim of the Project	6
	3.2. Scope and Objectives of the Project	6
	3.3. Functional and Non-Functional Requirements	6
	3.3.1 Functional Requirements	6
	3.3.2. Non-Functional Requirements	7
4.	SYSTEM DESIGN AND METHADODOLOGY	9
	4.1. Hardware Requirements	9
	4.2. Software Requirements	9
	4.3. Introduction of Tools and Softwares used	9

4.3.1. Cloud Computing	9
4.3.2. Python	17

	4.3.3. Heroku	18
	4.3.4. Gunicorn	20
	4.3.5. Flask	22
	4.3.6. AWS Boto3	24
	4.4. Implementation	25
5.	RESULTS AND DISCUSSIONS	28
6.	CONCLUSION AND FUTURE WORK	31
	REFERENCES	32
	APPENDIX	33

LIST OF FIGURES

FIGURE NO.	FIGURE NAME.	PAGE NO.
4.1	Gunicorn	20
4.2	Gunicorn Architecture	21
4.3	Flask	22
4.4	Project Architecture	25
5.1	Instance Request	27
5.2	Software Configuration	28
5.3	Instance Type	29
5.4	IP Address Retrieval	30
5.5	Twilio Sandbox	31
5.6	Twilio Callback Address	31
5.7	Heroku Deployment	32
5.8	Heroku Hosting	32
5.9	Heroku Logs	33

LIST OF ABBREVIATIONS:

ABBREVIATION:

AWS

IAAC

HTTP

API

URL

IP

EXPANSION:

Amazon Web Services

Infrastructure as a Code

Hypertext Transfer Protocol

Application Programming

Interface

Uniform Resource Locator

Internet Protocol

CHAPTER – 1:

1.1. INTRODUCTION:

A person who required to complete some technical work either simple tasks or complex tasks they require a Computer, Storage, RAM, Processors etc., This requires a lot of money, research and Time to get the things done perfectly. He may choose or depend on Internet centres to complete the work but it also may require some efforts. For a Student if he want to do some College Projects he may need a Machine of High End Configurations which may require more money to buy a laptop or a Computer with that configurations. Buying a computer with high-end configurations is not possible for all the people as they cannot afford that much money for some tasks. For employees in the technical field, getting the work done easier and faster in a perfect manner is the primary goal to deliver the project, So to achieve the goal they need a System with the requisite RAM, Memory, Processor and Software installed to complete the work in a fast manner. If we automate this process and provide all the prerequisites required for the project to be done, then employee can work on it directly and easily without any headache of installing Software and arrangements.

1.1. EXISTING SYSTEM:

In the Industrial use, generally if any of the employee require a Virtual Machine, For such scenarios employees acquire compute and storage resources from a third person who handles such issues, might be a System Admin, Devops Engineer or a Cloud Engineer. But it involves for checking the policies of resources, taking care of governance, complying with the rules to meet the industry standard, If a new employee takes in charge of such positions it's difficult to cope with all these things quickly, then System Administrator manually make a Machine Image and give that IP Address or the URL to the employee. This is a long and manual process which requires some time and effort to get things done. Even for the normal people for daily needs they need to either go to Internet centre or depend on others to complete that

work or an Assignment. Buying a new Computer with the desired Configuration is not a possible or affordable thing for all the people which is a Costliest one to do for small works.

1.2. PURPOSE OF PROJECT:

In this competitive world everything needs to be faster and in an easier manner, so to make the work easier and cost efficient we automated the process of Infrastructure Deployment simply using AWS Cloud. This reduces the Cost and Time consumption for the company and get the things done easily and in a simple manner just by some easy WhatsApp Chat. Even this can be implemented for normal people as many students can use this methodology to complete their work simply by requesting for the desired machine in a simple WhatsApp Chat and pay for what they use.

1.3. PROPOSED SYSTEM:

The proposed project provides solutions for the problems stated above in a simple and easy manner. So we came up with an idea to automate all these Infrastructure Deployment(IaaS) through WhatsApp using Amazon Web Services(AWS). This automation eliminates the manual and complex tasks involved in the deployment activities with far more easier and user friendly manner which brings more agility in terms of productivity in Technology.

Developed a WhatsApp bot to get a simple online virtual machine of desired configuration and which can be accessed through a browser by simple chat. Here user can give the requirements of his system he needed, like amount of RAM, Processors and the type of Operating System he want to use.

The user requests are handled in the backend and sends a reply back to the user with an IP Address or URL. This reduces the Cost and eliminates some tedious task of creating and launching instances in Cloud, even a person with zero knowledge on Cloud can get his work done simply by this WhatsApp Bot.

CHAPTER – 2:

LITERATURE SURVEY:

Muhammad Younus etc., conducted a survey on Agile Software Development Using Cloud he proposed that Agile software development is successful due to self-organizing teams, adaptive planning, a cooperative environment with respect to communication with clients and team members, small development cycles, continuous design improvements, continuous delivery and feedback of clients.

Cloud computing helps to reduce cost, enables scalability and enhances communication through its services. A generic framework with the conjunction of Agile Development and Cloud Computing (ADCC) proposed in an earlier study is evaluated in this study. The Malaysia Research and education network (MyRen) cloud is utilized to implement the framework.

A case study is conducted to evaluate the framework. Before conducting the case study, the participants are educated on the ADCC framework. The results of the case study show that the performance of agile methods is improved with the usage of the ADCC framework. The improvement is measured in terms of local and distributed agile development environments.

He proposed and explained the facts that Cloud Computing will play a major role in the development of Software Industry or in the Industrial needs. This helps to reduce cost, enables scalability and enhances communication through its services.

Ali Server has proposed that in distributed agile software development, to conquer the problems raised by rapid development of complex products with ever changing technological advances and risks, there is a need of a model with specific tools and technology supported approaches to manage them. Distributed environment has challenges and complexities in many ways of over collocated teams.

However, they seem to enhance the basic principles in which agile methodology thrives on. Agile projects and distributed projects are compatible and therefore practices, principles, tools, and techniques for distributed agile software development are introduced to the distributed teams to ensure that the agile projects become successful. Although cloud computing is well established for software system developments, its effect on agile methodology has not been extensively studied. This study outlines how agile projects can be modeled in the cloud-computing era, namely, seamless agilecloud integration.

He proposed a model to improve Agile Methodology in distributed computing platform. This model has uncovered a gap between the distributed agile development phases that are overemphasized and the phases that require actual attention. In addition to this, emphasis on appropriate phases would help build desired software that can exploit the benefits of cloud computing in numerous ways.

Dayang N.A.Jawawi has stated that Agile software development is based on a set of values and principles. The twelve principles are inferred from agile values. Agile principles are composition of evolutionary requirement, simple design, continuous delivery, self-organizing team and face-to-face communication. Due to changing market demand, agile methodology faces problems such as scalability, more effort and cost required in setting up hardware and software infrastructure, availability of skilled resource and ability to build application from multiple locations.

Twelve (12) principles may be practiced more appropriately with the support of cloud computing. This merger of agile and cloud computing may provide infrastructure optimization and automation benefits to agile practitioners. They proposed a Systematic Literature Review (SLR) identifies the techniques employed in cloud computing environment that are useful for agile development. In addition, SLR discusses the significance of cloud and its challenges.

The result of SLR shows that the techniques using existing tools were reported in 35%, simulations in 20% and application developed in 15% of the studies. Evaluation of techniques was reported in 32% of the studies. The impact of cloud computing was measured by the classification of four major categories such as transparency 32%, collaboration 50%, development infrastructure 29% and cloud quality attributes in 39%. Furthermore, a large number of tools were reported in primary studies. The challenges posed by cloud adoption in agile was reported as interoperability 13%, security & privacy 18% and rest of the primary studies did not report any other research gaps.

The study concludes that agile development in cloud computing environment is an important area in software engineering. There are many open challenges and gaps. In particular, more quality tools, evaluation research and empirical studies are required in this area.

CHAPTER 3:

AIM AND SCOPE OF PRESENT INVESTIGATION:

3.1 AIM OF THE PROJECT:

The main aim of the project is to build an enhanced Bot which automate the Infrastructure Deployment(IaaS) through WhatsApp using Amazon Web Services(AWS). This automation eliminates the manual and complex tasks involved in the deployment activities with far more easier and user friendly manner which brings more agility in terms of productivity in Technology and also more ease of doing work in Cost efficient way.

3.2 SCOPE AND OBJECTIVES:

This Project traverses a lot of areas ranging from business concept to computing field, and required to perform several researches to be able to achieve the project objectives. The area covers include:

- This application can be used in companies to reduce cost and time.
- Useful for Individual people to reduce efforts and money to complete work in a simple and handy manner.
- Students can get a great benefit to learn many things and to test and deploy some Machine Learning Projects.
- This can be used to test some Machine Learning or Android Projects as they are Faster due to more RAM and Processors.

