UPRIGHT PUBLICATIONS
https://upright.pub/index.php/ijrstp/

Original Contribution

A Review of Hosting Enterprise SaaS with IaC on Multi-cloud Platforms

Sandesh Achar[1], Nathaniel Louis Tisuela[2]

Keywords: Infrastructure-as-Code, IaC, Multi-cloud, SaaS, Enterprise Architecture, Orchestration, Containerization

International Journal of Reciprocal Symmetry and Theoretical Physics
Vol. 7, Issue 1, 2020 [Pages 14-23]

This paper explores the concepts of Enterprise Software as a Service (SaaS), the reasons for hosting a SaaS solution on a multi-cloud platform, and the benefit of using Infrastructure as Code (IaC) on a multi-cloud for quick provisioning. The paper also details the various challenges faced while adopting a multi-cloud solution. In conclusion, the paper recommends using multi-cloud and IaC to host all Enterprise SaaS solutions.

INTRODUCTION

Enterprises in the modern economy want to explore and exploit the contemporary business world is to offering their business and processes as services to other organizations in the industry. Working on shared and managed services models is becoming necessary for organizations of all sizes. Smaller enterprises are greatly benefited from this model offered by larger organizations (Atkins et al., 2010). The objective is not just to reduce the cost of the operations but also to create a second layer of business operators in the form of smaller organizations, which can further the services of the organizations to smaller pockets. The other aspect that most enterprises focus on is adopting cloud infrastructure to reduce their costs, improve their infrastructure utilization, build better availability and scalability, and enforce more robust security. The choice of the cloud provider depends on several non-functional requirements of the solutions deployed on the infrastructure (Burkon, 2013).

CONCEPTS

Enterprise SaaS

Enterprises use SaaS for two purposes: improving operational efficiency and expanding business penetration. It is possible either by using SaaS or by offering SaaS. In either of the cases, SaaS plays the first role for enterprises to embark on the wave of Industry 4.0. When an enterprise uses a SaaS, it generally looks for services in infrastructure, platforms, and standard products, like CRM. Based on these demands, most SaaS offerings are available from cloud service providers. The objective is to reduce the cost of infrastructure provisioning and management. The aim is also to improve resource utilization and increase the performance of the solutions. On the other side, some enterprises are the ones that offer their business processes and products as a SaaS for others to use them. A good example is SalesForce which offers its CRM services to others. The objective of such enterprises is to increase the usage of their products and expand their businesses by helping smaller players to become part of their business model. It can be seen as a counterpart of the franchise business model in the physical business

[1]Director of Cloud Engineering, Workday Inc., Pleasanton, California, USA
sandeshachar26@gmail.com
[2]Software Engineer, Palo Alto, California 95035, USA
natetisuela@gmail.com

world. Even start-ups offer services ranging from ticket reservations in the hospitality industry to invoicing and tax calculation services in the finance domain.

Infrastructure as Code

When planning to expose their services as a SaaS offering for others, efficient management of the resources is critical. All the resources should be controlled, monitored, and managed like all other artifacts collected. That makes it necessary to codify the infrastructure and all its resources. The new representation of the infrastructure is like any other piece of code and can be stored under an SCM and executed using an appropriate orchestration tool (Achar, 2016). IaC is an excellent way to standardize the process of provisioning and improves the efficiency of the provisioning process. In addition, it brings a high level of consistency in the infrastructure across multiple environments and reduces the differences in the configuration. However, the most important benefit from the perspective of SaaS is that the codified infrastructure can be stored alongside the application code, making it extremely easy and repeatable to build the application, create the infrastructure, and deploy the application (Li et al., 2011).

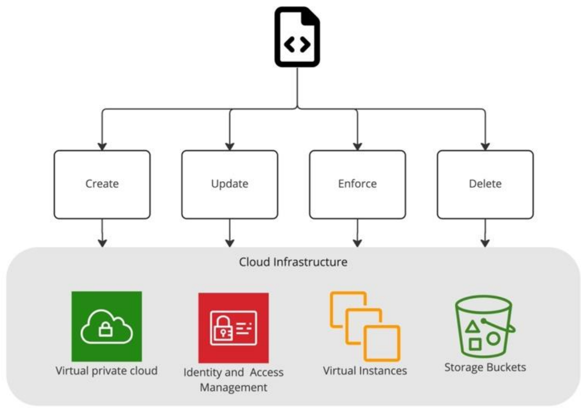
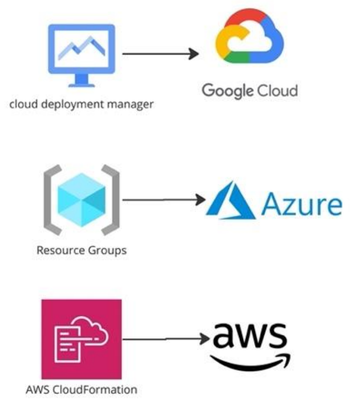


Fig. 1 High-level example of IaC

Multi-cloud Platform

Hosting a SaaS requires infrastructure provisioning on a cloud platform. The cloud may be a private, a public cloud, or a hybrid cloud. However, every cloud service provider has some or other limitations and restrictions. For example, AWS services are incredibly efficient in the North American regions; however, Alibaba cloud services are better in China and Russia. Like geographic limitations, cloud service providers experience performance degradation, network latency, and even legal restrictions in different places.

