Deep Science Publishing
https://doi.org/10.70593/978-93-49910-08-9_5

Chapter 5: Infrastructure as code and automation tools for efficient multi-cloud resource provisioning

5.1. Introduction

Cloud computing, with its features such as low initialization cost, low long-term cost, and high scalability, provides technical support for enterprises to build large-scale information systems and adopt advanced technologies such as big data, artificial intelligence, and the Internet of Things. With the rapid development of cloud computing, especially in the last decade, various cloud service providers have sprung up. Enterprises are provided with a cloud-computing platform with different service modality of Infrastructure as a Service, but they have been struggling with resource management and utilization. Given the resource management and utilization become increasingly complicated with the extremely wide variety of factors involved in the decision-making process, it has made the whole process inefficient and ineffective. Workload Management Systems are commonly used by Service Providers and cloud customers to manage the distribution of workloads among resources. However, they are designed for a single-cloud environment and cannot efficiently and effectively manage the workflow tasks in a Multi-Cloud Environment (Lakshman & Malik, 2010; Sharma & Lamba, 2020; Liu & Shao, 2021).

Due to the emergence of multi-cloud systems, it is getting common for an enterprise to utilize resources from multiple cloud service providers. And for an enterprise to efficiently and effectively utilize the resources from the Cloud Service Providers, it prefers to set up a Multi-Cloud Environment which consists of various types of resources from diverse providers. However, the heterogeneous properties of the assets, works, and service levels in a Multi-Cloud Environment with multiple assets originating from multiple providers have rendered the workflow management a much more complicated problem than for a single-cloud system. On the one hand, for a more complicated multi-cloud environment with more diverse conditions, objectives, properties, and specifications of different resources, it is getting demanding and crucial that a workflow is optimized according to its defined settings and characteristics. On the other hand, with the development of cloud-computing technologies and innovations such as storage management and deployment technologies, more and more large-scale workflows are migrated to the cloud for management and execution. It has consequently increased the pressure on the interest convergence of multiple providers in the booming multi-cloud system (Zaharia et al., 2010; Tsai et al., 2019).

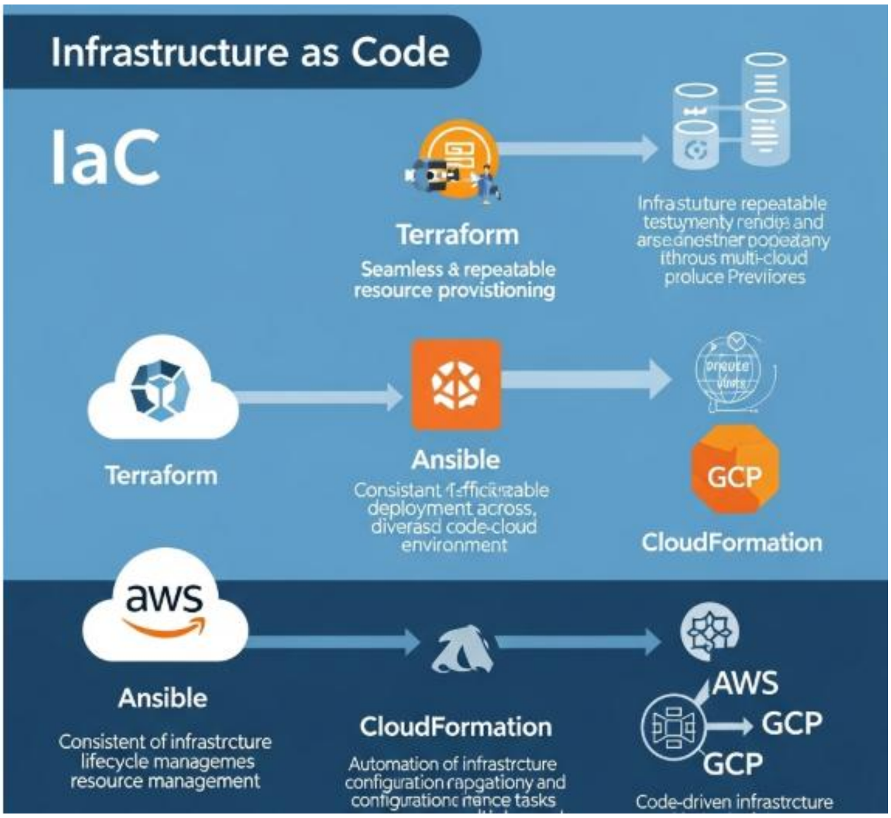
Infrastructure as Code
IaC
Terraform
Seamless & repeatable resource provisioning
Infrastuture repeatable testymenty rendys and arse.dnestner popedany ithrous multi-cloud proluce Previiores
Terraform
Ansible
Consistant 1-fficirable deployment across, diverasd code-cloud environment
GCP
CloudFormation
aws
Ansible
Consistent of infrastrcture lifecycle managemes resource management
CloudFormation
Automation of infrastrcture configuration rapgationy and configurationc rence tasks
AWS
GCP
GCP
Code-driven infrastrcture

