Volume: 11 Issue: V Month of publication: May 2023
DOI: https://doi.org/10.22214/ijraset.2023.51356

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue V May 2023- Available at www.ijraset.com

Serverless and IaC

Divya Srivastava[1], Rudrendra Bahadur Singh[2] (Assistant Professor), Jayant Pratap Singh[3], Harsh Pratap Singh[4], Prateek Saxena[5]
Department of Computer Science, Babu Banarsi Das Institue of Technology and management, Lucknow, India

Abstract: The objective of this research paper is to investigate the potential advantages of integrating Terraform and Serverless Computing to construct a scalable and efficient cloud infrastructure. Terraform, an infrastructure-as-code open-source tool, and Serverless Computing, a new paradigm that enables developers to run code without worrying about the underlying infrastructure, are briefly described along with their benefits. The paper then explores how these two can be combined to build a dynamic and robust infrastructure while also addressing the difficulties that arise when constructing a Serverless Computing infrastructure with Terraform, proposing solutions to overcome them. Finally, the study concludes that integrating Terraform and Serverless Computing can help organizations create an efficient, adaptable, and scalable infrastructure while decreasing expenses and enhancing developer productivity.

Keyword: Serverless, Iac, Terraform, DevOps, Agile

I. INTRODUCTION

Serverless computing is gaining traction as a novel and promising method for deploying cloud applications, primarily because of the move towards containerization and microservices in enterprise application architectures [2]. With serverless computing, users only pay for what they use, and servers are started and stopped automatically, fulfilling the original vision of cloud computing as a utility [20]. Developers utilizing serverless computing can benefit from cost savings and scalability without requiring extensive cloud computing knowledge, which can be a time-consuming endeavor to acquire.

Serverless computing has become increasingly popular due to its simplicity and economic benefits, as evidenced by the rising search volume for the term "serverless" on Google Trends. The market for serverless computing is expected to reach $7.72 billion by 2021, according to estimates. Major cloud providers such as Amazon, IBM, Microsoft, and Google have already introduced serverless computing capabilities, and there are also several open-source efforts led by industry and academic institutions, as seen in the CNCF Serverless Cloud Native Landscape1.

As far as IaaS customers are concerned, the shift towards serverless computing presents both advantages and disadvantages. On one hand, it simplifies the programming model for developing cloud applications by abstracting operational concerns such as scalability, fault tolerance, over/under provisioning of VM resources, and server management. This enables developers to focus on the business aspects of their applications, while reducing the cost of deployment by charging only for execution time. However, on the other hand, deploying such applications on a serverless platform can be challenging, as it requires platform providers to make design decisions related to quality-of-service (QoS) monitoring, scaling, and fault-tolerance properties. Relinquishing these decisions to the platform provider may cause conflicts if the application requirements evolve beyond the capabilities of the platform.

Serverless computing is a cloud-native platform that provides developers with the ability to run code on-demand, with automatic scaling and billing only for the duration of code execution. This platform hides server usage from developers, offering two key features: cost-effectiveness, with a pay-as-you-go model, and elasticity, enabling scaling from zero to "infinity"

There are several misunderstandings regarding serverless computing, beginning with its name. While servers are still necessary, developers do not have to manage them. Instead, the serverless platform handles decisions such as server count and capacity, with server capacity being automatically allocated as required by the workload. As a result, computation (in the form of a stateless function) is decoupled from its execution location, providing an abstraction layer.

Our approach in this paper is to provide a layered design and comprehensive summary of the research domains from multiple perspectives. This can assist researchers and practitioners in gaining a deeper understanding of the fundamental characteristics of serverless computing. In Figure 2, we present an analysis of its design architecture using a bottom-up approach, which allows us to separate the different components of serverless computing.