One key disadvantage cloud service users face is vendor-technology lock-in issues. As a result, applications become dependent on specific services of cloud providers and lose their ability to port across multiple platforms. This leads to cost escalation, and upgrades are problematic. Multi-cloud options are the way forward to overcome all such issues. For example, an application is hosted concurrently on the cloud environments offered by different service providers. In addition, the application uses the best services of other service providers to present a seamless experience to its users. In the world of SaaS, it helps the enterprise to cater to high scalability and availability at a lower cost by leveraging the services of the provider that are closest and most relevant to the user (Odun-Ayo et al., 2019).



.

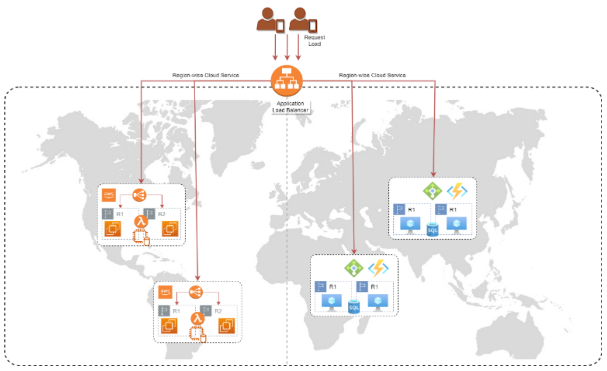
Fig. 2 Cloud-specific Observability Suites are an example of vendor lock-in issues.

OPPORTUNITIES

Enterprises require multi-cloud solutions to meet various opportunities available. Therefore, evaluating the many perspectives of using a multi-cloud is essential when launching a SaaS solution.

Region-based Factors

There are many public and private cloud service providers across the globe. Most of these service providers cover multiple regions of the world. It helps a SaaS enterprise to increase the reach and depth of its services to all areas of the world using different cloud providers. As a result, the primary cloud provider may experience high latency or complete unavailability. In such cases, another region-specific cloud provider comes to the rescue. A classic example of this case is the use of Alibaba Cloud in China and close by regions. A SaaS solution covering all areas of the world may require their services to be more efficient in China.


Request Load
Region-wise Cloud Service
Region-wise Cloud Service
Application Load Balancer
aws
R1
R2
aws
R1
R2
R1
R1
SQL
R1
R1
SQL
Fig. 3 Region-wise separate Cloud setup

Capability-based Factors

Some service providers are better in some regions of work than others providers. For example, one cloud vendor may quickly process many requests with large data transfers while another may cater to large data transfers with high efficiency. A typical enterprise-grade SaaS solution requires many services in application management, data management, infrastructure, monitoring, governance, and more. The best examples are AWS and Azure (Achar, 2017). While Azure offers one of the best cognitive services by exposing AI services as API, AWS offers better machine learning services.

Redundancy-based Factors

When launching an Enterprise-grade SaaS solution, it is essential to maintain its access, reliability, and security. Irrespective of the cloud service provider, disasters are a part of life. Every solution experiences some failure at the application or infrastructure layers. While DevOps teams take care of the application issues, the infrastructure issues are primarily centered on the cloud provider and their service SLAs. It is a great way to mitigate the risk of complete failure by redundantly deploying all solution components on a separate and different cloud. Even if a natural calamity or unforeseen conditions cause a network failure or failure, the SaaS solution can quickly switch over to the other cloud provider and continue to serve its customers. This is one of the most common reasons for the popularity of multi-cloud solutions. Furthermore, Infrastructure as Code allows the infrastructure deployment process to be abstracted into a series of scripts. Being able to deploy infrastructure in a consistent and automated fashion rapidly is an ability that is critical for disaster recovery.

Innovation-based Factors

Like capability-based factors, innovation is another critical driving factor for SaaS solutions. A SaaS solution undergoes various phases during its lifecycle. Enterprises always need to evaluate the latest trends and evolve a solution around the same. However, only some cloud vendors support the latest technology trends from day 1. Instead, they let the dust settle before offering new technology services to their customers. Google is one cloud provider that provides new cutting-edge technology-based services to their customers. In addition, it gives an excellent opportunity for Enterprise customers to keep evolving their SaaS solutions for better performance, scalability, and analytics.

Compliance-based Factors

An Enterprise SaaS solution serving customers across the globe must comply with several laws in the countries where they provide their services. Many of these compliances must use the underlying cloud infrastructure to adhere to the regulations. In such a case, many private cloud providers score better than public cloud providers. To comply with the policies and laws of the land, such as those about the storage and management of financial data or privacy-related information, SaaS operators need to partition their data and processing units across multi-cloud setups. While maintaining the integrity of the SaaS solution, multi-cloud helps the owning enterprise fully comply with the country's regulations.

