

Comparing performance of bastion host on cloud using Amazon web services vs terraform

Sahana Bailuguttu, Akshatha S. Chavan, Oorja Pal, Kavya Sannakavalappa, Dipto Chakrabarti

Department of Electronics and Communication Engineering, Rashtreeya Vidyalaya College of Engineering, Bengaluru, India

Article Info

Article history:

Received Jul 11, 2022

Revised Dec 16, 2022

Accepted Jan 9, 2023

Keywords:

Amazon elastic cloud computing

Amazon web services

Bastion host

Cloud computing

Firewall

Terraform

ABSTRACT

In addition to security advantages like implementing defense in depth and complying with compliance standards, current bastion services are simple to deploy and fit into the DevOps culture. Bastions continue to be the most dependable and secure options for secure access to cloud infrastructures because they offer administrative simplicity without surrendering compliance and security. In this paper, an experimental set up was conducted to measure the cycle time it takes to provision resources using manual point-and-click graphical user interface (GUI) in Amazon web services (AWS) and time it takes for codified infrastructure to make application programming interface (API) calls using terraform. It also focuses on the design and deployment of Bastion host on AWS and terraform, and the comparison between the two with respect to various parameters.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Oorja Pal

Department of Electronics and Communication Engineering, Rashtreeya Vidyalaya College of Engineering
R V Vidyanikethan Post, Mysuru Road Bengaluru-560059, India

Email: oorja.pal2000gmail.com

1. INTRODUCTION

An on-demand delivery of computing resources is known as cloud computing. It is a system that enables fast, on-demand network access to a reservoir of common reconfigurable computing resources that can be easily provided and released without the need for administration work or service provider interaction. Cloud computing offers infrastructure, but it also has flaws that make it difficult to host and access resources via the internet, with security and privacy being one of the most prominent issues. However, there are ways to reduce these issues, including encryption, proxy servers, and VPNs. Firewalls are a type of such technology; they are the network security tools that monitor and filter network traffic in accordance with specified security standards. A bastion host is a dedicated system that hosts an application and is set up to filter traffic (ideally, single application). It is referred to as a “jump box” or “jump server” and has access to public networks. It functions as a bridge between the public internet and private instances, shielding the latter from nefarious traffic. In this paper, we will be demonstrating the deployment of a bastion host on the cloud using two softwares, namely Amazon web services (AWS) and terraform, and making a comparative study of various parameters after deployment on each platform.

Cloud computing is far more complicated today, as are the risks that come with it. Users may not even be employees because they are no longer seated in offices. Their devices can now access resources from any place because they are no longer confined to a desk and the hosting of the resources on the cloud is increasing. The instances in AWS are accessible via remote connection ports to the public internet, posing security threats to the private network. In this case, Bastion hosts serve as a “fortified checkpoint” to fend off the danger. Bastion hosts are often set up with the bare minimum of an operating system and protocol-specific servers like a recombination detection program (RDP) gateway or an OpenSSH server.

The use of terraform for cloud computing services such as infrastructure as a code (IaaS) [1] has its advantages. From on-premises to one or more cloud vendors, that infrastructure may be dispersed over several topographies. Additionally, terraform provides a fantastic approach to bundle and reuse common code in the form of code chunks referred to as modules. By accepting inputs and returning outputs, it offers a common interface for resource production and collaboration. Fotin and Cauz [2] dwells on the project named Magic Castle which is an infrastructure-as-code concept that creates a cluster architecture in a public or private cloud infrastructure. With the aid of Magic Castle, the authors show that it is simple, rapid, and affordable to provision virtual high-performance computing (HPC) systems on-demand in a public or private cloud architecture. Additionally, it is also demonstrated that these infrastructures may accommodate accelerators and quick interconnects, allowing them to still be regarded as “real” HPC resources. By 2022, multi cloud models will control 75% of the cloud computing business, according to surveys. To manage the high feature set of computing capacity under this paradigm, which is sometimes referred to as “Sky computing,” infrastructure solutions like cloud orchestrators have emerged. The topology and orchestration specification for cloud applications (TOSCA) standard and the tools that have been cited the most in the literature—cloudify, heat, cloud formation, terraform, and cloud assembly—are both examined in this paper [3]. It was demonstrated through a practical experiment and a review of the literature in [3] that terraform and cloudify are well-suited for use in sky computing scenarios. In the trial, terraform outperformed cloudify in a number of areas. The authors concluded that terraform showed an unexpected level of maturity during the experiments for this study, particularly in regards to error handling and delta operations. The services offered by various cloud providers have been compared and analyzed in [4] which helps in determining the feasibility of the choice of these cloud service providers for big data and IoT applications.

