Cloud Infrastructure Self Service Delivery System using Infrastructure as Code

Ankita Dalvi
Cloud Engineer
Datametica Solutions
Pune, India
anki27899@gmail.com

2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS) | 978-1-6654-6200-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCCIS56430.2022.10037603

***Abstract*—To succeed in cloud adoption and thrive in multi-cloud environments, most organizations develop centralized platform teams. The primary function of these centralized platform teams popularly known as Cloud Center of Excellence (CCoE) teams is to industrialize application delivery. The platform teams which are made up of individuals who provision, run and manage shared infrastructure in cloud environments across teams, act on the requests generated by developers. This activity is time-consuming and prone to manual errors. This paper defines a system that enables developers to access platform capabilities on-demand using templates via a self-service mechanism. Application delivery in cloud environments may evolve into proven repeated patterns useful for standardizing how people function and what tools they consume and which process they follow. These standardized steps can be converted into a template made with infrastructure as Code. Using this system, the developers can launch their required resources using templates without intervention from platform teams while adhering to internal compliance of the organization. These templates built using Infrastructure as Code (IaC) by the platform teams make it easy to create new resources in the cloud and release new capabilities more quickly; ensuring that desired configurations are set by Infrastructure, Network, Security, and Operations Engineers.**

Keywords—DevOps; Cloud Infrastructure; Multi-Cloud; Cloud Application; Terraform; Compliance

I. Introduction

Cloud has become a go-to choice for organizations to deliver new solutions. As the cloud architectures of organizations have matured, they have become more complex to manage and dynamic to tackle. This demands businesses run their workloads using the most efficient and innovative environment. Organizations that have attained a certain maturity working on the cloud are further tuning their processes, people, and tools. They are establishing centralized teams to manage platforms that allow for smooth cloud adoption enterprise-wide and help with scaling. With cloud deployments being the norm, these centralized teams are required to deliver several standardized workflows compliant with the organization's internal policies. These deployments require shared infrastructure, runtimes, or services in the cloud. To deliver these standardized workflows with minimal manual interventions and execute them as a self-service this paper defines a system that would enable platform teams to create templates with IaC to provision the required resources.

II. Dominance of Multi Cloud Adoptions and Infrastructure as Code

As existing cloud ecosystems of organizations have matured, many organizations have found there's no magic 'one size fits all' solution. As a result, the shift to multi-cloud, which was already happening, has been accelerated due to several factors like organic adoption for digital transformation, deliberate diversification to avoid vendor lock-in, and business acquisitions. According to Hashicorp's 2021 State of the Cloud Survey Report, 76 percent of IT enterprises have embraced a multi-cloud strategy. The report also suggests that the shift to a multi-cloud environment is a dominant strategy that most enterprises are adopting [1]. Hence, it's evident that organizations prefer multi-cloud.

This demands businesses to run their workloads using the most efficient and innovative cloud strategy that can optimize performance, provide unique advantages over traditional practices, and allow for evolution-based growth. Infrastructure as code has become a crucial part of cloud computing. It frees professionals from performing manual, error-prone tasks. Plus, it reduces costs and improves efficiency at all stages of the DevOps lifecycle. The report states that multi-cloud is tightly connected to other modern technology practices like Infrastructure as Code [1]. There is a correlation between the adoption of multi-cloud and infrastructure as code and their utilization is expected to grow across organizations of all shapes and sizes.

III. Establishing Centralized Teams for managing Cloud Infrastructure

Operational challenges hinder cloud and multi-cloud adoption. As cloud environments mature, their complexity increases. With frequent cloud deployments becoming the norm, ensuring strong governance and security of resources deployed across different cloud platforms at every aspect is an important focus point for many organizations. Multi-cloud tooling is becoming essential for managing cloud resources as it involves tracking and maintaining numerous resources. To tackle this, organizations are increasingly establishing central cloud teams and Centers of Excellence as shown in Fig. 1 to best leverage in-house expertise, especially in managing and configuring cloud resources [2]. These centralized teams would perform better utilization of cloud environments whilst ensuring internal compliance policies are being adhered to.