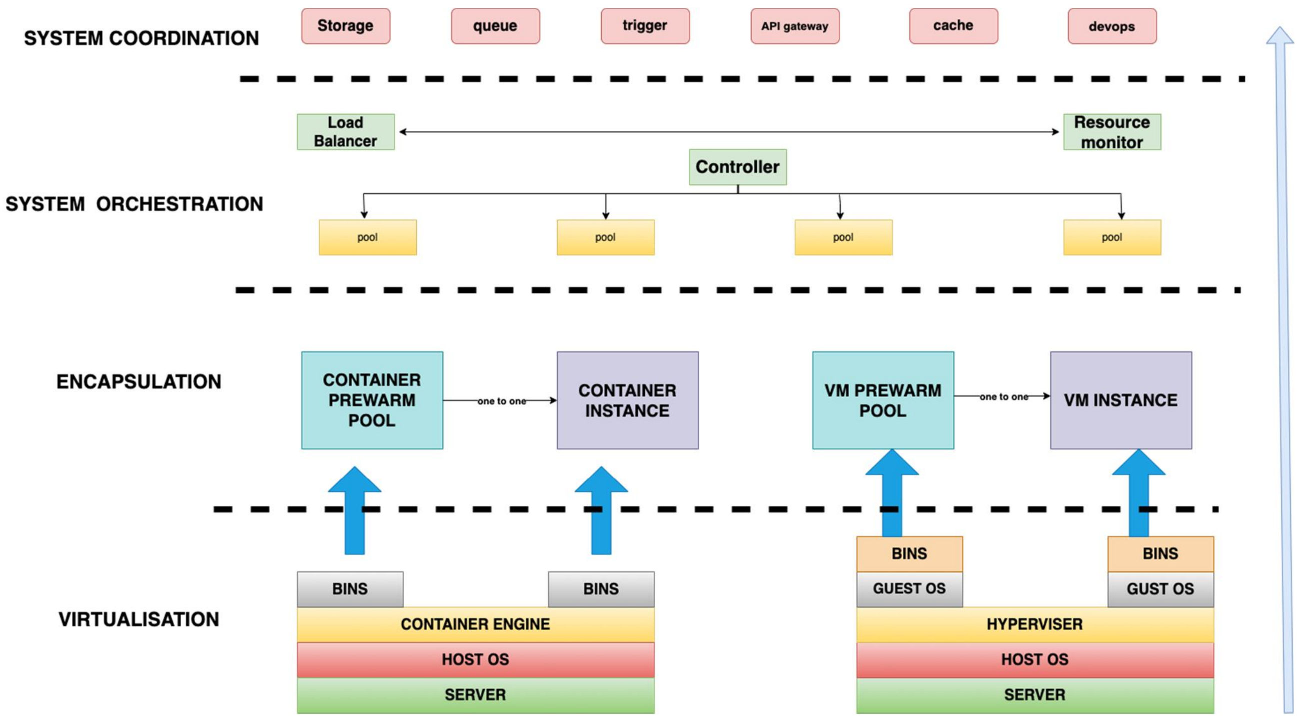


Fig.1 Layered serverless architecture

At the lowest level, the Virtualization layer provides function isolation and a secure sandbox environment with optimal performance and functionality. Moving up, the Encapsule layer comprises various middlewares that allow customized function triggers and executions, as well as data metrics collection for communication and monitoring purposes.
The System Orchestration layer comes next, which allows users to configure triggers and binding rules to ensure high availability and stability of their applications by dynamically adjusting to changes in load.
Finally, the System Coordination layer consists of Backend-as-a-Service (BaaS) components that use unified APIs and SDKs to integrate backend services into functions. .
From the viewpoint of a cloud provider, serverless computing offers several advantages. It provides an opportunity to control the entire development stack, optimize and manage cloud resources efficiently to reduce operational costs, offer a platform that promotes the use of additional services in their ecosystem, and reduce the effort required to develop and manage large-scale cloud applications. Infrastructure as code (IaC) is a methodology that involves using code to automate the process of setting up virtual machines and networks, installing software packages, and configuring the
environment for a specific application, rather than relying on manual commands. This code manages the infrastructure, which can include physical equipment, virtual machines, containers, and software-defined networks.

A. Terraform

Terraform is a tool that applies the IaC for provisioning infrastructure. It simplifies the provisioning of cloud infrastructure by allowing users to build, modify and version control it using configuration files. The infrastructure state is defined in the Terraform configuration files and is provisioned by executing the scripts through the Terraform binary. One of the main advantages of using Terraform is its ease of use and flexibility to add or modify infrastructure for different cloud providers. Additionally, Terraform's IaC capabilities allow for dynamic provisioning of infrastructure, reducing the risk of vendor lock-in due to its support for multiple providers. Terraform uses HCL (HashiCorp Configuration Language), a language developed by HashiCorp (the creators of Terraform) that is used consistently across all providers supported by Terraform.