The researches [5]–[9] explain how to secure data in the cloud, both with and without a bastion host, while keeping costs in account. The systematic mapping study conducted in [9] brings attention to the paucity of articles in the area of cloud services. It is simply an analysis of security-related factors [10] associated with cloud data and other factors related to it. To achieve maximum data protection by lowering risks and threats, the authors go into detail about the data protection methods and tactics utilized around the world. Data security concerns for data-at-rest and data-in-transit are also included in the study. The study makes use of all platforms as a service (PaaS), software as a service (SaaS), and Infrastructure as a service (IaaS tiers). Data security, as well as the threats to it as well as potential remedies in cloud computing, is one of the paper’s main themes. They give a brief review of block cypher, stream cypher, and hash function used to encrypt data in the cloud, whether it is at rest or in transit. Mukherjee [11] discusses the advantages of using AWS in cloud computing.

The researches [12], [13] explained the emphasis is laid on automated cloud infrastructure, while the papers [14], [15] deal with multiple challenges facing cloud technology and security assessment. The ever-increasing trends in IoT and cloud are discussed in [16]. The data security aspects of virtualization in cloud scenarios are presented in [17]–[23], whilst [24], [25] present the performance review.

Despite all the studies into the usefulness, importance, and implementation of bastion hosts in the cloud, there is a need for a comparative study between platforms to aid developers and administrators to choose which platform is best suited for their configuration and/or use case. Thus, the paper presents a comparison between the implementation of bastion hosts through AWS and terraform and performs a detailed analysis of various parameters including setup times and its variation with scale of the configuration.

2. METHOD

The design flow is as shown in Figure 1. It is used as a blueprint for implementing a bastion host using AWS and terraform softwares to compare feasibility, operational capabilities and various other performance metrics. Some features, like the computing capacity, storage, and the speed of the Amazon elastic cloud computing instances, were limited for the pricing plans in AWS and terraform that were chosen in this work.

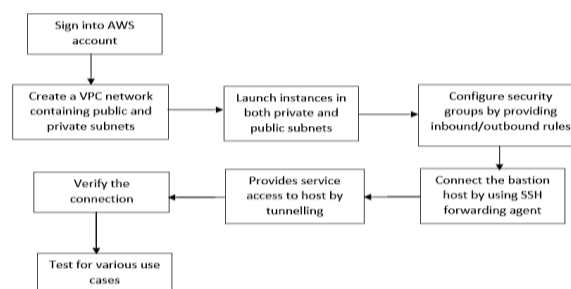


Figure 1. Design methodology

2.1. Amazon web services

Amazon web services is a popular and widely used cloud provider service due to its scalability and affordability. It provides businesses with access to a variety of on demand services such as computational power, database storage, content distribution and so on. Some of these products include pay-as-you-go pricing structure, storage, networking, analytics, and mobile development tools. In this research work, following features are used to implement the design.

2.1.1. Security groups

Security groups function very closely to virtual firewalls. It functions quite similarly to a conventional firewall. A virtual private cloud (VPC) instance can utilize it to track and filter the incoming and outgoing traffic using a set of rules. Based on protocols and ports, filtering is carried out.

2.1.2. Instances

The pool of public IPv4 addresses provided by Amazon is used to provide the instance with a public IP address that is not associated with any individual AWS account. After being detached from a specific instance, a public IP address is added back to the public IPv4 address pool and is no longer available for usage. An infrastructure-specific bastion host should only function with that unit and nothing else. To prevent the creation of superfluous security gaps, bastion host utilization is restricted to a particular instance or requirement.