3.3 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

3.3.1 *Functional Requirements:*

Requirement analysis is a software engineering technique that is composed of the various tasks that determine the needs or conditions that are to be met for a new or altered product, taking into consideration the possible conflicting requirements of the

various users.

Functional requirements are those requirements that are used to illustrate the internal working nature of the system, the description of the system, and explanation of each subsystem. It consists of what task the system should perform, the processes involved, which data should the system holds and the interfaces with the user. The functional requirements identified are:

- This have simple user Interface just making use of WhatsApp which is very easy for everyone to handle.
- Clients can request the Machine Images through WhatsApp.
- The requested Information is processed in the backend and the output IP Address or URL is given to the Client as a reply in WhatsApp to get access to the Machine.
- In the backend we used Heroku, Twilio, Gunicorn, Flask, AWS, Boto3, to procees the requests.

3.3.2 Non-Functional Requirements :

It describes aspects of the system that are concerned with how the system provides the functional requirements. They are:

- Security: The subsystem should provide a high level of security and integrity of the data held by the system, only authorized personnel of the company can gain access to the company's Database.
- Performance and Response time: The system should have high performance rate when executing user's input and should be able to provide response within a short time span usually 1 to 2 seconds for highly

complicated task it may take 2 to 3 seconds.

- Error handling: Error should be considerably minimized and an appropriate error message that guides the user to recover from an error should be provided. Validation of user's input is highly essential.
- Availability: This system should always be available for access at 24 hours, 7 days a week. Also, in the occurrence of any major system malfunctioning, the system should be available in 1 to 2 working days, so that the business process is not severely affected.
- Ease of use: Considered the level of knowledge possessed by the users of this system, a simple but quality user interface should be developed to make it easy to understand and required less training.

CHAPTER – 4:

SYSTEM DESIGN & METHODOLOGY:

System requirements:

4.1 Hardware Requirements:

- 2 GB RAM
- 8 GB Hard Disk
- Desktop / Laptop

4.2 Software Requirements:

- Python
- Flask
- Heroku
- GIT
- NGROK
- VS Code
- AWS
- Twilio

4.3. INTRODUCTION OF TOOLS OR SOFTWARES USED

4.3.1 Cloud Computing:

Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. The term is generally used to describe data centers available to many users over the Internet. Large clouds, predominant today, often have functions distributed over multiple locations from central servers. If the connection to the user is relatively close, it may be designated an edge server.

Clouds may be limited to a single organization (enterprise clouds), or be available to

multiple organizations (public cloud).

Cloud computing relies on sharing of resources to achieve coherence and economies of scale. Advocates of public and hybrid clouds note that Cloud computing allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that Cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand, providing the burst computing capability: high computing power at certain periods of peak demand.

Cloud providers typically use a "pay-as-you-go" model, which can lead to unexpected operating expenses if administrators are not familiarized with cloud-pricing models.

The availability of high-capacity networks, low-cost computers and storage devices as well as the widespread adoption of hardware virtualization, service-oriented architecture and autonomic and utility computing has led to growth in cloud computing. By 2019, Linux was the most widely used operating system, including in Microsoft's offerings and is thus described as dominant.

The objective of cloud computing is to permit clients to take profit by these innovations, without the requirement for profound information about or skill with every single one of them. The cloud plans to reduce expenses and helps the clients center around their center business as opposed to being hindered by IT hindrances. The primary empowering innovation for cloud computing is virtualization. Virtualization programming isolates an actual computing gadget into at least one "virtual" gadgets, every one of which can be handily utilized and figured out how to perform computing errands. With working framework level virtualization basically making a versatile arrangement of different free computing gadgets, inactive computing assets can be assigned and utilized all the more effectively. Virtualization gives the dexterity needed to accelerate IT tasks and decreases cost by expanding foundation use. Autonomic computing mechanizes the cycle through which the client can arrangement assets on- request. By limiting client contribution, robotization speeds up the cycle, lessens work costs and diminishes the chance of human

blunders. Cloud computing uses concepts from utility computing to provide metrics for the services used. Cloud computing

attempts to address QoS (quality of service) and reliability problems of other grid computing models.

Cloud computing shares characteristics with:

- Client-server model—Client-server computing refers broadly to any Distributed application that distinguishes between service providers (servers) and service requestors (clients).
- Computer bureau—A service bureau providing computer services, particularly from the 1960s to 1980s.
- Grid computing - A form of distributed and parallel computing, whereby a 'super and virtual computer' is composed of a cluster of networked, loosely coupled computers acting in concert to perform very large tasks.
- Fog computing - Distributed computing paradigm that provides data, compute, storage and application services closer to the client or near-user edge devices, such as network routers. Furthermore, fog computing handles data at the network level, on smart devices and on the end-user client-side (e.g. mobile devices), instead of sending data to a remote location for processing.
- Mainframe computer - Powerful computers used mainly by large organizations for critical applications, typically bulk data processing such as census; industry and consumer statistics; police and secret intelligence services; enterprise resource planning; and financial transaction processing.
- Utility computing - The "packaging of computing resources, such as computation and storage, as a metered service similar to a traditional public utility, such as electricity".
- Peer-to-peer - A distributed architecture without the need for central coordination. Participants are both suppliers and consumers of resources (in contrast to the traditional client-server model).
- Green computing - Study and practice of environmentally sustainable computing or IT.
- Cloud sandbox - A live, isolated computer environment in which a program, code or file can run without affecting the application in which it runs.

Service models:

Cloud computing service models arranged as layers in a stack:

Though service-oriented architecture advocates "Everything as a service" (with the acronyms EaaS or XaaS, or simply aas), cloud-computing providers offer their "services" according to different models, of which the three standard models per NIST are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These models offer increasing abstraction; they are thus often portrayed as layers in a stack: infrastructure-, platform- and software-as-a-service, but these need not be related. For example, one can provide SaaS implemented on physical machines (bare metal), without using underlying PaaS or IaaS layers, and conversely one can run a program on IaaS and access it directly, without wrapping it as SaaS.

Infrastructure as a service (IaaS):

"Infrastructure as a service" (IaaS) refers to online services that provide high-level APIs used to abstract various low-level details of underlying network infrastructure like physical computing resources, location, data partitioning, scaling, security, backup, etc. A hypervisor runs the virtual machines as guests. Pools of hypervisors within the cloud operational system can support large numbers of virtual machines and the ability to scale services up and down according to customers' varying requirements. Linux containers run in isolated partitions of a single Linux kernel running directly on the physical hardware. Linux C groups and namespaces are the underlying Linux kernel technologies used to isolate, secure and manage the containers. Containerisation offers higher performance than virtualization because there is no hypervisor overhead. Also, container capacity auto-scales dynamically with computing load, which eliminates the problem of over-provisioning and enables usage-based billing. IaaS clouds often offer additional resources such as a virtual-machine disk-image library, raw block storage, file or object storage, firewalls, load balancers, IP addresses, virtual local area networks (VLANs), and software bundles.

The NIST's definition of cloud computing describes IaaS as "where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)."

IaaS-cloud providers supply these resources on-demand from their large pools of equipment installed in data centers. For wide-area connectivity, customers can use either the Internet or carrier clouds (dedicated virtual private networks). To deploy their applications, cloud users install operating-system images and their application software on the cloud infrastructure. In this model, the cloud user patches and maintains the operating systems and the application software. Cloud providers typically bill IaaS services on a utility computing basis: cost reflects the amount of resources allocated and consumed.

Platform as a service (PaaS):

The NIST's definition of cloud computing defines Platform as a Service as:

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

PaaS vendors offer a development environment to application developers. The provider typically develops toolkit and standards for development and channels for distribution and payment. In the PaaS models, cloud providers deliver a computing platform, typically including operating system, programming-language execution

environment, database, and web server. Application developers develop and run their

software on a cloud platform instead of directly buying and managing the underlying hardware and software layers. With some PaaS, the underlying computer and storage resources scale automatically to match application demand so that the cloud user does not have to allocate resources manually.

Some integration and data management providers also use specialized applications of PaaS as delivery models for data. Examples include iPaaS (Integration Platform as a Service) and dPaaS (Data Platform as a Service). iPaaS enables customers to develop, execute and govern integration flows. Under the iPaaS integration model, customers drive the development and deployment of integrations without installing or managing any hardware or middleware. dPaaS delivers integration—and data-management—products as a fully managed service. Under the dPaaS model, the PaaS provider, not the customer, manages the development and execution of programs by building data applications for the customer. dPaaS users access data through data-visualization tools.