Terraform can be used with multiple cloud providers, making it cloud-agnostic. However, it is not possible to use a single configuration to deploy the same infrastructure across different providers.Heat is a software that works similarly to Terraform but it is limited to only one platform, which is OpenStack. In contrast, Terraform can perform similar tasks and enable multiple providers. For example, it can orchestrate an AWS and an OpenStack cluster simultaneously
To replicate an infrastructure on two different providers, separate configurations need to be created with some shared functions like variables. However, changing providers is straightforward as the syntax, functions, and thought process to write code remain the same. HCL can be combined with JSON to provide more flexibility. For instance, a configuration for an AWS cloud is shown in Listing 1. When executed, Terraform creates one t2.micro instance in the us-east-1 region using the user's access and secret key. The provider block specifies the provider, and the resource block describes the provisioned resources. Terraform can also manage storage, networking, DNS entries, SaaS features, and much more beyond compute instances.

II. INFRASTRUCTURE AS CODE (IAC)

The modern software delivery life cycle covers various stages, including requirements analysis, configuration, and production operations, and encompasses both development and operations activities.

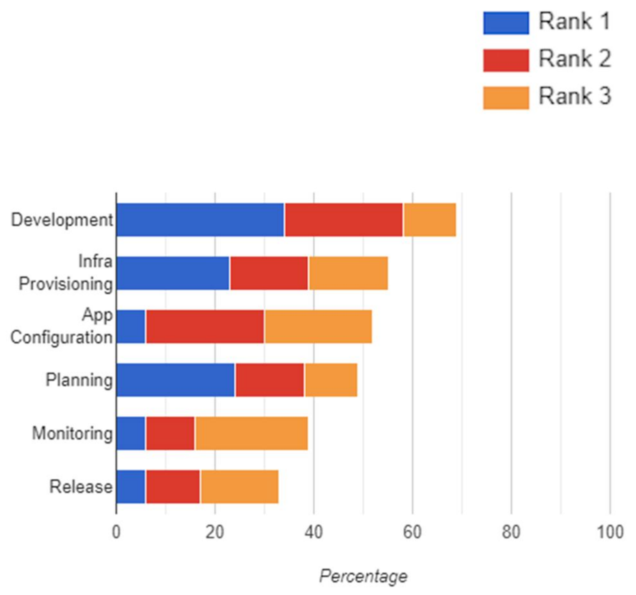


Fig.2 Issues in lifecycle

A study(conducted by Forrester Consulting on behalf of Microsoft, February 2015) conducted revealed that respondents faced the most issues and conflicts in development/testing and infrastructure and application configuration. Therefore, organizations require a solution that can help eliminate friction and delays across all development and testing sub-cycles and establish faster, more reliable, and repeatable processes for configuring both applications and infrastructure.
According to a survey, the primary obstacle to faster delivery for Ops and Dev teams is the lack of collaboration between the two teams, with [46%] of respondents citing it as a major challenge. This lack of collaboration is worsened by a skills mismatch [27%], as Dev and Ops teams often use different languages to specify configurations. This can lead to miscommunication, misunderstandings, and a higher frequency of errors, particularly when development or test environments are not configured to match production environments.
The most obvious and proven way to speed up any technology process is through automation. By defining infrastructure configurations as code, IaC enables automation of infrastructure creation and management processes, reducing the likelihood of errors and misconfigurations. This can lead to faster, more reliable, and more repeatable processes for configuring both applications and infrastructure, ultimately improving the delivery lifecycle for the entire business.