Operations-based Factors

The operation methodology is also essential for selecting a multi-cloud platform. For example, some enterprises require a private cloud setup to store their Intellectual Property artifacts, such as the design documents and the code. In contrast, they use a public cloud provider to host their applications (Achar, 2018). This allows a great mechanism to keep the internal development issues separate from the production issues. Also, it will enable the implementation of different levels of security standards in different environments on completely different clouds, such as having an MFA-based VPN connectivity for the private cloud and standard token-based security for the production environment. In addition, the organization may build teams for their respective capabilities to maintain such environments on different clouds.

CHALLENGES

With the fast-increasing curve of multi-cloud adoption, enterprises experience numerous challenges. Many digital reports and surveys are available highlighting the common challenges faced regularly (Palos-Sánchez et al., 2017). Understanding these challenges upfront and mitigating their impact on the overall solution is an essential activity for every Enterprise planning to host a SaaS across a multi-cloud hosting platform.

Security

One of the biggest challenges of a SaaS-based solution is ensuring the security of the offering. A SaaS offering always suffers from the risk and threat of data loss and theft. Any vulnerability in the application or the infrastructure of the SaaS can allow unauthorized access and malicious script executions and expose the APIs in an insecure manner to the external world. Therefore, not just the PII data of the users but also the Enterprise level IP data is under threat. Even the internal communication channels are vulnerable to eavesdropping and hacking when a solution is extended across a multi-cloud platform. Furthermore, each service provider has its security realm, and it is not easy for the application solution team to align the security configuration of each of the providers. Besides, it is difficult to monitor the entire solution spread across multi-cloud because of the increased attack surface of the solution. The last challenge in the case of SaaS, as well as multi-cloud, is related to the uniform enforcement of the security policies of the organization as well as those prescribed by the law of the land. Furthermore, since the multi-cloud locations may differ, a single application may be subject to different rules about the hosting place.

Reliability

A SaaS solution is chosen over a self-hosted solution because of its cost benefits and reliability. When a standard solution may suffer disruptions and shutdowns, a SaaS solution is expected to always be online through one route or another. It is ensured using load balancers across zones and regions with the configuration of on-demand infrastructure. In the case of a multi-cloud SaaS, the availability and reliability of the entire solution rest on the strength of its integration points. However, there are complex issues with the orchestration of the application instances running on different cloud platforms or the synchronization of services running on various cloud platforms. In addition, the multi-cloud setup brings more flexibility in choices concerning services like virtual machines, storage services, database services, and more.

Observability

Multi-cloud SaaS solutions have unique challenges when it comes to observability. The method of obtaining logs and metrics of SaaS solutions may differ between different cloud vendors. Thus, tools to collect and present logs and metrics from SaaS solutions should be carefully chosen to reduce implementation and maintenance costs. Additionally, SaaS solutions may generate many kinds of metrics. For example, when developing a SaaS solution, it is crucial to be intentional about what metrics the solution should produce. Furthermore, metrics should be presented in a way that provides value for site reliability and critical performance indicators. SaaS solutions require a lot more monitoring than the standard on-premises applications. For example, a SaaS solution would simultaneously serve several customers, and a single glitch may result in loss of business or data or both for many users. Along with monitoring, a robust alert mechanism is needed to notify the administrators and support teams of any impending trouble in the application. In addition, proactive health checks need to be set up for the SaaS solution on each of the underlying cloud platforms. On a multi-cloud SaaS solution, the challenge is that more than a single tool may be needed to monitor the infrastructure provided by different vendors, and configuring a device to perform multi-platform monitoring takes work. In addition, the support teams struggle with the constant need to know what is running and what is down. While support teams can lean on cloud-specific observability suites, there is also value in maintaining a single pane of glass when it comes to monitoring. By observing a multi-cloud SaaS solution through a single pane of glass, support teams can better identify the state of the resolution, what is changing, and if any issues exist. The other challenge is that any legacy monitoring tool may need to be better for all the cloud platforms. Apart from monitoring, a mechanism for notification is needed to keep the users updated about the infrastructure's status, which means either integration with a centralized notification center or connecting with vendor-specific notification services.

Expertise and References

Setting up a multi-cloud solution requires knowledge and experience with the services of multiple cloud vendors (Achar, 2019). Although having all the required skillsets available within a single team is desirable, it is tough to hire such a team. The challenge is two-fold: identifying the exemplary services offered by the right cloud vendor and identifying the learning path for the team to get onto it quickly.

Governance

Managing the entire multi-cloud solution from a single place is one of the most challenging things to achieve. There needs to be a single authentication and authorization server to define an exhaustive set of users, user roles, and their respective permissions. A centralized mechanism would ensure the correct levels of security and control across all various cloud platforms from different vendors. Therefore, it is essential to govern the primary custody of the SaaS solution hosted on a multi-cloud platform. Governance is an area where Infrastructure as Code shines. Infrastructure as Code can enforce security policies and restrictions across all users and user roles when used effectively. Equally important, a single pane of glass for multi-cloud governance is better achieved through Infrastructure as Code because it avoids using cloud-specific resource management tools.