Software as a service (SaaS):

The NIST's definition of cloud computing defines Software as a Service as:

The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

In the software as a service (SaaS) model, users gain access to application software and databases. Cloud providers manage the infrastructure and platforms that run the applications. SaaS is sometimes referred to as "on-demand software" and is usually

priced on a pay-per-use basis or using a subscription fee.[83] In the SaaS model, cloud

providers install and operate application software in the cloud and cloud users access the software from cloud clients. Cloud users do not manage the cloud infrastructure and platform where the application runs. This eliminates the need to install and run the application on the cloud user's own computers, which simplifies maintenance and support. Cloud applications differ from other applications in their scalability—which can be achieved by cloning tasks onto multiple virtual machines at run-time to meet changing work demand. Load balancers distribute the work over the set of virtual machines. This process is transparent to the cloud user, who sees only a single access-point. To accommodate a large number of cloud users, cloud applications can be multitenant, meaning that any machine may serve more than one cloud-user organization.

The pricing model for SaaS applications is typically a monthly or yearly flat fee per user, so prices become scalable and adjustable if users are added or removed at any point. It may also be free. Proponents claim that SaaS gives a business the potential to reduce IT operational costs by outsourcing hardware and software maintenance and support to the cloud provider. This enables the business to reallocate IT operations costs away from hardware/software spending and from personnel expenses, towards meeting other goals. In addition, with applications hosted centrally, updates can be released without the need for users to install new software. One drawback of SaaS comes with storing the users' data on the cloud provider's server. As a result,[citation needed] there could be unauthorized access to the data.[87] Examples of applications offered as SaaS are games and productivity software like Google Docs and Word Online. SaaS applications may be integrated with cloud storage or File hosting services, which is the case with Google Docs being integrated with Google Drive and Word Online being integrated with Onedrive.[citation needed]

Mobile "backend" as a service (MBaaS):

In the mobile "backend" as a service (m) model, also known as backend as a service

(BaaS), web app and mobile app developers are provided with a way to link their applications to cloud storage and cloud computing services with application programming interfaces (APIs) exposed to their applications and custom software

development kits (SDKs). Services include user management, push notifications, integration with social networking services and more.

Cloud computing types:

Private cloud

Private cloud is cloud infrastructure operated solely for a single organization, whether managed internally or by a third party, and hosted either internally or externally. Undertaking a private cloud project requires significant engagement to virtualize the business environment, and requires the organization to reevaluate decisions about existing resources. It can improve business, but every step in the project raises security issues that must be addressed to prevent serious vulnerabilities. Self-run data centers are generally capital intensive. They have a significant physical footprint, requiring allocations of space, hardware, and environmental controls. These assets have to be refreshed periodically, resulting in additional capital expenditures. They have attracted criticism because users "still have to buy, build, and manage them" and thus do not benefit from less hands-on management, essentially "the economic model that makes cloud computing such an intriguing concept".

Public cloud:

Cloud services are considered "public" when they are delivered over the public Internet, and they may be offered as a paid subscription, or free of charge.[101] Architecturally, there are few differences between public- and private-cloud services, but security concerns increase substantially when services (applications, storage, and other resources) are shared by multiple customers. Most public-cloud providers offer direct-connection services that allow customers to securely link their legacy data centers to their cloud-resident applications.

Several factors like the functionality of the solutions, cost, integrational and organizational aspects as well as safety & security are influencing the decision of enterprises and organizations to choose a public cloud or on-premise solution.

Hybrid cloud:

Hybrid cloud is a composition of a public cloud and a private environment, such as a private cloud or on-premises resources, that remain distinct entities but are bound together, offering the benefits of multiple deployment models. Hybrid cloud can also mean the ability to connect collocation, managed and/or dedicated services with cloud resources. Gartner defines a hybrid cloud service as a cloud computing service that is composed of some combination of private, public and community cloud services, from different service providers.[106] A hybrid cloud service crosses isolation and provider boundaries so that it can't be simply put in one category of private, public, or community cloud service. It allows one to extend either the capacity or the capability of a cloud service, by aggregation, integration or customization with another cloud service.

4.3.2 PYTHON:

Python is a simple to learn, incredible programming language. It has proficient undeniable level information structures and a basic yet compelling way to deal with object-oriented programming. Python's exquisite syntax and dynamic composing, along with its interpreting nature, make it an ideal language for scripting and fast application advancement in numerous regions on most stages. At the point when you write programs in Python, you never need to make a big deal about low-level subtleties, for example, dealing with the memory utilized by your program.

Regularly, software engineers fall head over heels in love for Python as a result of the expanded efficiency it gives. Since there is no aggregation step, edit-test-debug

is unbelievably quick. Troubleshooting Python programs is simple: a bug or bad input

won't ever cause a Segmentation fault. All things considered, when the mediator finds a blunder, it raises an exemption. At the point when the program doesn't get the exception, the interpreter prints a stack trace. A source level debugger permits examination of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, vouching for Python's contemplative force. Then again, regularly the speediest method to troubleshoot a program is to add a couple of print proclamations to the source: the quick fast edit-test-debug cycle simplifies this methodology viable.

4.3.3 HEROKU:

Heroku is a cloud platform as a service (PaaS) supporting several programming languages. One of the first cloud platforms, Heroku has been in development since June 2007, when it supported only the Ruby programming language, but now supports Java, Node.js, Scala, Clojure, Python, PHP, and Go. For this reason, Heroku is said to be a polyglot platform as it has features for a developer to build, run and scale applications in a similar manner across most languages. Heroku was acquired by Salesforce.com in 2010 for \$212 million.

The Heroku Platform:

The Heroku network runs the customer's apps in virtual containers which execute on a reliable runtime environment. Heroku calls these containers "Dynos". These Dynos can run code written in Node, Ruby, PHP, Go, Scala, Python, Java, or Clojure. Heroku also provides custom buildpacks with which the developer can deploy apps in any other language. Heroku lets the developer scale the app instantly just by either increasing the number of dynos or by changing the type of dyno the app runs in.

Heroku Postgres:

Heroku Postgres is the Cloud Database (DBaaS) service for Heroku based on PostgreSQL. Heroku Postgres provides features like continuous protection, rollback,

and high availability; also forks, followers, and dataclips.

Heroku Redis:

Heroku Redis is the customized [Redis](#) from Heroku to provide a better developer experience. It is fully managed and is provided as a service by Heroku. It helps in managing instances with a CLI, associate data with Postgres to gain business insights using SQL tools, and lets customer gain performance visibility.

Heroku Teams:

Heroku Teams is a team management tool which provides collaboration and controls to bring a customer's developers, processes, and tools together in order to build better software. With Heroku Teams, teams can self-organize, add, and manage members, get fine-grained control with app-level permissions and also use collaboration tools like Heroku Pipelines. It also provides delegated administration and centralized billing.

Heroku Enterprise:

Heroku Enterprise provides services to large companies which help them to improve collaboration among different teams. It provides a set of features like fine-grained access controls, identity federation, and private spaces to manage their enterprise application development process, resources, and users.

Heroku Connect:

Heroku Connect lets users create Heroku apps that can easily integrate with deployments at scale. This is done by having a seamless data synchronization between Heroku Postgres databases and Salesforce organizations.

Heroku Elements:

Heroku Elements provides users with Add-ons (tools and services for developing, extending, and operating the app), Buildpacks (which automate the build processes

for the preferred languages and frameworks) and Buttons (a tool for the one-click provisioning, configuring, and deployment of third party components, libraries and patterns).

4.3.4 GUNICORN:



Fig.no 4.1

Green Unicorn, commonly shortened to "Gunicorn", is a Web Server Gateway Interface (WSGI) server implementation that is commonly used to run Python web applications.

Gunicorn is one of many WSGI server implementations, but it's particularly important because it is a stable, commonly-used part of web app deployments that's powered some of the largest Python-powered web applications in the world, such as Instagram.

Gunicorn implements the PEP3333 WSGI server standard specification so that it can run Python web applications that implement the application interface. For example, if you write a web application with a web framework such as Django, Flask or Bottle, then your application implements the WSGI specification.

Gunicorn is an implementation of the WSGI servers concept. Learn how these pieces fit together in the deployment chapter or view the table of contents for all topics.

Gunicorn knows how to run a web application based on the hook between the WSGI server and the WSGI-compliant web app. Here is an example of a typical Django web application and how it is run by Gunicorn. We'll use the `django_defaults` as an example Django project. Within the `django_defaults` project subdirectory, there is a short `wsgi.py` file with the following contents:

WSGI config for django_defaults project.