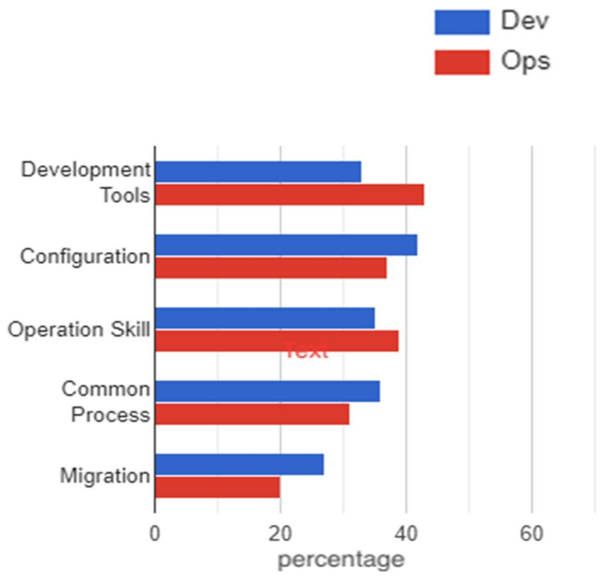


Fig.3 Most valuable benefits of IaC from Dev and Ops
Point of view

Infrastructure as code (IaC) is a DevOps practice that enables automation of the infrastructure deployment process. By writing code that describes the infrastructure, IaC allows resources to be created, destroyed, resized, replaced, and moved easily. This results in faster deployment speed, improved security, high scalability, and automated backup and restore. IaC involves scripting the entire environment, from installing the operating system and configuring servers on instances to specifying how the instances' software communicates with one another, and more. Redeploying infrastructure became more convenient with IaC.Sharing and utilizing code and infrastructure also became easier. IaC enables infrastructure to be easily scaled up or down as needed. The pay-per-use policy associated with IaC helps to reduce infrastructure costs.

While there are many benefits of IaC, one of the most significant is improved collaboration between Dev and Ops teams. By using IaC, Ops teams can configure apps and infrastructure more quickly, reliably, and repeatedly, resulting in fewer errors. Additionally, IaC allows Ops teams to use languages and methodologies that are familiar to developers, enabling better collaboration and reducing friction between the two groups.

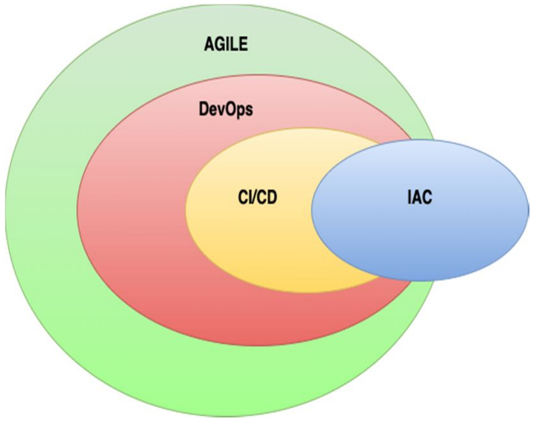


Fig.4 relation between Agile,DevOps,Iac

B. Relationship Between IaC, Agile, and DevOps

IaC is a modern software development practice that shares some similarities with Agile and is closely related to DevOps. While IaC automates the creation of execution environments, it also promotes Agile values by facilitating faster and more flexible software development. At the same time, it aligns with DevOps goals by automating deployment, reducing cycle time, and improving the quality of the deployment delivery mechanism. However, IaC is not a subset of either Agile or DevOps, but rather a complementary practice that helps organizations achieve their goals in software development and delivery

C. Iac and iaas

Historically, IT system operators have used ad hoc scripting to automate tasks, but the emergence of cloud computing, specifically infrastructure-as-a-service (IaaS) technology, gave rise to IaC technology and practices. In an IaaS-based environment, virtualization is used for all computation, storage, and network resources, and these resources need to be allocated and configured using application programming interfaces (APIs). IaC technology was developed to address the challenges of creating and destroying environments frequently, which could contain many virtual machines, even thousands in the case of large-scale services on the internet. IaC tools generally consist of a scripting language for defining the desired system configuration and an orchestration engine that executes the scripts and invokes the IaaS API.

III. SERVERLESS AND TERAFORM

Terraform is a tool that enables the management of the entire serverless application lifecycle, starting from the initial development and testing phase to production deployment and maintenance. Below are the primary stages of the serverless application lifecycle where Terraform can be employed:

1) *Development:* Developers have the ability to define the required infrastructure and resources for their serverless application using code. The resources, such as AWS Lambda functions and API Gateway, can be defined in configuration files within Terraform.
2) *Testing:* Develepors can leverage Terraform's built-in testing framework, Terratest, to verify that the infrastructure is correctly provisioned and configured. By writing tests in code, developers can easily replicate the testing process across different environments and ensure consistent results. Terratest supports a wide range of tests, including unit tests, integration tests, end-to-end tests, And more, providing a comprehensive testing solution for serverless applications.
3) *Deployment:* Using Terraform, developers can automate the deployment process of a serverless application and ensure consistent infrastructure deployment in the production environment
4) *Maintenance:* Terraform can be utilized to handle the continuous maintenance of the serverless application. Developers can utilize Terraform to modify the infrastructure and resources whenever necessary, such as adding new functions, updating API Gateway configurations, and other related tasks.