Automation

Automation is another excellent challenge faced by organizations on a multi-cloud solution. Automation tools must be set up to span their work areas across all the cloud vendor platforms. One challenge in automation in a multi-cloud environment is configuration management. Standard automation tools for configuration management are Chef and Puppet. Like most configuration management tools, Chef and Ansible follow a master-agent design. This requires setting up the tool in one location, configuring a master-agent serving mechanism, and running platform agents on all participating cloud platforms. Agentless configuration management software such as Ansible is available, but still, a master is required. Furthermore, automation scripts may need to be cloud-specific, especially for niche tasks. Regardless of choice, automation in a multi-cloud environment requires a great deal of expertise along with time to set up and verify. After identifying the right orchestration tool, the code pipelines need to be configured (Pasupuleti et al., 2019). It includes ensuring access to the code and artifact servers and appropriate build and deployment scripts. Orchestrating the entire process on the tool is critical to achieving good reliability and robustness.

Configuration Drift

Aqua Security, a cloud-native cyber security company, defines configuration drift as "when the configuration of an environment 'drifts,' or in other words, gradually changes, and is no longer consistent with an organization's requirements". Configuration drift can introduce several organizational problems: security, cost inefficiency, and maintenance difficulties. By nature, Enterprise SaaS typically consists of multiple components such as databases, network configurations, and load balancers. Configuration drift can be introduced as these pieces of infrastructure move through their lifecycles. In addition, moving to multi-cloud presents valuable services such as VPCs, security groups, and more. Such infrastructure may also be duplicated across different regions for redundancy or to support new customers. Consequently, detecting and managing configuration drift across cloud vendors can become a growing challenge.

Cost Optimization

Identifying the right reasons for setting up a multi-cloud environment and addressing all the challenges mentioned above requires a reasonable amount of time and cost. Apart from the cost of identifying, analyzing, and setting up everything, the most important cost center is the cost of the actual operation. Services need to be configured and operated on each cloud vendor environment, meeting the principles of cloud and SaaS solutions. In addition, organizations must define their target SLAs, performance, and availability factor to calculate their cost load (Villarino & Orea, 2012). Finally, precise planning is needed to meet the cost targets while ensuring the other KRAs are met appropriately.

INFRASTRUCTURE SOLUTIONS

One of the most significant challenges in a multi-cloud solution is setting up a standard uniform infrastructure using different cloud providers' additional services and achieving the same reliability, security, and performance levels (Rahman et al., 2019). Traditionally, most of the infrastructure activities have remained entirely manual; hence, making it uniform becomes even more difficult. Infrastructure as Code (IaC) is the perfect solution for such cases.

Push button provisioning

Infrastructure management has remained a manual activity, but it needs to be fully automated in the world of SaaS and the cloud. The best desirable state is to have a push-button mechanism to provide the required infrastructure in no time. With the help of IaC, all prerequisites and dependencies of infrastructure can be configured in a single file and executed using an orchestration tool such as Terraform or Ansible.

Standard and Uniform

IaC configuration files contain a standard definition of the infrastructure. For example, if an EC2 instance needs to be provisioned on AWS, the IaC file would have the whole purpose of CPU, memory, and storage. The same file would also contain a similar description for provisioning a VM on Azure. One standard file can help provide similar services on different cloud platforms with ease and certainty.

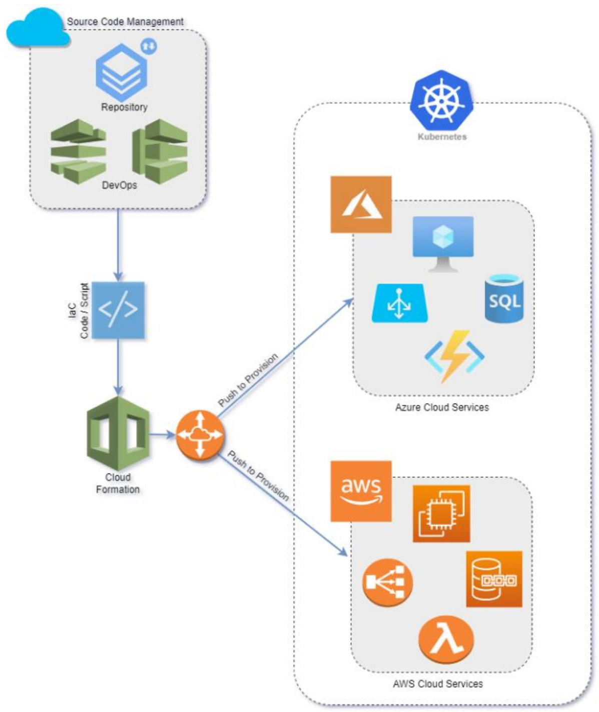


Fig. 4 Uniform IaC code pushed to Cloud platforms

Fast and Reliable

Once an IaC configuration is prepared and tested, it can be executed reliably using a tool or custom scripts on any platform. While the preparation of the configuration file may take time, the execution is generally fast and reliable. It fetches all required dependencies and creates the environment quickly without any manual interventions. In a multi-cloud environment, it reduces the difficulty of moving between different platforms for provisioning services (Venkatachalam et al., 2014).

Repetitive and On-demand

The best part of using IaC on a multi-cloud platform is that a single configuration file can provision multiple instances of the services in many environments on different clouds. It can be done as often as needed and on a need basis. The IaC concept makes it smooth and easy to repeat the same repeatedly without any problems or delay.