Fig 5.1: Infrastructure as Code and Automation

5.1.1. Background and Significance

Traditionally, the creation of IT infrastructure to support application development has been both an art and a science. The vital importance of such infrastructure for application development, particularly new application development, has made it an area of investment and innovation in all types of organizations. For new application development, the requirement to accomplish development objectives using weighted combinations of timing, cost, and quality, often leads to the creation of significant technical debt. The importance of IT infrastructure in concept projects has dramatically increased. Indeed, it is difficult to imagine a concept project that would be completed by developing the application alone without using IT infrastructure. Consequently, the recognition of the critical importance of IT infrastructure for both application development and for organizational competitive advantage has been rewarded with heavy investment to reduce the burden of infrastructure provisioning, often seen as an administrative function. The evolution of cloud computing and cloud services has made it easier and faster than ever to provision IT infrastructure.

Infrastructure, in this model, simply becomes another product or service on the menu. However, reducing the burden of infrastructure has not always been an easy business decision, especially when the pressure to deliver new applications is intense. On the one hand, there is the argument that just-in-time provisioning of IT infrastructure is the best means of encouraging speed and agility in new application development. After all, every day of delay in allocating IT infrastructure inevitably adds to the cost of delay. And, it is not uncommon for sponsors of new business initiatives to constantly monitor progress for lack of focus. Burdening already stressed IT departments with complex portfolio management processes around allocations of IT infrastructure would be counterproductive.

5.2. Understanding Infrastructure as Code (IaC)

Understanding Infrastructure as Code (IaC) Infrastructure as Code (IaC) is a technology revolution marked by the introduction of high-level languages into computer science. Moreover, in the theater of cloud computing, the concept of infrastructure as code emerged allowing the management of cloud resources using programming techniques. Cloud users realized that they could manage resource creation using separate programming techniques, and later the programming team began to create their own libraries, offering support to users of these environments. Thus, libraries were created to meet the demand for resource management, exposing the language provided by cloud providers, which allowed the modeling of all or part of the capabilities offered by service providers. These libraries allow the creation and documentation of processes that execute resource management on any of the cloud providers on a scheduled basis or on demand. They expedite several actions and give consistency to the processes as they avoid failures caused by execution time since libraries contain and utilize characteristics defined for each resource during its modeling.

5.2.1. Definition and Importance Infrastructure as Code (IaC) as a concept came to widen the thinking of system administrators. In the early days of the internet, system administrators and network administrators implemented and configured all devices manually. As the number of devices grew larger and larger, they would script repetitive tasks to make their lives easier. Some years later, Configuration Management Tools (CMT) started to appear making it easier for administrators to map an entire environment and implement and manage their configurations in a simpler way.

5.2.2. Benefit of IaC The automation of these repetitive tasks, whether via shell scripts, CMT, or IaC tools, released the administrators of these smaller tasks, allowing them to focus their valuable time on what was more important to their organization: ensuring security and improving the infrastructure. With the appearance of cloud computing, many resources and services needed to be created and configured. These services required a defined configuration to have a consistent environment and avoid failures caused by misconfigurations. These configurations were delivered through the cloud providers' interfaces exposing their APIs, which allowed the creation of more sophisticated scripts. Soon, libraries were developed to wrap the APIs and started popping everywhere.

5.2.1. Definition and Importance

Infrastructure as Code (IaC) is a revolutionary approach to managing and provisioning IT infrastructure that automates the entire process, transforming manual work into an efficient code-based solution. With IaC, developers or operations engineers write machine-readable configuration files that contain the exact rules and parameters to provision and manage the necessary infrastructure for app components. The IaC approach involves three broad levels of automation, namely Client-Service, Scripted Automation, and Fully Integrated Automation. Though automation is a necessary goal of these levels, passing the task of configuring and managing the infrastructure to code is the primary motivation for adopting the IaC approach. Adoption of the IaC approach has several proven advantages, the most important ones being cost reduction, consistency, and speed.