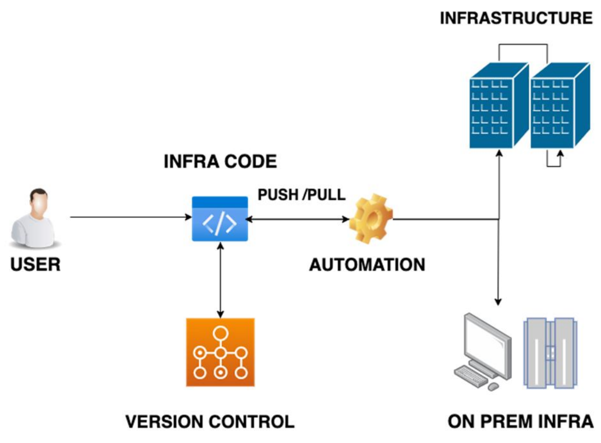


Fig. 5 Basic Iac representation

A. Teraform and Its Modules for Serverless:

Modularization is an essential feature of Terraform, allowing developers to organize and manage their serverless infrastructure as a collection of reusable components. A Terraform module is a collection of resources, data, and other elements that can be used together to define a specific piece of infrastructure. Modules can be shared and reused across different projects, making it easier to manage and maintain complex serverless applications. The first step in managing serverless resources with Terraform modules is to create a module for the serverless resource to be managed. This module should contain all the necessary resources required for the serverless resource, such as IAM roles, API Gateway, and Lambda functions. Next, inputs for the module should be defined, allowing variables to be passed into the module, making it more flexible and reusable. Examples of variables that can be defined include function name, runtime, and handler. Outputs should also be created for the module, allowing the resources created by the module to be accessed from other parts of the Terraform configuration. For instance, an output for the Lambda function's ARN can be created, which can be used in other parts of the configuration. Once the module has been created, it can be reused in other parts of the Terraform configuration, making it possible to use the same Lambda function module in multiple environments, such as production, staging, and development. If changes need to be made to the serverless resource, the module can be updated, and the changes applied using Terraform, with the changes being propagated to all the resources that use the module.

B. Optimization of Serverless resources

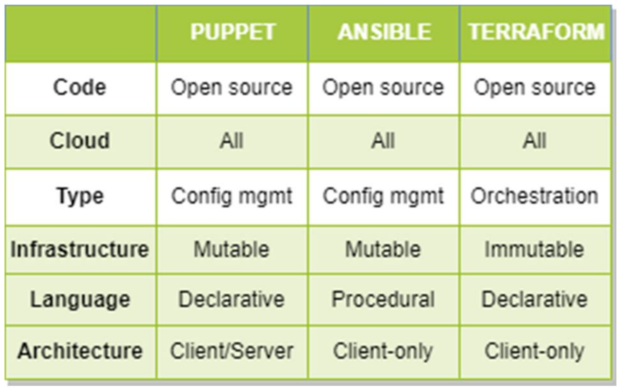
These are some recommended approaches to maximize the performance and cost-effectiveness of serverless resources through the use of Terraform:

1) To optimize serverless resources for performance and cost using Terraform, it is crucial to right-size them. This involves selecting the appropriate size and configuration for the resources, based on the anticipated workload. Terraform enables the use of variables to define resource sizes, while tools like AWS Trusted Advisor can assist in identifying underutilized or overprovisioned resources.
2) Autoscaling: is a crucial aspect of optimizing serverless resources for both performance and cost. Autoscaling policies)) automatically adjust the number of resources based on the current workload, ensuring that the resources are appropriately sized to handle the workload without incurring unnecessary costs due to overprovisioning.
3) Cold start optimization:To mitigate the issue of cold start in serverless resources, which causes the first request to take longer to process, you can use strategies like pre-warming the resources or leveraging a warm start cache. Terraform provides the ability to define configurations for pre-warming and warm start of serverless resources, which can help improve cold start performance.
4) Monitoring and alerting: Effective monitoring and alerting are crucial for optimizing the performance and cost of serverless resources. Using Terraform, you can define monitoring and alerting configurations using tools like CloudWatch or DataDog. This enables you to proactively identify performance issues and take corrective measures before they affect end-users, ensuring the smooth running of your serverless applications.
5) .Serverless-specific optimizations:Optimizing serverless resources is crucial to improving performance and reducing costs. Specific optimizations can be applied to serverless resources, such as using smaller function sizes, leveraging native language libraries, or utilizing a serverless-specific database like DynamoDB. These optimizations can be defined in Terraform configurations to ensure consistent deployment and management of serverless resources