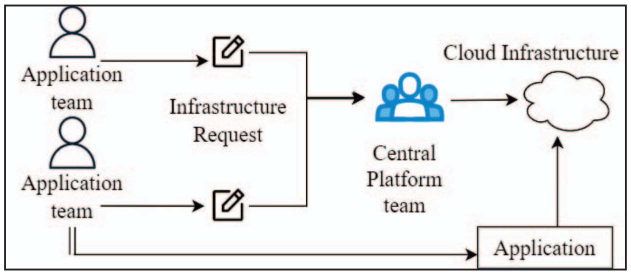

Fig. 1. *Central Platform team for managing cloud infrastructure.*

IV. Standardizing Application Team Workflows

Heritage IT projects have a definite start and end date but that is not the case with cloud platforms. A cloud environment is never truly finished. Digital transformations with the cloud are always a work in progress. One of the most significant challenges for a platform team is delivering a single path to production that suits the needs of many development teams and maintaining visibility on the utilization of resources required. However, each development team builds applications with a distinct blend of tools, backing services, target runtimes, deployment frequencies, and operational considerations.

Traditionally application teams request infrastructure and the IT teams provision them. The details of provisioning may not be germane to development teams' primary interest and hence may also be typically abstracted away. As the application grows or the team expands some of these requests could turn repetitive in nature. The repetitive nature of the requirement could be because it is short-lived or simply, just made frequently. The important goal for such cases of repetition of infrastructure provisioning is to identify a pattern in the delivery of such requests, providing platform teams with a way to plan and provision resources efficiently.

V. Cloud Management and Avoiding Misconfigurations

Cloud security breaches are common today and often stem from cloud resources' misconfiguration. Misconfigurations can happen easily, and often by accident [3]. Today's cloud environments of IT enterprises are large, which makes it tough to track and maintain tens of thousands of resources and accounts. Mismanagement of multiple connected resources like instances, serverless functions, databases, containers, etc could lead to misconfigurations. Misconfigurations are often a result of lack of visibility and not fully understanding which resources are interacting with one another, and therefore being more permissive than required [4]. Configurations, if set by developers, might be too liberal and they might even lose track of critical assets. Such misconfigurations put the organization at significant risk.

With the proliferation of resources in the cloud, many enterprises lack awareness of assets that are active, inactive, or orphaned, and how they have been configured. This lack of visibility can cause misconfigurations to go undetected for long durations, making it difficult to secure the cloud environment. This is more challenging for multi-cloud environments because even though the controls are somewhat similar for each cloud platform, the deployment approach, architecture, tools, and processes differ. Since cloud environments have become vast, and are required to comply with specific organizations' cloud security and governance policies, ensuring that the configurations are set 'just right' could be tricky. Manual management is unreliable, leading to automation as a solution.

VI. Existing System

Requests for platform teams could be simple requirements like creating similar instances for scaling, adding a DNS record entry, requesting roles for an identity, or creating a service account. In the existing system, as shown in Fig. 2 requests from the application team are taken up by the platform teams and have an operational delay. This delay arises between the time the request was made and the infrastructure was made available. Operational delay is dependent on the complexity of the cloud environment and efficiency of the platform teams in a particular organization. The operational delay is also dependant on the methods used to deploy cloud infrastructure i.e. manually or using IaC. Using IaC provides automation and is far more reliable than making changes manually [5]. IaC languages like Terraform provide modules that can be used as reproducible code. However, the code to utilize that module still needs to be added by the platform team for each new request. The code is centrally managed by the platform teams. This code has to be in terms with the organization's shared infrastructure, allowed configurations and internal policies, and calls for proper documentation. The added code must also be reviewed and approved each time the resources are to be provisioned as the CI/CD workflow to execute this code has definitive steps.

VII. Proposed System

The two stages of using IaC are definition and provisioning. The 'Definition' is to code the infrastructure resources that will be provisioned on a cloud platform. 'Provisioning' employs IaC tools to execute the code and hence, orchestrate cloud infrastructure provisioning [5]. In this paper, the focus is on the IaC approach in terms of provisioning the cloud infrastructure resources. The proposed system shifts the onus of provisioning from the admin to the application team.

The proposed system as illustrated in Fig. 2, introduces a self-service mechanism that allows platform teams to automatically route all requests from application teams through pre-approved IaC templates. This eases the process to deliver on-demand access to infrastructure. In this process, the decoupling of the template-creation process done by the platform team and provisioning done by the developers reduces the time taken for any application to meet its resource requirements, since developers do not depend on the infrastructure team as long as they use a pre-approved template. This results in the reduction of misconfiguration risk by enabling a single workflow regardless of who provisions the cloud resources whilst improving operators' productivity and increasing the developers' agility.