IAC TECHNIQUES

Using VCS with IaC Configuration Management

Hosting IaC in a repository with version control is a straightforward concept and standard in the industry. However, practicing this to use VCS to validate all IaC challenges is essential. Automating configuration management through IaC is a powerful tool to disseminate new features and bugs across a multi-cloud environment. From deploying an application to managing machine configurations, any new automation should be tested and validated in appropriate settings.

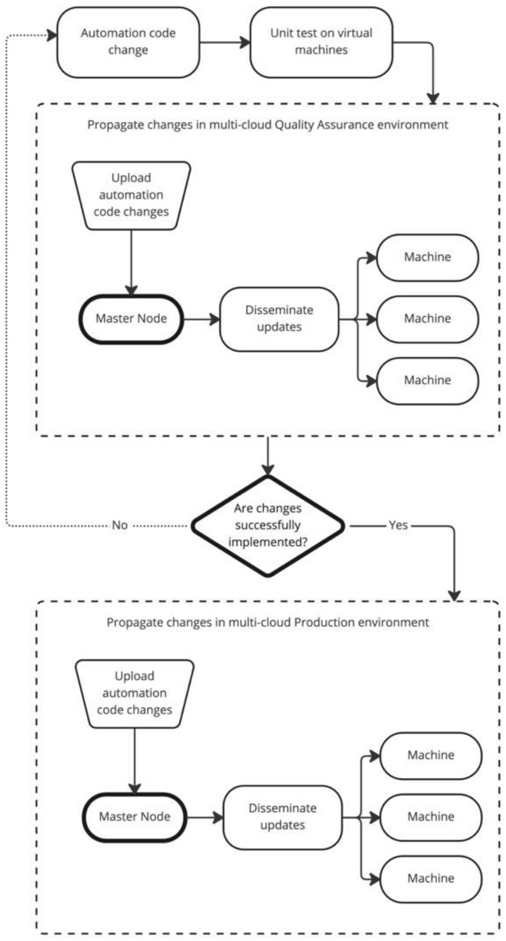

Automation code change
Unit test on virtual machines
Propagate changes in multi-cloud Quality Assurance environment
Upload automation code changes
Master Node
Disseminate updates
Machine
Machine
Machine
Are changes successfully implemented?
No
Yes
Propagate changes in multi-cloud Production environment
Upload automation code changes
Master Node
Disseminate updates
Machine
Machine
Machine

Fig. 5 An example of testing and validating configuration management automation changes

Such continuous testing should be integrated with the constant development of IaC by not allowing any production changes unless tested appropriately. This will solidify the sense of version control and provide confidence when deploying new infrastructure changes because the past, current, and subsequent versions have been validated.

Scaling Cloud Infrastructure

As previously mentioned, using declarative IaC tools for provisioning cloud infrastructure will help reduce configuration drift when infrastructure stacks are provisioned in different environments. In addition, it becomes more intuitive to deduce each infrastructure component when looking at declarative IaC. Consequently, one can easily anticipate the outcome of applying IaC resource definitions when scaling a SaaS platform in the cloud. Equally important, it is good to recognize that different environments may call for specific cloud infrastructure configurations. This is not configuration drift. A simple example is using other hostnames for additional virtual machines which serve the same purpose but in different regions. For example, instead of defining these machines separately, IaC resource definitions can be generalized and supplemented with environment variables so that provisioning can be specified for different environments. This results in less code being maintained without sacrificing the documentation of cloud infrastructure. Furthermore, this approach eases the application of relevant updates to all environments. From a SaaS perspective, this allows the platform's stack to be deployed to multiple regions to scale with a growing customer base with less code change overhead. Additionally, increases in virtual machine memory and CPU specifications can easily be updated across all regions through a generalized definition of virtual machine resources in IaC.

CONFIGURATION DRIFT AND MANAGEMENT – IAC TECHNIQUES

Configuration Drift Prevention and Detection

IaC offers measures to prevent configuration drift. Configuration drift occurs when infrastructure becomes inconsistent with an organization's requirements, defined in the documentation. IaC can become both the documentation and method of provision for cloud infrastructure. Using declarative IaC tools, such as Terraform, to provision infrastructure allows all resources to be defined clearly in code. This also helps guarantee that the infrastructure deployed in one region is consistent with infrastructure in another area. Thus, declarative IaC tools can become accurate living documentation of cloud infrastructure. However, this initiative can be defeated if infrastructure changes are made outside the IaC tool of choice. When all infrastructure changes are made through IaC, IaC becomes the historical and current source of truth of cloud infrastructure (Imam et al., 2016). As a result, IaC becomes an essential tool in preventing configuration drift by closing the distance between requirements and reality.

Of course, configuration drift can happen after the infrastructure is provisioned. For multi-cloud SaaS, detecting configuration drift across multiple cloud vendors is crucial. Cloud-agnostic configuration drift detection tools provide critical observability to this problem. For example, Terraform builds drift detection on top of its IaC foundation. By using Terraform state files that detail the expected current state of cloud infrastructure, terraform can report drift incidents across different cloud vendors. Cloud-agnostic configuration drift detection tools avoid learning overhead and fractured observability of cloud-specific drift detection tools.