By implementing these best practices, you can effectively optimize your serverless resources for both performance and cost using Terraform. This approach ensures that your resources are efficiently utilized and cost-effective, while still maintaining high-performance levels

C. Terraform vs Others

Terraform sets itself apart from other tools and frameworks for managing serverless resources by providing the capability to manage the entire infrastructure stack, including serverless resources, virtual machines, databases, and networking components. This makes it an ideal choice for organizations with complex infrastructure needs, as they can use a single tool to manage their entire infrastructure stack rather than relying on multiple tools and frameworks.



	PUPPET	ANSIBLE	TERRAFORM
Code	Open source	Open source	Open source
Cloud	All	All	All
Type	Config mgmt	Config mgmt	Orchestration
Infrastructure	Mutable	Mutable	Immutable
Language	Declarative	Procedural	Declarative
Architecture	Client/Server	Client-only	Client-only

Fig.6 Comparision of teraform

Other tools such as the Serverless Framework and AWS CloudFormation are more specialized in managing serverless resources, which makes them easier to use for developers who are solely focused on building serverless applications. However, they may not be as suitable for organizations with more complex infrastructure requirements that require management of a wider range of resources beyond serverless

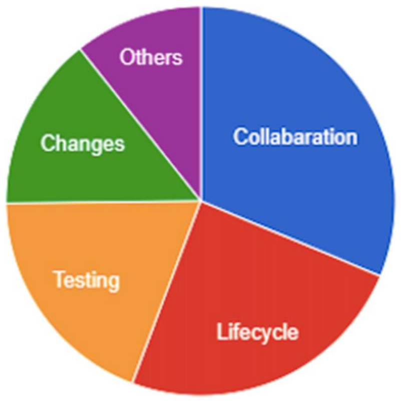


Fig.7 Major Benefits for Iac user

IV. CONCLUSION

Infrastructure as Code (IaC) promotes improved collaboration between development and operations teams, leading to fewer errors and increased efficiency. It improve efficiency by reducing troubleshooting times and minimizing errors between different stages of the development lifecycle. Even a small configuration error can cause significant delays and extend the release time. As mentioned earlier Dev and Ops teams have identified development, testing, and configuration as the most challenging stages in the software delivery process. IaC helps to alleviate the difficulties associated with these stages, which are interdependent and can lead to delays in delivery. By removing friction from these stages, IaC can improve delivery timelines, enhance efficiency, and reduce errors.

By using IaC, organizations can accelerate their problem resolution process, deploy new versions, and improve applications in a more efficient and dependable way. With IaC, companies can carry out more frequent testing of new code, ensure more dependable configuration of components, and minimize the time spent waiting for another team member to finish a task within the software development life cycle.Expanding there ability to reach new markets and customers and ultimatemy improving customer experience.

Organizations that don't use IaC have limited time to focus on new initiatives as they spend more time making adjustments to existing applications. In contrast, companies that leverage IaC can be more proactive in pursuing new ventures as they prioritize faster software delivery. A recent survey revealed that 44% of IaC users cited faster software delivery as a crucial factor in entering new product areas, compared to only 25% of non-IaC users.

The integration of multiple teams can allocate more development resources towards new business initiatives with improved team productivity and performance.

Both organizations that have implemented IaC and those who are considering it recognize the core benefits and the positive impact it can have on their business. According to a survey, organizations that have implemented IaC have reported improved customer satisfaction, expanded business capabilities, and maintained a competitive edge in their markets. Improved collaboration was identified as the most significant benefit by both IaC users and those who plan to adopt it in the future.

While there are many benefits to using serverless and IaC, there are also some drawbacks that must be considered. One major challenge that companies face is overcoming key organizational and business constraints

The implementation of IaC may face challenges such as budget restrictions and resistance to change within the organization.

One potential obstacle to adopting IaC is the lack of in-house expertise required to understand and manage the implementation effectively.

Companies may also face challenges related to a lack of appropriate tools, skills, and concerns about losing control when implementing IaC

V. FUTURE SCOPE

The field of serverless computing is relatively new, and therefore there are numerous opportunities for the research community to explore and address.