2.2. Terraform

2.2.1. Configuring infrastructure

In the following project, the AWS provider is configured in region 'us-east-1' and credentials are initialized using arguments 'access_key' and 'secret_key' to authenticate the terraform AWS provider that allows creating resources. After the configuration for infrastructure is specified using Hashicorp's configuration language, it is then provisioned onto the provider using terraform workflow. 'Terraform init' command is used to initialize a working directory containing terraform configuration files and download plugins to manage various types of resources. 'Terraform apply' command performs a plan and then carries out the changes to be applied by making relevant application programming interface (API) calls.

2.3. Access control

Implementation of access control is done to add layers of security filters for the incoming traffic. User accounts (users), such as root and administrator accounts, system accounts for storing system files and processes, and service accounts for running programmes, are created for system administration. IP blacklisting (blacklist) technique used to prevent harmful or unauthorized IP addresses from connecting to specific networks. These lists are used in conjunction with traffic filtering solutions such as firewalls and intrusion prevention systems (IPS). Blacklists are created and used to filter malicious traffic in accordance with policies or by manually adding IP addresses. The ability to add new addresses to be blacklisted is a feature of many network security systems that use blacklists. This can be done in response to the findings of event analysis or whenever lists that are externally referred to are updated. Bash profile (profile) file is created wherein 'config/config' credentials are used to log into the Bastion host to configure the user list, blacklist of IPs and any other bastion configurations required; credentials 'foo/bar' is used to log into the private instance. In case the client IP belongs to the blacklist, the appropriate message is printed, and connection is refused or if the user credentials are invalid, the appropriate message is printed on the screen and connection is refused.

3. RESULTS AND DISCUSSION

this section graphically compares the time taken for deployment by various resources in both aws and terraform. it presents the various techniques that were applied to enhance the network security in the cloud. finally, a comparison is made between the two softwares about the advantages and disadvantages by taking into consideration the many factors that were studied during the process.

3.1. Hardening of infrastructure

To minimize the blast radius in the event of an incursion, the VPC is separated into sub-networks. This strategy entails creating and implementing a ruleset to regulate communications between particular hosts and services. The sub-networks are then designated hosts and resources with different availability zones. A more granular segmentation is incorporated by explicitly assigning security groups to each server in the network for a given zone. Communication between these services is established by explicitly adding routing rules to each sub-network. Therefore, the strategy of segmentation is used to reduce attack surfaces in the cloud infrastructure and bolster cloud security.

3.2. Access control and blacklisting IPs

3.2.1. Incorrect credentials

Upon entering incorrect credentials, the user is not permitted to enter the system. The username and password entered are hey and you are respectively. As they are incorrect, the user is denied access.

3.2.2. Blacklisting IPs

Certain IPs have been blacklisted to prevent access into the system. The IPs that have been blacklisted are basically the ones that are identified as malicious and could be a potential threat to the system. Thus, a user that tries to login from a server whose IP is blacklisted, will not be allowed to enter the system. This serves the purpose of a firewall as it provides security to the system.

3.3. Enhancement provided by terraform over AWS

Infrastructure consisting of bastion host, isolated server, and database instance, where bastion host acts as a ‘jump’ server is deployed on AWS providers. This was configured using terraform’s configuration language on terraform. Then the terraform initializes the backend and installs plugins of the provider to create resources. Once this infrastructure is deployed it is connected from a local machine using Secure Shell Session using the private key pairs.

First, the private key pairs are added on pageant and then SSH agent forward is established to connect to the remote server. The bastion host was accessed by the local machine establishing an SSH connection over port 22. Once the bastion server is accessed, a SSH command is given to connect to the remote server. Then the private instance was accessed by the local machine after hopping from the bastion host. Finally, secure connection into the database instance is established through the bastion server.

With AWS, on the other hand, creating separate instances involves making use of the IPv4 addresses provided by Amazon and adding back the detached public IP address to the public IPv4 address pool. It also involves restricting the utilization of the bastion host to specific instances to avoid creating superfluous security gaps. As a result, this increases the overall time taken to create the instances. In this case, the main enhancement provided by terraform is that it takes a much smaller amount of time to deploy the same resources, which becomes a crucial aspect of the software when working on very large databases and using priced cloud services. This is further elaborated in the following section where the quantitative comparison between the performances of AWS and terraform is illustrated.