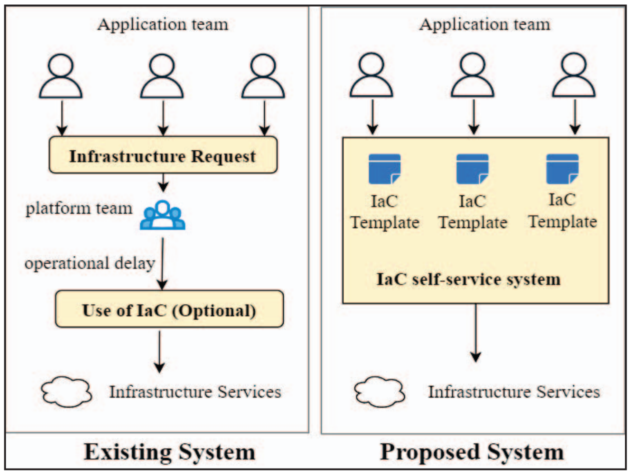

Fig. 2. *Existing System and Proposed System*

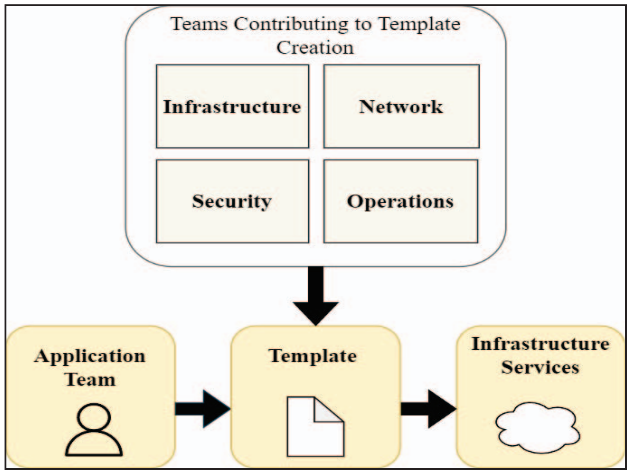

Fig. 3. *Teams contributing to Template Creation*

VIII. IaC Template Creation

The definition process ideally impacts all teams in the enterprise that decide how the cloud environments are set up and monitored i.e. infrastructure, security, networking, and operations [6]. It proves effective to deliver the dynamic services necessary for rapid application delivery while making no compromises on maintaining a strong security posture and operational efficiency. That's why enterprises should include all the required teams to create the templates and review the configurations being set. For effective infrastructure provisioning, platform teams should start by implementing reproducible infrastructure as code practices, and then layering compliance, security, and governance workflows to enforce appropriate controls. This combination enables the consistent delivery of cloud infrastructure while meeting necessary compliance, security, and networking requirements and thus, avoiding any misconfigurations. Since the templates would be utilized as inputs in the GUI-based application the variables declared should be carefully defined, hardcoded, or allowed for customization by the users consuming these templates without being experts in the matter of setting configurations for provisioning resources.

These IaCs can be written in Terraform, YAML format, JSON format, etc. Here's an example of creating a template using the popular choice of IaC i.e. Terraform. The CCoE teams can create a Terraform resource that acts as a module and expresses the configuration of services from one or more providers. The module has the resources to be created and only certain variables are to be passed to instantiate the module. Finally, a module that makes the 'template' can be extended with all teams responsible for template creation to include monitoring agents, application-specific workflows, security tooling, networking, databases, and more. Once defined with the best practices across all the required four teams as shown in Fig. 3, the templates can be provisioned as many times as required in an automated way. In doing so, Terraform template becomes the bridge language for teams provisioning resources to extend the platform capabilities to various application teams.

IX. Sample Use Cases

Several requirements and tasks performed by the developers become a chore or follow a repeated pattern and thus, become an added dependency on the team that provisions the infrastructure. Using IaC, tasks could be automated partially or entirely.

Here is a use-case: in an organization, each data scientist needs their own instance to build a model. However, once the model is built, the instance is to be terminated and the accesses given to them to perform this task are no longer required. If this task is delegated to the IT team that provisions this infrastructure in the cloud, it could lead to delay. Repetition of the same task reduces productivity and may lead to increased costs from over-provisioning, unused or orphaned infrastructure. Also, if the data scientist were permitted to create instances themselves, then there's no stopping on what parameters they would choose to create the instance. It would be challenging to assure if it is being run on the right subnet, the right region, or is of the right compute size. To prevent a scenario like this, the platform team creates an IaC template that has the generic code to create an instance. In this template, certain configurations like the subnet, region, RAM size, etc are fixed by the platform team, and liberty is given to the template consumer, here, the data scientist, to choose certain parameters or customize them, like zone, instance name, disk size, etc. This template as a whole is one solution that not only creates the infrastructure but also has the code that gives access to the data scientist on the instance, and other required services like databases, images, or storages. Post the usage, the template consumer can view, update and delete the required created resources using the same template in this service.