Idempotency as a means against Configuration Drift

IaC is a powerful tool against configuration drift in a multi-cloud environment. At the same time, improper use of IaC can introduce configuration drift. It is crucial to consider the proper techniques for using IaC and idempotency. Idempotency is important when changes are applied using IaC. All IaC changes must result in anticipatable outcomes. If infrastructure is already at the desired state, IaC should keep the environment the same. Otherwise, unexpected changes may be introduced to the cloud infrastructure. For example, many IaC configuration management tools, such as Ansible, provide idempotency out of the box. At the same time, not all IaC changes on such IaC tools may be idempotent. Therefore, it becomes crucial to ensure the usage of IaC is idempotent. This can be achieved by writing checks in IaC to look at current configurations before applying a change.

Thus, it is an engineer's due diligence to test IaC for idempotency, even if the IaC tool of choice is built around idempotency. Terraform offers a planning stage before execution so that all changes can be reviewed before they are applied (Yadin, 2012). Ansible and Chef both offer testing suites to test IaC in different environments. Hummer, Rosenberg, Oliveria, and Eilam describe effective techniques in testing idempotency in IaC in their research paper, "Testing Idempotency for Infrastructure as Code". Their test designs for automation tasks written in Chef provide excellent examples and describe the importance of considering different environments and infrastructure states when testing IaC for idempotency.

Configuration Enforcement as a means against Configuration Drift

There are IaC tools that specialize in Configuration Management. Ansible, Chef, and Puppet are examples. Each tool can automate how a machine's software is configured. Typically, Configuration Management IaC tools provide a level of support for idempotency (Fadziso et al., 2018). Thus, these tools can minimize Configuration Drift by enforcing uniform software configuration across all machines in a multi-cloud environment.

Configuration Management IaC tools such as Chef can be beneficial in such enforcement. Chef is built around a server-client design. The Chef Server will contain a list of hosts and the user-defined procedures for configuring software in each type of host. The Chef Client runs on each host – it will pull the set of policies from the Chef Server and execute them on its host. The Chef Client will also report to the Chef Server when it fails to run the set of procedures. Finally, the Chef Client will run regularly on each host. This server-client design ensures that configuration changes are propagated from a single source of truth. Should Configuration Drift occur on a host, it will not last long -- the desired configuration state is enforced on each host because the client runs the procedures regularly.

Cloud Infrastructure Provisioning vs. Configuration Management

While the same tool can be used to automate both cloud infrastructure provision and the subsequent configuration management of that infrastructure, there are reasons why some are more effective in provisioning. In contrast, others are effective in configuration management. Most cloud providers provide their own IaC tool for provisioning infrastructure. Cloud-agnostic IaC tools that succeed in provisioning infrastructure aim to unify all cloud provider infrastructure provisioning under a uniform API. On the other hand, configuration management IaC tools seek to configure provisioned infrastructure, which includes automating tasks that may be performed on machines with varying operating systems. Both provisioning and configuration management are concerned with configuration drift and automation. However, their responsibilities in these areas differ enough that having focused tools will benefit more than a single tool to rule them all. Thus, it is good to depict a separation of concerns when using Cloud Infrastructure Provisioning IaC tools vs. Configuration Management IaC tools.

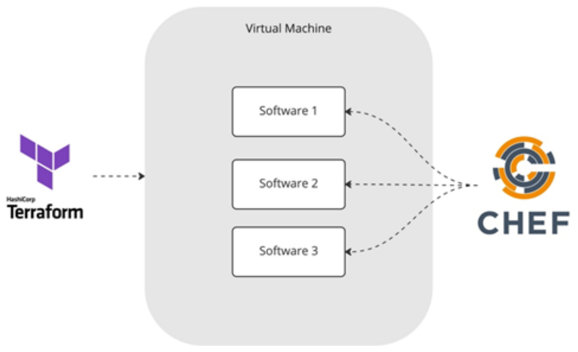


Fig. 6 Cloud Infrastructure Provisioning IaC should be responsible for the machine, while Configuration Management IaC is responsible for the software in the machine.

From there, one can understand how each tool's strength can be leveraged for automating each stage of a system's lifecycle. For example, one may argue that there are benefits of using one tool to rule them all – less time spent learning and simplified maintenance. However, automation can happen gracefully with less technical debt when the right tool is used for the right job.

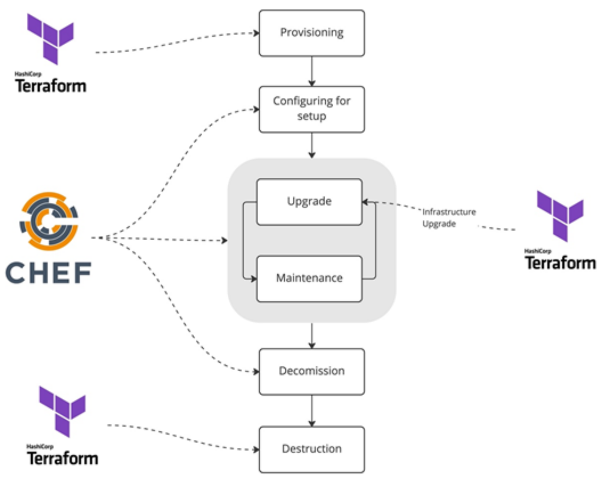


Fig. 7 By using the lifecycle of a system as context, one can be mindful of which type of IaC tool to use.