It exposes the WSGI callable as a module-level variable named ``application``.

```
import os
from django.core.wsgi import get_wsgi_application
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "django_defaults.settings")
application = get_wsgi_application()
```

That wsgi.py file was generated by the django-admin.py startproject command when the Django project was first created. Django exposes an application variable via that wsgi.py file so that a WSGI server can use application as a hook for running the web app. Here's how the situation looks visually:

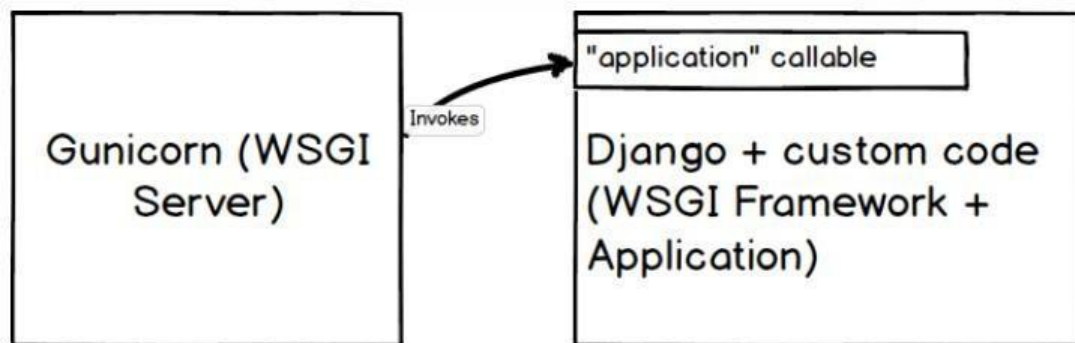


Fig.no 4.2

4.3.5 FLASK:



Fig.no 4.3

Flask is a web framework, it's a Python module that lets you develop web applications easily. It's has a small and easy-to-extend core: it's a microframework that doesn't include an ORM (Object Relational Manager) or such features. It does have many cool features like url routing, template engine. It is a WSGI web app framework.

Web Framework:

A Web Application Framework or a simply a Web Framework represents a collection of libraries and modules that enable web application developers to write applications without worrying about low-level details such as protocol, thread management etc.,

WSGI:

The Web Server Gateway Interface (Web Server Gateway Interface, WSGI) has been used as a standard for Python web application development. WSGI is the

specification of a common interface between web servers and web applications.

Werkzeug:

Werkzeug is a WSGI toolkit that implements requests, response objects, and utility functions. This enables a web frame to be built on it. The Flask framework uses Werkzeug as one of its bases.

Jinja2:

jinja2 is a popular template engine for Python. A web template system combines a template with a specific data source to render a dynamic web page.

This allows you to pass Python variables into HTML templates like this:

```
<html>

  <head>

    <title>{{ title }}</title>

  </head>

  <body>

    <h1>Hello {{ username }}</h1>

  </body>

</html>
```

Microframework:

Flask is often referred to as a microframework. It is designed to keep the core of the application simple and scalable. Instead of an abstraction layer for database support, Flask supports extensions to add such capabilities to the application.

Unlike the Django framework, Flask is very Pythonic. It's easy to get started with

Flask, because it doesn't have a huge learning curve. On top of that it's very explicit, which

increases readability. To create the “Hello World” app, you only need a few lines of code.

It’s a microframework, but that doesn’t mean your whole app should be inside one single Python file. You can and should use many files for larger programs, to handle complexity.

Micro means that the Flask framework is simple but extensible. You may all the decisions: which database to use, do you want an ORM etc, Flask doesn’t decide for you.

Flask is one of the most popular web frameworks, meaning it’s up-to-date and modern. You can easily extend it’s functionality. You can scale it up for complex applications.

4.3.6 AWS BOTO3:

Boto3 has two distinct levels of APIs. Client (or "low-level") APIs provide one-to-one mappings to the underlying HTTP API operations. Resource APIs hide explicit network calls but instead provide resource objects and collections to access attributes and perform actions.

Up-to-date and Consistent Interface:

Boto3's 'client' and 'resource' interfaces have dynamically generated classes driven by JSON models that describe AWS APIs. This allows us to provide very fast updates with strong consistency across all supported services.

Waiters:

Boto3 comes with 'waiters', which automatically poll for pre-defined status changes in AWS resources. For example, you can start an Amazon EC2 instance and use a waiter to wait until it reaches the 'running' state, or you can create a new Amazon DynamoDB table and wait until it is available to use. Boto3 has waiters for both client

and resource APIs.

Service-specific High-level Features:

Boto3 comes with many features that are service-specific, such as automatic multi-part transfers for Amazon S3 and simplified query conditions for Amazon DynamoDB.

4.4 IMPLEMENTATION:

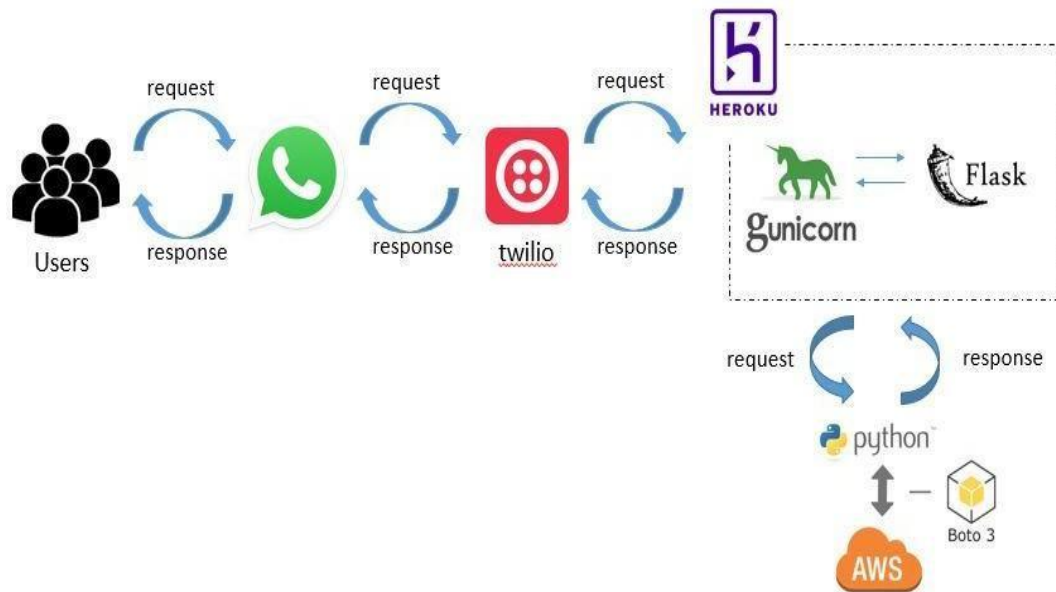


Fig.no 4.4

- WhatsApp Bot is provided to the user with the help of Twilio API to send and receive the requests so that we can interact in a user friendly manner.
- The requests from the WhatsApp Bot are transferred through Twilio API and redirected to the Flask.
- Heroku is a PaaS platform in which the Flask and Gunicorn is hosted there to handle the requests.
- Gunicorn is a Web Server Gateway Interface which synchronizes the requests coming from different users and send it to the Flask one by one.
- The Flask processes the request and send the specified function to perform to AWS Boto3 to execute the required scripts.
- AWS Boto3 will execute the scripts and creates a Machine Image of desired configuration requested by the user.

- The IP Address of the Machine Image created was returned to Flask in back as an output.
- Flask Transfers the IP Address to Twilio WhatsApp API as a response to the request asked.
- This IP Address will be received to the user as a response which can accessed by him to complete his work.

CHAPTER 5:

RESULTS AND DISCUSSIONS:

As a result of execution of the project it will act like a service which provides Virtual Machines to the user in his desired configuration. The user will message the bot and gets the reply like a simple chat.