1) *Opportunities for Research at the System Level:* Minimizing the cold start issue while maintaining zero scaling is crucial, and the research community is exploring various techniques. There is a fundamental inquiry as to whether containers are the appropriate abstractions for running serverless applications or if smaller-footprint abstractions like unikernels are better suited. Further research needs to be carried out in this domain.
2) *Stateful Serverless Application*: Most serverless platforms currently do not support stateful applications, which raises questions about whether there will be a need for inherently stateful serverless applications in the future that can provide different degrees of quality-of-service while maintaining scalability and fault tolerance properties. This is an open question that remains to be answered, as it requires a deeper understanding of how stateful applications can be designed and operated in a serverless environment. While there are some emerging solutions and workarounds for stateful scenarios, such as using external databases or messaging systems, they often come at the cost of increased complexity and potential performance issues. As such, there is a need for further research and development in this area to explore how stateful applications can be effectively supported in a serverless paradigm
3) *Novel Serverless use cases and Applications*: Scientists are exploring the use of serverless technology for various applications, such as Pywren and ExCamera for scientific computing, high-performance computing, numerical analysis, and AI chatbots. These examples demonstrate the versatility and potential of serverless computing in advancing research and development in various fields.
4) *Decomposition of Legacy Code* :A critical challenge for businesses moving to serverless computing is determining how much of their existing legacy code can be automatically or semi-automatically decomposed into smaller, more granular pieces to take advantage of the cost benefits of serverless architectures. Legacy code was designed for traditional computing architectures and may not be easily decomposed into smaller functions.
5) *Serverless in Edge Computing Environment:* AWS Greengrass is an extension of Amazon's serverless capabilities to an edge-based cloud environment. This allows code to run not only on embedded devices but also in the cloud and enables virtualization, which allows for the movement of code between devices and the cloud. This may create specific requirements that redefine the cost of serverless computing. This could be another area of intrest in future.

REFERENCES

[1] Rahman Akond and Williams Laurie. 2019. Source Code Properties of Defective Infrastructure as Code Scripts. Information and Software Technology (2019). https://doi.org/10.1016/j.infsof.2019.04.013
[2] Anonymous Authors. 2020. Dataset for IaC Testing Practices Paper. https: //figshare.com/s/605c5b636450a29f420e

[3] Yevgeniy Brikman. 2018. 5 Lessons Learned From Writing Over 300,000 Lines of Infrastructure Code. https://www.hashicorp.com/resources/lessons-learned300000-lines-code/. [Online; accessed 20-Jun-2020].

[4] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. Educational and psychological measurement 20, 1 (1960), 37ś46.

[5] Wikimedia Commons. 2017. Incident documentation/20170118-Labs. https: //wikitech.wikimedia.org/wiki/Incident_documentation/20170118-Labs. [Online; accessed 27-Jan-2019].

[6] Benjamin F Crabtree and William L Miller. 1992. Doing qualitative research.. In Annual North American Primary Care Research Group Meeting, 19th, May, 1989, Quebec, PQ, Canada. Sage Publications, Inc

[7] Kief Morris. 2016. Infrastructure as code: managing servers in the cloud. " O'Reilly Media, Inc.".

[8] Mohamed, "A history of cloud computing," Computer Weekly.com, 2018. [Online]. Available: https://www.computerweekly.com/feature/A-history-of-cloudcomputing

[9] Google, "Serverless," GCP, 2018. [Online]. Available: https://cloud.google.com/serverless.

[10] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In Proceedings of the 13th International Conference on Mining Software Repositories (Austin, Texas) (MSR '16). ACM, New York, NY, USA, 189ś200. https://doi.org/10.1145/2901739.2901761

[11] Will Thames and others of Redhat. 2018. Ansible Lint Documentation. https: //docs.ansible.com/ansible-lint/.

[12] R. Jabbari, N. bin Ali, K. Petersen, B. Tanveer, What is devops?: A systematic mapping study on definitions and practices, in: Proceedings of the Scientific Workshop Proceedings of XP2016, XP '16 Workshops, ACM, New York, NY, USA, 2016, pp. 12:1–12:11. doi:10.1145/2962695.2962707. URL http://doi.acm.org/10.1145/2962695.2962707

[13] K. Ikeshita, F. Ishikawa, S. Honiden, Test suite reduction in idempotence testing of infrastructure as code, in: S. Gabmeyer, E. B. Johnsen (Eds.), Tests and Proofs, Springer International Publishing, Cham, 2017, pp. 98–115.