Another use-case: developers on the team are configured to have access to the non-prod environment and similar or fewer accesses to the prod environment. They may also have fine-grained access to their instances, datasets, storage units etc. However, when a new joiner is to be on-boarded the platform team has to manually confirm that similar access to the cloud environment is given to the new joiner as the rest of the team. This activity could be prone to errors and could also add a dependency on the platform team. A template could be created which has the code that provides access. Here, during the template creation the roles and the scope could be fixed in the code whereas the principal to whom the role is to be given remains variable. Hence the new joiner would have to enter their principal/identity in the GUI of the service and once the template executes the accesses would be given to that principal. Similarly, when a developer quits, this access can be revoked from the system by destroying the resources in the template with a simple click.

X. System Design

As shown in Fig. 4 the system enables platform teams to create identical cloud resources for multiple clouds from the deployment of a single source code template in the code repository. A common use case is by creating such common source codes, the platforms teams can standardize cloud infrastructure. The cloud admin defines the template and application teams consume it. Configurations are maintained in a file and versioning of the templates' code can be done which makes it easier to introduce newer features without disturbing the previous ones. This enables the template consumers to migrate to a newer version at their convenience resulting in quicker and safer releases.

To consume the templates the consumer has to pass variables through the GUI of the tool and then they are validated by the Populator. The Populator would enter the variables by the consumer into the configuration file. The build tool combines the variables passed by the user into the configuration file by setting them as environment variables and then executes the source code on a container that has the packages and dependencies installed to successfully run the code. Every deployment done using a template is unique per developer and the information regarding the configuration of the resources deployed using the template, the principal deploying it, the version of the template, and when it was deployed is retained in the system. Cloud admin has added visibility to see how many people have used it, what configurations they have deployed, and when the resources were terminated.

Platform teams that desire more organization can use Collections to group the templates for sharing them easily across different teams. Roles provided on the templates help to further restrict who has access to what resources. This system bridges the gap between the application team and infrastructure deployments. The workflow lets the infrastructure teams collaborate with other teams to create, manage, and consume pre-approved products in a centralized way. This service becomes a one-stop shop for IaC deployment by self-service for multiple clouds. This provides developers with the flexibility to request resources on any cloud in one central tool.

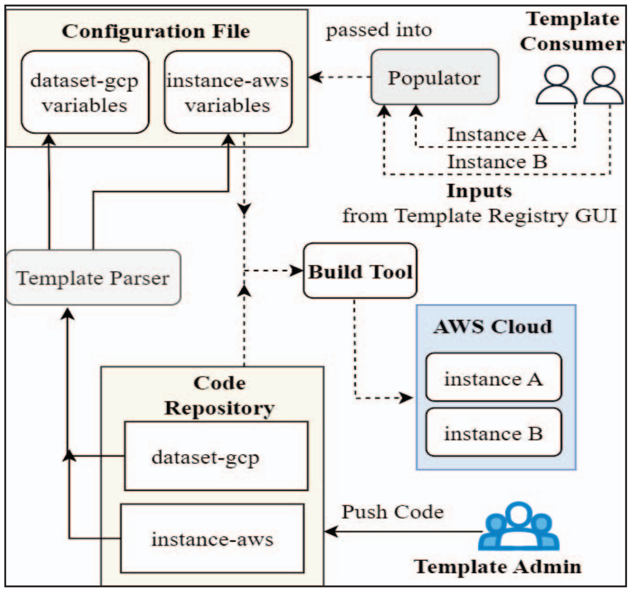

Fig. 4. *Architecture Diagram*

XI. Components of the System

A. Template

A template is a file that contains the reproducible Infrastructure as Code. A code with a cloud resource or multiple connected cloud resources, with certain parameters preset by the template creator and certain parameters set as variables. Templates would be utilized by several consumers to create their own unique deployment by passing values to the variables.

B. Template Registry

Template Registry is where all the templates reside. A library of approved infrastructure made up of versioned and validated templates to be consumed for on-demand provisioning. Template Registry as shown in Fig. 5 is simply a mapping of the UI of the templates to the source code which resides in a private repository.