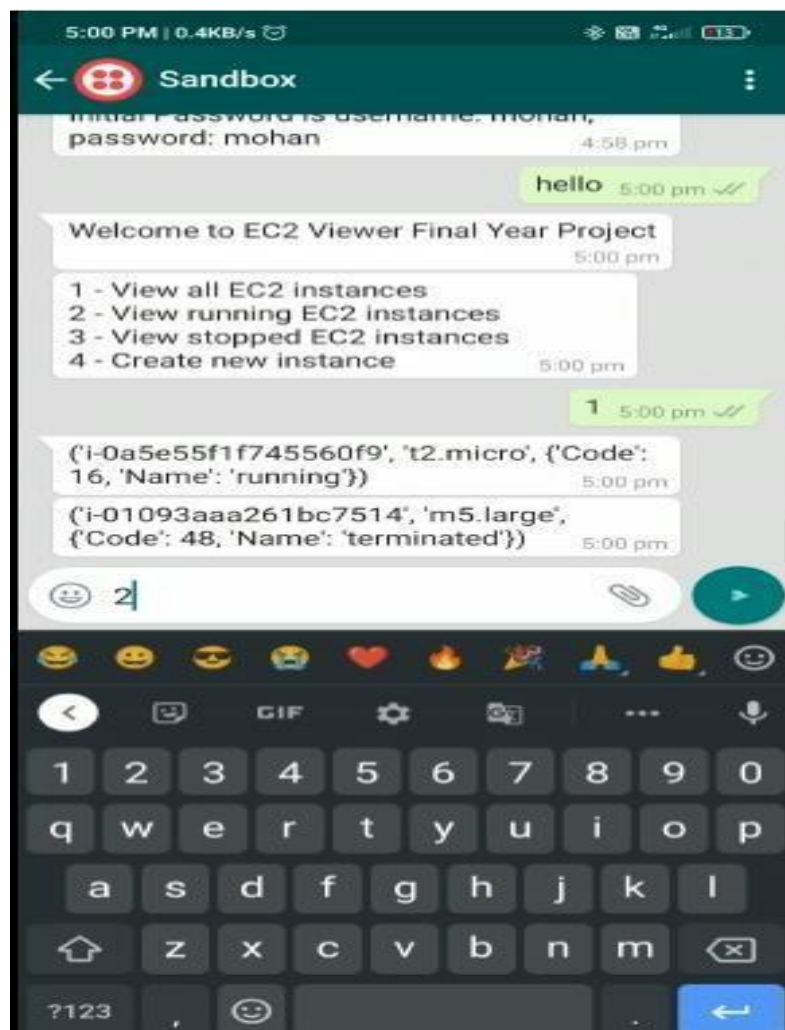


Fig.no 5.1

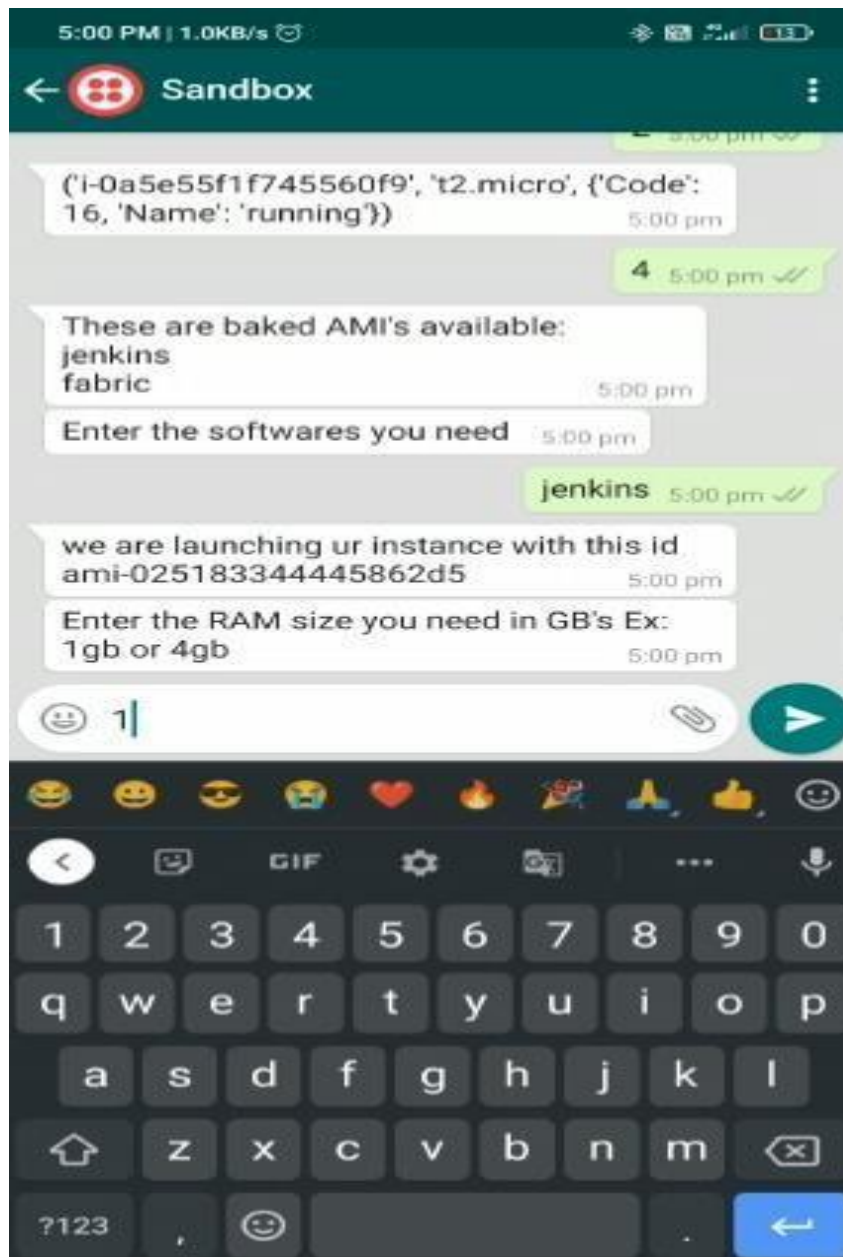


Fig.no 5.2

In the above figure the user asked to create a new instance then the machine replied to the user to grab information of the configuration the user needed and also displayed the baked AMI's available such that these softwares will be pre-installed which simplifies the work. After the user reply, it asks for the RAM size.



Fig.no 5.3

- After the user giving his answer it prompts to confirm whether to launch the instance or not, if yes it asks user in which format he wants either IP address or the DNS.

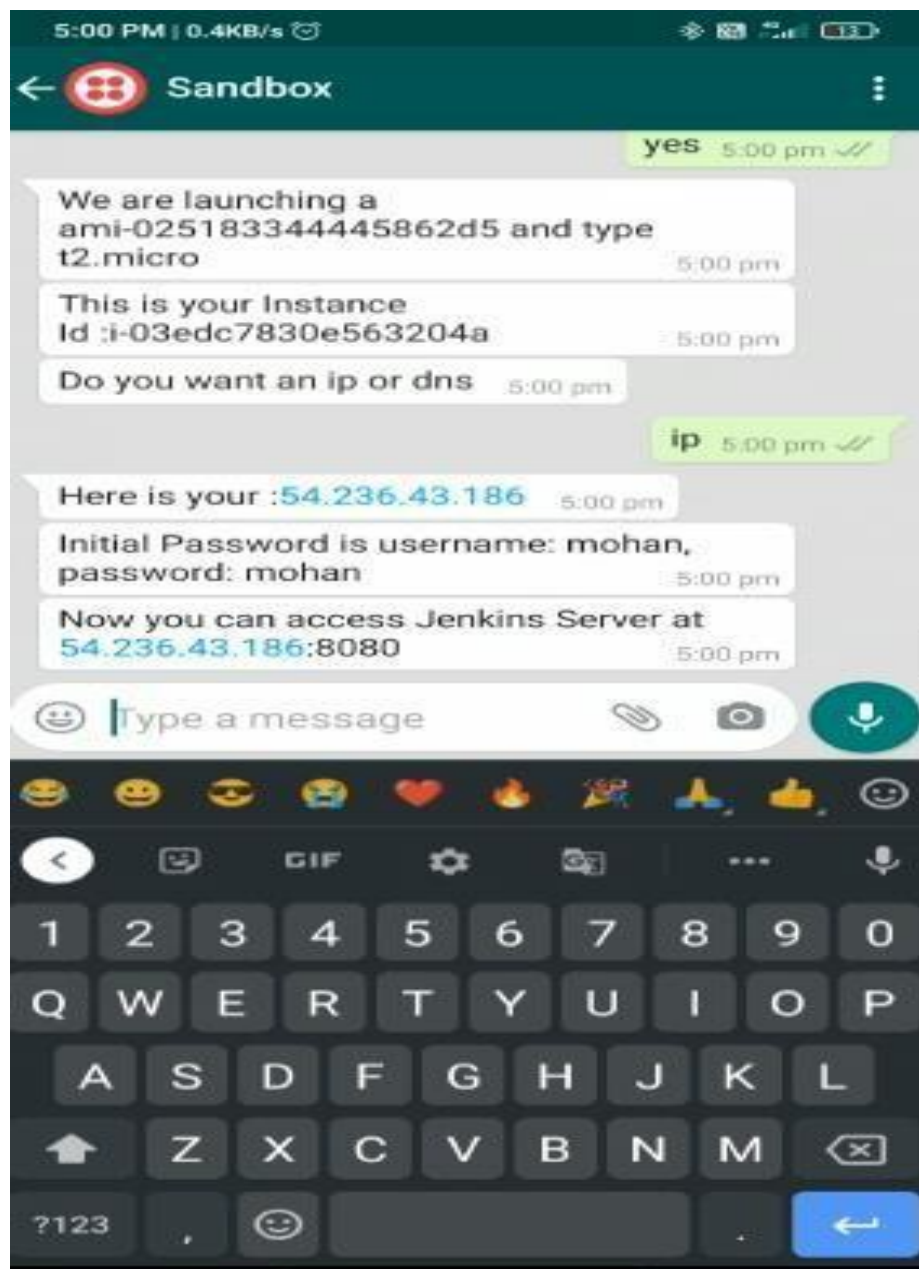


Fig.no 5.4

- If he clicks the IP address the user will get the IP Address to launch it directly in the chrome or any browser to use it.