3.4. AWS and terraform performance comparison

An experimental set up was conducted to measure the cycle time it takes to provision resources using manual point-and-click graphical user interface (GUI) in AWS and time it takes for codified infrastructure to make API calls using terraform. Basically, we timestamp the resources from configuring to deployment and tabulate it as shown in the table. This time data is then extrapolated to timing graphs. Figure 2 shows timing graphs for manually configuring and deploying resources in AWS, as well as for the time taken by terraform to initialize plugins and make relevant API calls to provision infrastructure. The following equation calculates the percentage decrease in time taken for terraform when compared to AWS, and the results for various resources are tabulated in Table 1.

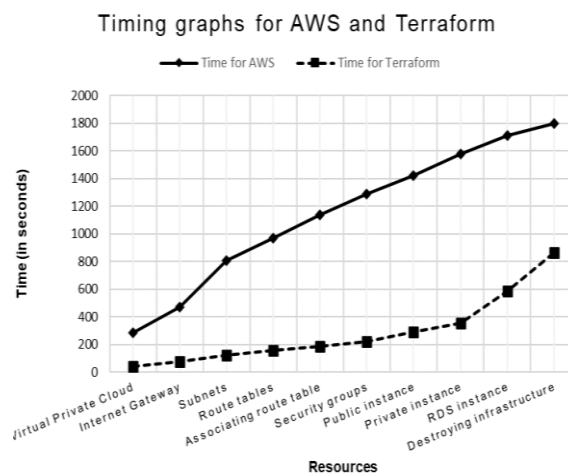


Figure 2. Resources vs time for AWS and terraform

Table 1. Resources vs time for AWS and terraform

Resources	Time for AWS (in seconds)	Time for terraform (in seconds)	Decrease in time in percentage
Virtual private cloud	288	44	84.72
Internet gateway	468	78	83.33
Subnets	806	120	85.11
Route tables	971	157	833.83
Associating route table	1,139	186	83.86
Security groups	1,288	221	82.84
Public instance	1,421	293	79.88
Private instance	1,579	356	77.45
RDS instance	1,714	585	65.86
Destroying infrastructure	1,800	867	51.33

$$\% \text{ Decrease} = ((\text{Time for Terraform} - \text{Time for AWS}) \div (\text{Time for AWS})) \times 100$$

- Provisioning resources, namely VPC, Internet gateway, subnets, route tables, associating route tables and security groups takes significantly less time using terraform than AWS by 82.82%.
- Deploying bastion, private and database instances using AWS takes 65.85% more time than applying changes to infrastructure using terraform.
- Overall, it can be concluded that the cycle time to spin up resources onto providers is less in terraform compared to AWS by 78.91%.
- In addition to this the time taken to destroy infrastructure in terraform is 51.83% less than in AWS.

4. CONCLUSION

In the field of data and communication technology, the emergence of cloud computing heralds the beginning of a new phase since it introduces a paradigm shift in computing's approach to development. Users are still learning about this technology, and a gradual transition from conformist subtracting to cloud computing will take place. Because of this technology, programmers with innovative ideas for internet services won't have to shell out a lot of money to organize their tools and applications.

The infrastructure for bastion host is configured using AWS by manual point-and-click GUI and using HashiCorp's terraform infrastructure-as-code tool on AWS cloud provider. This infrastructure consists of bastion host, isolated server, and database instance, where the bastion host acts as a 'jump' server. Strategies such as network segmentation and explicit assignment of security rules to each service are used to reduce attack surface and harden the infrastructure which enhance the network security in the cloud. Access control and blacklisting of IPs is provided to gain more control on traffic flow and thus bolster the security of infrastructure.