C. Template Collection

Template Collection helps to categorize various templates in groups as shown in Fig. 6. A single template can be a part of one or more collections. Collections simplify sharing of several templates to different application teams who serve different business use cases.

D. Template IAM

To provision resources using a template, authentication will be done using Identity awareness. Role-based access control (RBAC) would be configured. According to the role provided to the user, they would be able to consume, create, edit or delete a template, or simply audit the resources created using templates.

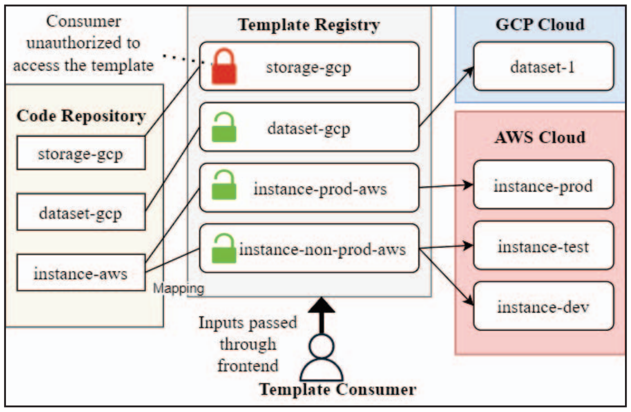

Fig. 5. *Application Diagram*

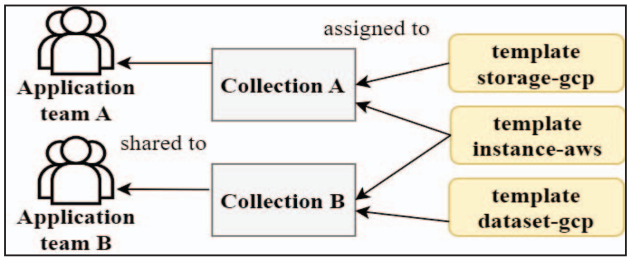

Fig. 6. *Collection and Templates*

E. Template Parser

Once the IaC is pushed to the private repository, the code is parsed and a configuration file is created by the parser that stores the variable names mapped to the provided values or default values if any. If not provided, the variables are marked to be taken as an input from the user. This file is created by the parser per template which has all the variables and their data types, required to be passed into the code.

F. Template Populator

Using the GUI, the template consumer can submit variables that will be populated into the configuration file created by the parser. Based on the data type required to be submitted, the GUI would contain non-editable elements that would refrain consumers from entering incompatible values but still provide them with visibility of the deployment configurations.

XII. Benefits of the Proposed System

A. Avoids Bottleneck of Operations

Application teams deploy the resources, identities, and access controls, as code, through pre-approved templates. This avoids the operational delay that arises between the request made and the infrastructure made available.

B. Distribution Control

Users can control or access templates with extensible role-based access controls. Role-based access control restricts access required to edit, administer or consume the Template or the Collection as a whole based on the roles given to individual users. With authentication allowed on the template level, administrators of templates can provide access only to those who require it.

C. Increased Visibility

One can browse all created resources using templates from a centralized repository along with its version. This allows monitoring of created resources and avoids over-provisioned, unused, or orphaned infrastructure.

D. Ensure Governance

Users are offered the most up-to-date, compliant, and verified IaC included in the templates which ensures that deployments follow the organization's policy. A joint effort between infrastructure, network, operations, and security teams ensures the template not only satisfies the application requirements but is approved to suit organizations' internally set rules.

E. Deployment Guardrails

Empowers Cloud Admin to limit what the end-user can deploy according to preset parameters, including machine types, RAM, regions, and more. It allows the creation of several interconnected resources in one go which could've been prone to errors if done manually multiple times.

F. Versioning

Admins have full control over the tools and solutions that are made available in the template registry. They can update the code by introducing a newer version of the template. Users can safely migrate to a newer version at their convenience.

G. Increased Discoverability

Template Registry becomes the single location for all pre-approved products, eliminating the need for internal sites, or documents.

XIII. Results

Implementation of the proposed system based on the design primarily focused on the variation in the time for the code definition and the provisioning of cloud resources. The template definition was created in Terraform for self-service (proposed system) and Terraform Module was created for Traditional IaC (existing system). As shown in Fig. 7. The system was tested to provision identical cloud resources on a scale of fifty in the experiment. The time required to provision through platform teams' intervention exceeded the time required to provision through the self-service mechanism. The primary cause of this is that each resource deployment demands the writing of code to call Terraform Module, but in the case of Self-Service, this step is skipped as the code definition is reusable in the generic template.