An example of successfully separating the concerns of cloud infrastructure provisioning and configuration management is using Terraform for provisioning and Ansible for provisioning.

Terraform excels in provisioning through a declarative and uniform approach to defining cloud resources across different vendors (Herrera-Cubides et al., 2019). This benefits multi-cloud SaaS solutions by lowering deployment overhead to various cloud providers while also providing a readable source of truth of all infrastructure across all used cloud providers (Lessard, 2014). Furthermore, the provisioning/ updating of each resource – such as when one resource is dependent on the other – is delegated to Terraform. This also means that ensuring idempotency also rests on Terraform itself. Therefore, the user is only responsible for defining each resource. In this way, Infrastructure Provisioning IaC tools can be used to automate provisioning, update gracefully, and destroy cloud infrastructure (Gajakosh & Takalikar, 2013)

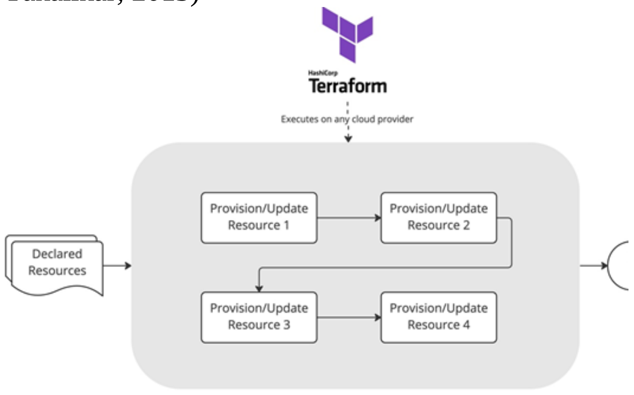

HashiCorp Terraform
Executes on any cloud provider
Declared Resources
Provision/Update Resource 1
Provision/Update Resource 2
Provision/Update Resource 3
Provision/Update Resource 4

Fig. 8 Terraform succeeds in managing the state of infrastructure.

One-way Ansible excels in configuration management through a procedural approach in defining how infrastructure should be configured. This benefits multi-cloud SaaS by making the workflow of automated configuration management visible through this procedural approach – it becomes easier to debug automation issues for different types of infrastructure. In addition, configuration Management IaC tools inherently make it easier to define a process of steps compared to Infrastructure Provisioning IaC tools. Like Terraform, Ansible also provides built-in checks for idempotency, but Ansible also gives the user the power to define specific reviews for each process step. This divergence is because terraform "understands" how to check for idempotency for each cloud vendor it deals with; therefore, the user can delegate such responsibility to Terraform. The same cannot be said for idempotency when configuring software – the user cannot expect Ansible to "understand" the software is configured. Finally, like most Configuration Management tools, Ansible makes it easy to run the same procedure for multiple hosts or conditionally run steps of each process for hosts. In this way, Configuration Management IaC tools allow users to manage infrastructure better to set up and upgrade software and other higher-level systems within the infrastructure (Hummer et al., 2013).

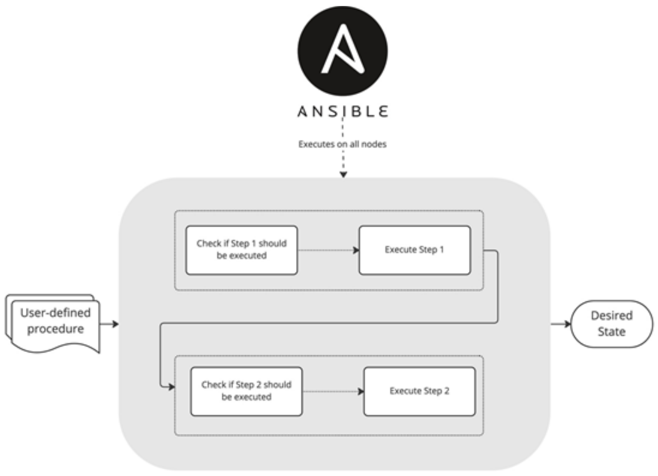


Fig 9. Ansible excels in enforcing the desired state of software in many machines via user-defined procedures.

CONCLUSION

Hosting an Enterprise SaaS on a multi-cloud is always a better option. It reduces the risks and costs of hosting while improving the end users' efficiency, performance, and experience. However, it does suffer from security, monitoring, and governance challenges. However, automation can help in optimizing the overall cost. The findings also suggest that using IaC can be highly beneficial to provisioning standard infrastructure across multi-cloud platforms easily and reliably using scripts and tools.