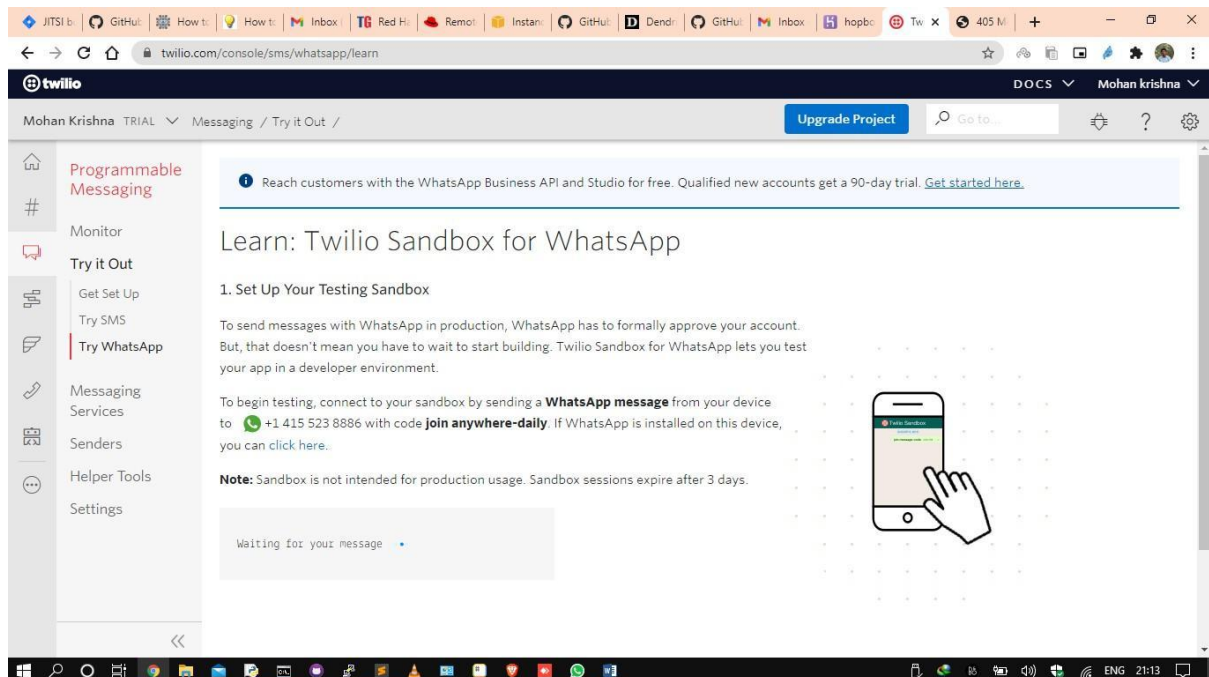


Fig.no.5.5

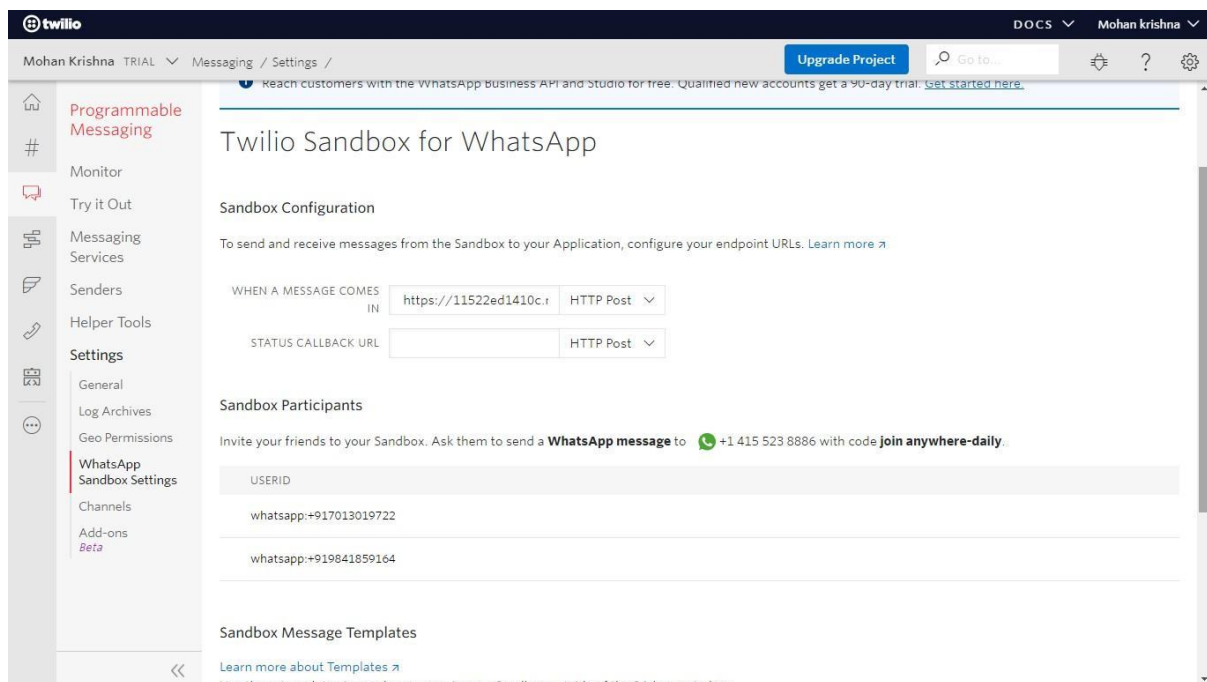


Fig.no 5.6

The image shows a Visual Studio Code editor window titled 'app.py - hopbottest - Visual Studio Code'. The editor has three tabs: 'app.py', 'db.py', and 'gui.py'. The 'app.py' tab is active, showing Python code. The code includes a function 'create_instance' that uses 'ec2_resource.create_instances' to create EC2 instances, a function 'parse_message' that splits a message and converts the first part to lowercase, and a function '__help' that takes a message and response. The terminal at the bottom shows the output of the deployment process, including installing pip, setuptools, and wheel, installing SQLite3, installing requirements with pip, discovering process types, declaring process types as web, compressing the code, and launching the application. The terminal output ends with 'Released v10' and 'https://hopbottest.herokuapp.com/ deployed to Heroku'. The status bar at the bottom shows 'Ln 158, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', and the system clock 'ENG 03:47'.

```
145 def create_instance(ami_type, types):
146     instances = ec2_resource.create_instances(ImageId=ami_type, MinCount=1, MaxCount=1, InstanceType=types, KeyName='project-sathyabama')
147     return (instances[0].id)
148
149
150
151
152
153
154 def parse_message(message):
155     str_array = message.split(' ')
156     str_array[0] = str_array[0].lower()
157     return str_array
158
159
160 def __help(message, response):
```

remote: ----> Installing pip 20.1.1, setuptools 47.1.1 and wheel 0.34.2
remote: ----> Installing SQLite3
remote: ----> Installing requirements with pip
remote: ----> Discovering process types
remote: Procfile declares types -> web
remote: ----> Compressing...
remote: Done: 58.1M
remote: ----> Launching...
remote: Released v10
remote: https://hopbottest.herokuapp.com/ deployed to Heroku
remote: Verifying deploy... done.
To https://git.heroku.com/hopbottest.git
a274352..15389a4 master -> master