Code Definition in either of the methods i.e. Traditional IaC and Self-Service to make resources is carried out as a one-time activity and requires the same time. During the first time setup for the code definition, the proposed system would have an extra step to configure the inputs in Registry GUI. This additional initial setup of the template contributes to additional time for deployment. However, it is limited only to the first resource.

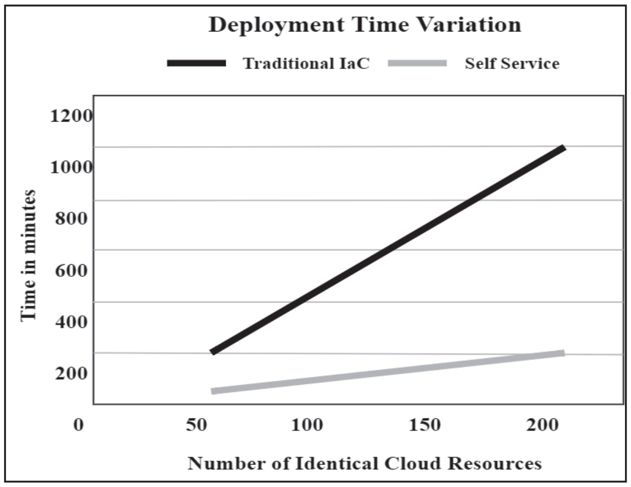

Fig. 7. *Graphical Representation for Provisioning Time Variation*

XIV. Future Scope

The proposed system tries to integrate the concept of self-service through automation which provides on-demand delivery to developers. Beyond that, there are some advanced levels of processes that can be included in the system. One of them would be enforcing a workflow to mandate template-creation teams to ensure appropriate controls for layering compliance, security, and governance. The inclusion of this additional feature will enrich the system with added security visibility and make sure all the decision makers involved in the template creation process necessarily provide their inputs.

Another feature that can be included in the system is its native integration with clouds. For this, the cloud providers would need to configure this system to work natively within the cloud console and not as a third party service. Its integration as a third party service would eliminate the need for setting up third party authentication.

XV. Conclusion

The sheer number of application processes in the cloud environment of today's enterprises imposes critical challenges for platform teams to maintain and create resources for their application development teams'. The platform teams are required to provision these frequent and on-demand requests either manually or using centralized IaC often leading to an operational delay. The proposed system would aid CCoE teams to provision the cloud infrastructure through a self-service mechanism with pre-approved templates whilst maintaining internal compliance and governance. Templates are easy to create, version, and discover using this system and they improve the productivity of platform teams by reducing their dependency. Using the defined system allows you to put controls in place due to which developers can access the resources they need by deploying the infrastructure as code specified in the template, quickly and safely. This system would have a positive impact on the existing process to provision the cloud resources by significantly reducing the operational delay.

References

[1] HashiCorp. 2022. HashiCorp State of Cloud Strategy Survey. [online] Available at: <https://www.hashicorp.com/state-of-the-cloud> [Accessed 18 July 2022]

[2] HashiCorp: Infrastructure enables innovation. 2022. Scale Your Cloud Operating Model with a Platform Team. [online] Available at: <https://www.hashicorp.com/cloud-operating-model> [Accessed 18 July 2022].

[3] Check Point Software. 2022. What is Cloud Security Posture Management (CSPM) - Check Point Software. [online] Available at: <https://www.checkpoint.com/cyber-hub/cloud-security/what-is-cspm-cloud-security-posture-management/> [Accessed 18 July 2022].

[4] D. R. Bharadwaj, A. Bhattacharya and M. Chakkaravarthy, "Cloud Threat Defense – A Threat Protection and Security Compliance Solution," 2018 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), 2018, pp. 95-99, doi: 10.1109/CCEM.2018.00024.

[5] J. Sandobalín, E. Insfran and S. Abrahão, "On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric," in IEEE Access, vol. 8, pp. 17734-17761, 2020, doi: 10.1109/ACCESS.2020.2966597.

[6] R. Rompicharla and B. R. P. V, "Continuous Compliance model for Hybrid Multi-Cloud through Self-Service Orchestrator," 2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE), 2020, pp. 589-593, doi: 10.1109/ICSTCEE49637.2020.9276897..