It was found that AWS's system of utilizing the IPv4 address pool provided by Amazon while creating instances, while being useful for the prevention of superfluous security gaps, resulted in a longer overall resource deployment time. Terraform, on the other hand, introduced a much faster and more efficient method of deploying resources with the help of Hashicorp's configuration language and the terraform workflow. This is especially useful in real life applications that involve a large number of records being stored in a database, or when using cloud services that must be paid for. It can be observed from the resource vs timing graph that the overall cycle time to spin up resources onto providers is less in terraform compared to AWS by 78.91%. In addition to this the time taken to destroy infrastructure in terraform is 51.83% less than in AWS. Thus, infrastructure-as-code tools such as terraform aid the automation process of DevSecOps and make it more time efficient and hence, makes faster iterative updates possible unlike slow and cumbersome manual way of provisioning. When compared to an existing software such as AWS, the feasibility of HashiCorp's configuration language, immutable approach of terraform and terraform cloud agnostic property makes it much easier to collaborate, innovate and scale technologies, thus accelerating cloud adoption.





REFERENCES

- [1] M. Howard, "Terraform-automating infrastructure as a service," 2022, *arXiv:2205.10676*.
- [2] F.-A. Fortin and A. Ó Cais, "Magic Castle-enabling scalable HPC training through scalable supporting infrastructures," *The Journal of Computational Science Education*, vol. 13, no. 1, pp. 21–22, Apr. 2022, doi: 10.22369/issn.2153-4136/13/1/3.
- [3] A. Ibrahim, A. H. Yousef, and W. Medhat, "DevSecOps: A security model for infrastructure as code over the cloud," in *MIUCC 2022 - 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference*, May 2022, pp. 284–288, doi: 10.1109/MIUCC55081.2022.9781709.
- [4] M. F. Falah *et al.*, "Comparison of cloud computing providers for development of big data and internet of things application," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 22, no. 3, pp. 1723–1730, Jun. 2021, doi: 10.11591/ijeecs.v22.i3.pp1723-1730.