Fig.no 5.7

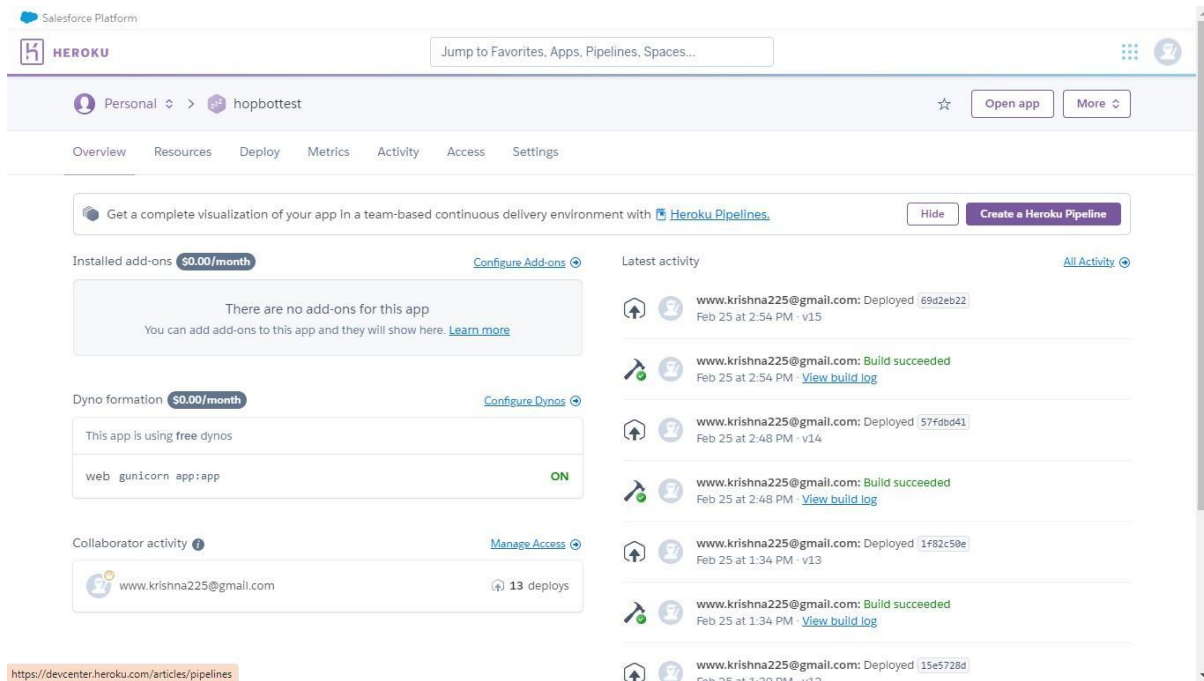


Fig.no 5.8

```
Command Prompt - heroku logs --tail - heroku logs - heroku logs - heroku logs
5.101.108.154" dyno=web.1 connect=0ms service=5ms status=405 bytes=377 protocol=https

C:\Users\MohanKrishna\Documents\GitHub\heroku\hopbottest>heroku logs
Warning: heroku update available from 7.48.0 to 7.50.0.
2021-03-09T15:41:35.629083+00:00 heroku[web.1]: Unidling
2021-03-09T15:41:35.668419+00:00 heroku[web.1]: State changed from down to starting
2021-03-09T15:41:43.047124+00:00 heroku[web.1]: Starting process with command gunicorn app:app
2021-03-09T15:41:47.201862+00:00 app[web.1]: [2021-03-09 15:41:47 +0000] [4] [INFO] Starting gunicorn 20.0.4
2021-03-09T15:41:47.206327+00:00 app[web.1]: [2021-03-09 15:41:47 +0000] [4] [INFO] Listening at: http://0.0.0.0:9576 (4)
2021-03-09T15:41:47.212283+00:00 app[web.1]: [2021-03-09 15:41:47 +0000] [4] [INFO] Using worker: sync
2021-03-09T15:41:47.231249+00:00 app[web.1]: [2021-03-09 15:41:47 +0000] [10] [INFO] Booting worker with pid: 10
2021-03-09T15:41:47.246020+00:00 app[web.1]: [2021-03-09 15:41:47 +0000] [11] [INFO] Booting worker with pid: 11
2021-03-09T15:41:47.586385+00:00 heroku[web.1]: State changed from starting to up
2021-03-09T15:41:51.572590+00:00 app[web.1]: request.form_body is hello
2021-03-09T15:41:51.577703+00:00 app[web.1]: 10.61.172.160 - - [09/Mar/2021:15:41:51 +0000] "POST /twilio HTTP/1.1" 200 249 "-" "TwilioProxy/1.1"
2021-03-09T15:41:51.577800+00:00 heroku[routel]: at=info method=POST path="/twilio" host=hopbottest.herokuapp.com request_id=1598d229-1207-4c8b-b8e0-05f4b8587e3d fwd="5.101.108.154" dyno=web.1 connect=1ms service=1834ms status=200 bytes=416 protocol=https
2021-03-09T15:42:34.775567+00:00 app[web.1]: 10.63.249.212 - - [09/Mar/2021:15:42:34 +0000] "GET / HTTP/1.1" 200 1261 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36"
2021-03-09T15:42:34.782594+00:00 heroku[routel]: at=info method=GET path="/" host=hopbottest.herokuapp.com request_id=18f6ad32-8886-44b1-aa4f-241e10abc68 fwd="175.101.108.154" dyno=web.1 connect=0ms service=27ms status=200 bytes=1423 protocol=https
2021-03-09T15:42:35.781549+00:00 app[web.1]: 10.63.249.212 - - [09/Mar/2021:15:42:35 +0000] "GET /favicon.ico HTTP/1.1" 404 232 "https://hopbottest.herokuapp.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36"
2021-03-09T15:42:35.781665+00:00 heroku[routel]: at=info method=GET path="/favicon.ico" host=hopbottest.herokuapp.com request_id=8fa2f176-775c-4d86-9f6b-a7b2e2a295dc fwd="175.101.108.154" dyno=web.1 connect=0ms service=4ms status=404 bytes=400 protocol=https
2021-03-09T15:42:42.278732+00:00 app[web.1]: 10.63.249.212 - - [09/Mar/2021:15:42:42 +0000] "GET /twilio HTTP/1.1" 405 178 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36"
2021-03-09T15:42:42.279853+00:00 heroku[routel]: at=info method=GET path="/twilio" host=hopbottest.herokuapp.com request_id=d72b0620-bdbf-4774-a915-89a1fb6c530a fwd="175.101.108.154" dyno=web.1 connect=0ms service=5ms status=405 bytes=377 protocol=https
2021-03-09T15:44:27.059831+00:00 app[web.1]: request.form_body is hello
2021-03-09T15:44:27.059927+00:00 app[web.1]: 10.47.153.32 - - [09/Mar/2021:15:44:27 +0000] "POST /twilio HTTP/1.1" 200 249 "-" "TwilioProxy/1.1"
2021-03-09T15:44:27.060172+00:00 heroku[routel]: at=info method=POST path="/twilio" host=hopbottest.herokuapp.com request_id=e453d0d5-0643-4632-932d-fdd04340815f fwd="5.101.108.154" dyno=web.1 connect=3ms service=10ms status=200 bytes=416 protocol=https
2021-03-09T15:44:34.726683+00:00 app[web.1]: request.form_body is 1
2021-03-09T15:44:35.416508+00:00 heroku[routel]: at=info method=POST path="/twilio" host=hopbottest.herokuapp.com request_id=d387face-cd6d-4213-8d6b-0c645e73cele fwd="5.101.108.154" dyno=web.1 connect=0ms service=701ms status=200 bytes=313 protocol=https
2021-03-09T15:44:35.427699+00:00 app[web.1]: 10.11.215.82 - - [09/Mar/2021:15:44:35 +0000] "POST /twilio HTTP/1.1" 200 146 "-" "TwilioProxy/1.1"

C:\Users\MohanKrishna\Documents\GitHub\heroku\hopbottest>
```

Fig.no. 5.9

CHAPTER 6

CONCLUSION AND FUTURE WORK:

CONCLUSION:

- This project eliminated the tedious task of deployment of Infrastructure using cloud.
- Automated the manual process of software installation and provided along with the Instance in a pre-installed format.
- This process thereby eliminated the Syntactical errors caused during the Project merging phase.
- A simple WhatsApp Bot with an easy user Interface is developed for the users to provide their requirements.
- Everything is automated in the backend without any manual process of deployment.
- A Front-End WhatsApp chatbot is made using Flask.
- This Bot was launched in Heroku to integrate with Cloud.
- Actions like viewing, listing and terminating the instances are coded to perform different actions.
- Integration of Front-End and Backend with Amazon Web Services was done successfully.
- Requests and Responses are sent and received through WhatsApp.

FUTURE WORK:

This project can be improved by adding a new feature by which the end customer can get the compute and storage from aws backend for a scheduled period of time so that billing doesn't exceed his expectations on usage of his resources. We can setup a cron job in cloudwatch by which scheduled period of request can be validated and after a certain time with a notification to the end user the resources provisioned can be terminated as well. By this feature both admin who takes care of this application as well as the end customer can be benefitted because the customer has no pain of making a note of the period of the resources he wants. As well the

admin also doesn't need to monitor the billing and check for resources requested.
Also an admin page

can be developed so that administrator of the application can maintain a list of resources that can be provided and software configurations and specifications of the resources can also be updated in that page itself instead of deploying the application again and again.

REFERENCES:

- Tuli, N. Hasteer, M. Sharma and A. Bansal, "Empirical investigation of agile software development: Cloud perspective", SIGSOFT Softw. Eng. Notes, vol. 39, pp. 1-6, 2014.
- F. Almudarra and B. Qureshi, "Issues in adopting agile development principles for mobile cloud computing applications", Procedia Comput. Sci., vol. 52, pp. 1133-1140, 2015.
- Nazir, A. Raana and M. F. Khan, "Cloud computing ensembles agile development methodologies for successful project development", Int. J. Mod. Educ. Comput. Sci., vol. 11, pp. 28-35, Nov. 2013.
- N. Jain and S. Dubey, "Agile development methodology with cloud computing", Int. J. Eng. Comput. Sci., vol. 3, no. 4, pp. 5373-5378, 2014.
- Inayat, S. S. Salim and Z. M. Kasirun, "Agile-based software product development using cloud computing services: Findings from a case study", Sci. Int. (Lahore), pp. 1065-1069, 2013.
- A. Tuli, N. Hasteer, M. Sharma and A. Bansal, "Empirical investigation of agile software development: Cloud perspective", SIGSOFT Softw. Eng. Notes, vol. 39, pp. 1-6, 2014.
- F. Almudarra and B. Qureshi, "Issues in adopting agile development principles for mobile cloud computing applications", Procedia Comput. Sci., vol. 52, pp. 1133-1140, 2015.
- A. Nazir, A. Raana and M. F. Khan, "Cloud computing ensembles agile development methodologies for successful project development", Int. J. Mod. Educ. Comput. Sci., vol. 11, pp. 28-35, Nov. 2013.
- N. Jain and S. Dubey, "Agile development methodology with cloud computing", Int. J. Eng. Comput. Sci., vol. 3, no. 4, pp. 5373-5378, 2014.