DECLARATION OF COMPETING INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

Achar, S. (2016). Software as a Service (SaaS) as Cloud Computing: Security and Risk vs. Technological Complexity. *Engineering International*, *4*(2), 79-88. https://doi.org/10.18034/ei.v4i2.633

Achar, S. (2017). Asthma Patients' Cloud-Based Health Tracking and Monitoring System in Designed Flashpoint. *Malaysian Journal of Medical and Biological Research*, *4*(2), 159-166. https://doi.org/10.18034/mjmbr.v4i2.648

Achar, S. (2018). Security of Accounting Data in Cloud Computing: A Conceptual Review. Asian Accounting and Auditing Advancement, 9(1), 60–72. https://4ajournal.com/article/view/70

Achar, S. (2019). Early Consequences Regarding the Impact of Artificial Intelligence on International Trade. *American Journal of Trade and Policy*, *6*(3), 119-126. https://doi.org/10.18034/ajtp.v6i3.634

Atkins, N.,Jr, Mitchell, J. W., Romanova, E. V., Morgan, D. J., Cominski, T. P., Ecker, J. L., . . . Gillette, M. U. (2010). Circadian integration of glutamatergic signals by little SAAS in novel suprachiasmatic circuits. *PLoS One*, *5*(9) https://doi.org/10.1371/journal.pone.0012612

Burkon, L. (2013). Quality of service attributes for software as a service. *Journal of Systems Integration*, *4*(3), 38-47.

Fadziso, T., Adusumalli, H. P., & Pasupuleti, M. B. (2018). Cloud of Things and Interworking IoT Platform: Strategy and Execution Overviews. *Asian Journal of Applied Science and Engineering*, *7*, 85–92. Retrieved from https://upright.pub/index.php/ajase/article/view/63

Gajakosh, S., & Takalikar, M. (2013). Multitenant software as a service: Application development approach. *International Journal of Advanced Computer Research*, *3*(3), 159-163.

Herrera-Cubides, J., Gelvez-García, N. Y., & López-Sarmiento, D. A. (2019). LMS SaaS: Una alternativa para la formación virtual. [SaaS LMS: An alternative to the virtual training] *Ingeniare: Revista Chilena De Ingenieria*, *27*(1), 164-179.

Hummer, W., Rosenberg, F., Oliveira, F., Eilam, T. (2013). Testing Idempotence for Infrastructure as Code. In: Eyers, D., Schwan, K. (eds) Middleware 2013. Middleware 2013. Lecture Notes in Computer Science, vol 8275. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-45065-5_19

Imam, M. O., Yousif, A., & Bashir, M. B. (2016). A proposed software as a service (SaaS) toolkit for cloud multi-tenancy. *Computer Engineering and Applications Journal*, *5*(2), 37-46.

Lessard, L. (2014). Designing and managing value co-creation in KIBS engagements. *Technology Innovation Management Review*, *4*(7), 36-43.

Li, D., Liu, C., & Liu, B. (2011). H-RBAC: A hierarchical access control model for SaaS systems. *International Journal of Modern Education and Computer Science*, *3*(5), 47-53.

Odun-Ayo, I., Geteloma, V., Falade, A., Oyom, P., & Williams Toro-Abasi. (2019). A systematic mapping study of utility-driven models and mechanisms for interclouds or federations. *Journal of Physics: Conference Series*, *1378*(4). https://doi.org/10.1088/1742-6596/1378/4/042008

Palos-Sánchez, P. R., Arenas-Márquez, F.,J., & Aguayo-Camacho, M. (2017). La adopción de la tecnología cloud computing (SaaS): Efectos de la complejidad tecnológica vs formación y soporte. [The adoption of cloud computing technology (SaaS): effects of technological complexity vs training and support] *Revista Ibérica De Sistemas e Tecnologias De Informação*, (22), 89-105. https://doi.org/10.17013/risti.22.89-105

Pasupuleti, M. B., Miah, M. S., & Adusumalli, H. P. (2019). IoT for Future Technology Augmentation: A Radical Approach. *Engineering International*, *7*(2), 105-116. https://doi.org/10.18034/ei.v7i2.601

Rahman, M. M., Pasupuleti, M. B., & Adusumalli, H. P. (2019). Advanced Metering Infrastructure Data: Overviews for the Big Data Framework. *ABC Research Alert*, *7*(3), 159-168. https://doi.org/10.18034/abcra.v7i3.602

Venkatachalam, N., Fielt, E., Rosemann, M., & Mathews, S. (2014). Small and medium enterprises using software as a service: Exploring the different roles of intermediaries. *Australasian Journal of Information Systems*, *18*(3). https://doi.org/10.3127/ajis.v18i3.1101

Villarino, T. G., & Orea, D. G. (2012). IMPRO4: UNA APLICACIÓN DE TIPO SAAS (SOFTWARE AS A SERVICE) PARA LA EVALUACIÓN DE IMPACTO AMBIENTAL/IMPRO4 - SAAS FOR ENVIRONMENTAL IMPACT ASSESSMENT: SCIENCE AND ENGINEERING NEOGRANADINA. *Ciencia e Ingeniería Neogranadina*, *22*(2), 179-195.

Yadin, A. (2012). Enhancing information systems students' soft skill - a case study. *International Journal of Modern Education and Computer Science*, *4*(10), 17-25.

--0--