- [5] T. Autio, "Tuomas autio securing a kubernetes cluster on google cloud platform," *Metropolia University of Applied Sciences*, 2021.
- [6] *Bastion Hosts: Protected Access for Virtual Cloud Networks*, Oracle, 2021. [Online]. Available: <https://docs.oracle.com/en-us/iaas/Content/Resources/Assets/whitepapers/bastion-hosts.pdf>.
- [7] L. R. De Carvalho and A. P. F. De Araujo, "Performance comparison of terraform and cloudify as multicloud orchestrators," in *Proceedings - 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020*, May 2020, pp. 380–389, doi: 10.1109/CCGrid49817.2020.00-55.
- [8] M. M. Hasan, F. A. Bhuiyan, and A. Rahman, "Testing practices for infrastructure as code," in *LANGETI 2020 - Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing, Co-located with ESEC/FSE 2020*, Nov. 2020, pp. 7–12, doi: 10.1145/3416504.3424334.
- [9] I. Odun-Ayo, T. A. Williams, O. Abayomi-Alli, and J. Yahaya, "Systematic mapping study of economic and business models of cloud services," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 18, no. 2, pp. 987–994, May 2020, doi: 10.11591/ijeecs.v18.i2.pp987-994.
- [10] S. Zaineldeen and A. Ate, "Improved cloud data transfer security using hybrid encryption algorithm," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 20, no. 1, pp. 521–527, Oct. 2020, doi: 10.11591/ijeecs.v20.i1.pp521-527.
- [11] S. Mukherjee, "Benefits of AWS in modern cloud," Cornell University, 2019.
- [12] S. Garg and S. Garg, "Automated cloud Infrastructure, continuous integration and continuous delivery using docker with Robust container security," in *Proceedings - 2nd International Conference on Multimedia Information Processing and Retrieval, MIPR 2019*, Mar. 2019, pp. 467–470, doi: 10.1109/MIPR.2019.00094.
- [13] P. S. P. Shenoy, S. S. Vishnu, R. P. Kumar, and S. Bailuguttu, "Enhancement of observability using Kubernetes operator," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 25, no. 1, pp. 496–503, Jan. 2022, doi: 10.11591/ijeecs.v25.i1.pp496-503.
- [14] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba, "Adoption, support, and challenges of infrastructure-as-code: Insights from industry," in *Proceedings - 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019*, 2019, pp. 580–589, doi: 10.1109/ICSME.2019.00092.
- [15] J. K. R. Sastry and M. T. Basu, "Securing SAAS service under cloud computing based multi-tenancy systems," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 13, no. 1, pp. 65–71, Jan. 2019, doi: 10.11591/ijeecs.v13.i1.pp65-71.
- [16] R. Rauscher and R. Acharya, "Virtual machine placement in predictable computing clouds," in *IEEE International Conference on Cloud Computing, CLOUD*, Jun. 2014, pp. 975–976, doi: 10.1109/CLOUD.2014.148.
- [17] R. Kaur and J. Kaur, "Cloud Computing security issues and its solution: A review," in *2015 International Conference on Computing for Sustainable Global Development, INDIACom 2015*, 2015, pp. 1198–1200.
- [18] N. C. Rao, S. De, and C. Choudhary, "Experimental analysis of improvement in firewall Traffic by deployment of Bastion Host," *International Journal of Advanced in Management, Technology and Engineering Sciences*, vol. 8, no. 3, pp. 2249–7455, 2018.
- [19] G. Vijayababu, D. Hariitha, and R. S. Prasad, "Review on bastion hosts," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 14, pp. 81-88, 2018.
- [20] A. Albugmi, M. O. Alassafi, R. Walters, and G. Wills, "Data security in cloud computing," in *2016 Fifth International Conference on Future Communication Technologies (FGCT)*, Aug. 2016, vol. 143, pp. 55–59, doi: 10.1109/FGCT.2016.7605062.
- [21] N. Jain and S. Choudhary, "Overview of virtualization in cloud computing," in *2016 Symposium on Colossal Data Analysis and Networking, CDAN 2016*, Mar. 2016, pp. 1–4, doi: 10.1109/CDAN.2016.7570950.
- [22] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, ser. CODASPY '17*. New York, NY, USA: ACM, 2017, doi: 10.1145/3029806.3029832.
- [23] A. Markandey, P. Dhamdhare, and Y. Gajmal, "Data access security in cloud computing: A review," in *2018 International Conference on Computing, Power and Communication Technologies, GUCON 2018*, Sep. 2019, pp. 633–636, doi: 10.1109/GUCON.2018.8675033.
- [24] K. Munir and S. Palaniappan, "Secure cloud architecture," *Advanced Computing: An International Journal*, vol. 4, no. 1, pp. 9–22, Jan. 2013, doi: 10.5121/acij.2013.4102.
- [25] C.-Y. Tseng, K.-Y. Liu, and L.-T. Lee, "Enhance the performance of virtual machines by using cluster computing architecture," *TELKOMNIKA (Telecommunication, Computing, Electronics and Control)*, vol. 11, no. 5, May 2013, doi: 10.11591/telkomnika.v11i5.2486.

BIOGRAPHIES OF AUTHORS



Sahana Bailuguttu     is an assistant professor in the department of electronics and communication engineering at RV College of Engineering, Bengaluru. She holds a Ph.D. in communication networks and a M. Tech. degree in computer network engineering from Dayananda Sagar College of Engineering. She has a teaching experience of 14 years and has taught the subjects basics of electronics engineering, analysis and design of digital circuits, computer communication networks, optical fiber communication, ARM processor to undergraduate and postgraduate students. Her current areas of interest are fault tolerance in networking and machine learning. She can be contacted at email: sahanab@rvce.edu.in.







Akshatha S. Chavan     received her B.E. in electronics and communication engineering from RV College of Engineering in 2022. Her interests are cloud computing and networking. She can be contacted at email: akshathaschavan.ec18@rvce.edu.in.







Oorja Pal     received her B.E. in electronics and communication engineering from RV College of Engineering in 2022. Her interests are cloud computing and networking. She can be contacted at email: oorja.pal2000@gmail.com.



Kavya Sannakavalappa     received her B.E. in electronics and communication engineering from RV College of Engineering in 2022. Her interests are cloud computing and networking. She can be contacted at email: kavyas.ec18@rvce.edu.in.



Dipto Chakrabarti     received his B.E. in electronics and communication engineering from RV College of Engineering in 2022. His interests are cloud computing and Networking. He can be contacted at email: diptoc.ec18@rvce.edu.in.