APPENDIX:

```
from flask import Flask, Response, request, render_template

from twilio import twiml

from twilio.twiml.messaging_response import Body, Message, Redirect,
MessagingResponse

import subprocess

app = Flask(_name_)

import boto3

import time

from botocore.config import Config

my_config = Config(
    region_name = 'us-east-1',
    signature_version = 'v4',

)

ec2_resource =
boto3.resource('ec2',config=my_config,aws_access_key_id="AKIAQEXJ7V7I4WQ3
KD",aws_secret_access_key="Q6bZAdN/04GZBx6C+Cmom7opcWJ3pnuBUWGSj")

def run_process(a_cmd):
    the_command = "docker %s" %a_cmd

    try:

        the_shell_cmd = subprocess.Popen(the_command, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

        the_stdout, the_stderr = the_shell_cmd.communicate()

        the_return_code = the_shell_cmd.returncode

        if the_return_code == 0:

            return the_stdout
```

except Exception as e:

```
raise e
```

```
s3_bucket = boto3.client('s3',region_name="us-east-1",aws_access_key_id="AKIAQEXJ7V7I4WQ3KD",aws_secret_access_key="Q6bZAdN/04GZBx6C+Cmom7opcWJ3pnuBUWGSjM")
```

```
#using client
```

```
def create_bucket(bucketName):
```

```
    new_bucket = s3_bucket.create_bucket(Bucket=bucketName)
```

```
    print("a new bucket is created as with public
```

```
access"+new_bucket['ResponseMetadata']['HTTPHeaders']['server'] +
new_bucket['ResponseMetadata']['HTTPHeaders']['location'])
```

```
#create_bucket("mohanharikrishna0544")
```

```
def attach_name_tags(right):
```

```
    tag_purpose_test = {"Key": Name, "Value": right}
```

```
    return tag_purpose_test
```

```
def attach_time_tags(right):
```

```
    tag_purpose_test = {"Key": time, "Value": right}
```

```
    return tag_purpose_test
```

```
def get_instance_name(fid):
```

```
    # When given an instance ID as str e.g. 'i-1234567', return the instance 'Name'
    from the name tag.
```

```
    ec2 = boto3.resource('ec2')
```

```
    ec2instance = ec2.Instance(fid)
```

```
    instancename = "
```

```
for tags in ec2instance.tags:
```

```
    if tags["Key"] == 'Name':
```

```
        instancename = tags["Value"]
```

```
return instancename
```

```

def create_instance(ami_type,types):

    #tag_purpose_test = attach_name_tags(software)

    #print(tag_purpose_test)

    instances =
ec2_resource.create_instances(ImageId=ami_type,MinCount=1,MaxCount=1,Insta
n ceType=types,KeyName='project-sathyabama',SecurityGroupIds=['sg-
07a712d7fae90e442'])

    return instances[0].id

```

#gets instance type based on ram and cpu

```
#def instance_type(ram):
```

```
def instance_type(ram):
```

```
    if(ram=="1"):
```

```
        type="t2.micro"
```

```
        print(type)
```

```
        return type
```

#gets ami based on software

```
def get_ami(software_needed):
```

```
    if(software_needed=="jenkins"):
```

```
        amis = "ami-025183344445862d5"
```

```
        return amis
```

```
    elif(software_needed=="fabric"):
```

```
        amis="ami-0a39501b0027a6428"
```

```
        print(amis)
```

```
        return amis
```

```

ec2_client = boto3.client('ec2',region_name="us-east-
1",aws_access_key_id="AKIAQEXJ7V7I4WQ3KDEZ",aws_secret_access_key="Q6
bZAdN/04GZBx6C+Cmom7opcWJ3pnuBUWGSjMSq")

```

#using client

```
def get_public_ip(instance_id):
```

```
    ip=ec2_client.describe_instances(InstanceIds=[instance_id])['Reservations'][0]['Instances'][0]['PublicIpAddress']
```

```
    return (ip)
```

```
#using client
```

```
def get_public_dns(instance_id):
```

```
    dns=ec2_client.describe_instances(InstanceIds=[instance_id])['Reservations'][0]['Instances'][0]['PublicDnsName']
```

```
    return (dns)
```

```
#get_public_ip("i-0777857a65e166dcb")
```

```
ami_gonna_launch=""
```

```
types=""
```

```
instance_id=""
```

```
def create_tags(instance_id,tag_name):
```

```
    response = ec2_client.create_tags(
```

```
        Resources=[
```

```
            instance_id,
```

```
        ],
```

```
        Tags=[
```

```
            {
```

```
                'Key': 'Name',
```

```
                'Value': tag_name,
```

```
            },
```

```
        ],
```

```
    )
```

```
#create_tags(instance_id,"jenkins")
```

```

@app.route('/')
def check_app():
    WELCOME = "Welcome to the App"
    return render_template('index.html')

@app.route("/twilio", methods=["POST"])
def inbound_message():
    # Retrieve response from twilio
    print("request.form_body is", request.form.get("Body"))
    response = MessagingResponse()
    # Gets the inbound message for SMS messages only
    inbound_message = request.form.get("Body")
    # Parses the inbound messages to only contain command and arguments
    str_array = parse_message(inbound_message)

    if str_array[0] == ("hello"):
        response.message("Welcome to EC2 Viewer Final Year Project")
        response.message("""1 - View all EC2 instances\n2 - View running EC2
instances\n3 - View stopped EC2 instances\n4 - Create new instance""")

    if str_array[0] == "1":
        for instance in ec2.instances.all():
            msg=(instance.id, instance.instance_type, instance.state)
            response.message(str(msg))

    if str_array[0] == "2":
        instanceses = ec2.instances.filter(Filters=[{'Name': 'instance-state-name',
'Values': ['running']}])
        for instance in instanceses:
            msg=(instance.id, instance.instance_type, instance.state)

```

```

        response.message(str(msg))
    if str_array[0] == "3":
        instances = ec2.instances.filter(Filters=[{'Name': 'instance-state-name', 'Values':
['stopped']}])
        for instance in instances:
            msg=(instance.id, instance.instance_type, instance.state)
            response.message(str(msg))
    if str_array[0] == "4":
        response.message("These are baked AMI's available:\njenkins\nfabric")
        response.message("Enter the softwares you need")
    if (str_array[0]=="jenkins"):
        global ami_gonna_launch
        global tag_name
        tag_name = str_array[0]
        ami_gonna_launch = get_ami(str_array[0])
        response.message("we are launching ur instance with this id
"+ami_gonna_launch)
        response.message("Enter the RAM size you need in GB's Ex: 1gb or 4gb ")

    if (str_array[0]=="fabric"):
        tag_name = str_array[0]
        ami_gonna_launch = get_ami(str_array[0])
        response.message("we are launching ur instance with this id
"+ami_gonna_launch)
        response.message("Enter the RAM size you need in GB's Ex: 1gb or 4gb ")

    if (str_array[0]=="1gb"):
        global types
        types = instance_type("1")
        response.message(types)

```

```
print(types)
```

```

#response.message("We are launching a wait for 2 min to launch")
response.message("Shall we launch with this? yes or no")

if (str_array[0]=="yes"):
    ami_launch = ami_gonna_launch
    global instance_id
    response.message("We are launching a "+ami_gonna_launch+" and type
"+types)
    instance_id = create_instance(ami_gonna_launch,types)
    response.message("This is your Instance Id :"+instance_id)
    print(instance_id)
    response.message("Do you want an ip or dns")
    #time.sleep(15)
    create_tags(instance_id,tag_name)

if (str_array[0]=="ip"):
    ipAddr = get_public_ip(instance_id)
    response.message("Here is your :"+ipAddr)
    temp = get_instance_name(instance_id)
    if(temp == "jenkins"):
        response.message("Now you can access Jenkins Server at
"+ipAddr+":8080")

        response.message("Initial Password is username: mohan, password:
mohan")

    return Response(str(response), mimetype="application/xml"), 200

def create_instance(ami_type,types):
    instances =
ec2_resource.create_instances(ImageId=ami_type,MinCount=1,MaxCount=1,Insta
n ceType=types,KeyName='project-sathyabama',SecurityGroupIds=['sg-
07a712d7fae90e442'])

    return (instances[0].id)

```

```
def parse_message(message):  
    str_array = message.split(' ')  
    str_array[0] = str_array[0].lower()  
    return str_array
```

```
def __help(message, response):  
    response.message("Invalid  
    Request")
```

```
if __name__ == '__main__':  
    app.run(debug=True, host='127.0.0.1')
```