The importance of IaC for Multi-Cloud strategy stems from the fact that the primary problem with a Multi-Cloud strategy today is that the inherent diversity across all major cloud offerings makes it tedious to provision and manage the required cloud infrastructure, especially as the number of application components and services scale into the hundreds. Each cloud provider has its own APIs for infrastructure management. This brings us back to the dreaded burden of dealing with the variability in the languages, libraries, and constructs needed to invoke the different APIs across the cloud providers. A critical capability that enables the needed automation is the ability to define infrastructure in a simple, code-based format, abstracting away the complexities of the underlying cloud provisioning architectures and implementation details. IaC is the means through which this capability is realized. This, coupled with a comprehensive Multi-Cloud resource manager, will allow us to enjoy the promise of our Multi-Cloud strategy without being bogged down by a tedious, error-prone, and inconsistent provisioning and management process.

5.2.2. Benefits of IaC

Organizations continue to adopt Digital Transformation strategies to be competitive and have scalable solutions. There are many components to successfully achieve this adaptation. Migrating Corporate Infrastructure to Multi-Cloud helps for a better dislocation and provides better services to the final customers. To operate and manage those configurations for infrastructure provisioning and other Automation services, having Automation Tools for this support is crucial. To request those services with the respective tools, having the resources managed through Infrastructure as Code is essential.

Infrastructure as Code is a Configuration Model where software development tools and methodologies for code development and maintenance are used to develop the configuration files for Infrastructure provisioning and configuration. That way it enables software development teams or DevOps teams to work together with the same tools and the same techniques they are used to. Through IaC, teams have Designs, Testing Frameworks, Version Control tools and Deployment processes for the infrastructure as they have for the application. It provides all the benefits from those techniques, to the Infrastructure configuration services.

There are many advantages for Infrastructure as Code and applying those techniques will provide several advantages to your Organization and your services. Adopting IaC will enhance Agility, Decrease Time, Improve Quality and Control Access. All of those advantages are obtained through the Automation tools adopting infrastructure as code as the way those tools will help on Building, Testing and Deploying the infrastructure requested. Besides the advantages obtained with an Infrastructure as Code, those Automation Tools will help with the Security and Compliance of your environment with Monitoring Systems for Services availability and configuration control.

5.2.3. Common IaC Tools

Among the numerous IaC tools available, the most popular are Terraform, CloudFormation, Ansible, and Azure Resource Manager Templates. Terraform: It is an open-source, low-level IaC tool that allows provisioning a wide range of cloud providers and supports DevOps workflows. Terraform has become increasingly popular because it supports the most widely used cloud providers, such as AWS, Azure, GCP, and DigitalOcean. CloudFormation: It is a high-level IaC tool that allows the declarative configuration of resources into a single template file. The major advantage of CloudFormation is that it exploits many of the unique features and services of AWS and has the greatest tight coupling with AWS, thanks to AWS being its sole provider. In addition to resources, CloudFormation also supports third-party resources available in the Marketplace. Ansible: It is an open-source project written in Python, designed for provisioning and automating deployment in both cloud and on-prem environments. Ansible uses playbook YAML files, is agentless, and supports infrastructure provisioning in a push mode. Ansible is best suited for automating deployment processes in hybrid cloud environments. Since its focus is on automation of deployments and not just infrastructure provisioning, Ansible is mostly used in the post-provisioning phase. ARM Templates: These are the building blocks of Azure Resource Manager, the deployment and management service for resources. ARM Templates create resources in a declarative way and are ideal for provisioning large and intricate interdependent architecture patterns. ARM Templates leverage Azure's unique services and features, as well as support third-party services connected with the Marketplace. ARM Templates are especially useful for deploying Azure-based microservices since they can deploy multiple instances of a microservice quickly and easily.

5.3. Overview of Multi-Cloud Environments

A multi-cloud strategy is to use multiple cloud computing providers with different cloud services. The cloud environments, services, and resources can be different: for example, mixing public cloud from a public cloud service provider, dedicated cloud in a private facility from another vendor, and private clouds instantiated in an on-premises datacenter using virtualization technology from a third vendor. A multi-cloud strategy may also be defined as the use of different cloud services from the same cloud environment. For instance, a private cloud is owned by a specific company, but it may host different cloud applications from its own private business as well as cloud applications from another organization, such as government agencies and healthcare services, in a local multi-cloud together. It also may deploy different cloud applications integrated with a public cloud owned by the cloud service provider, but should ensure a trusted multi-cloud with high levels of service quality. A company may also use multiple public clouds from different cloud service providers with complementary cloud services.

From a business point of view, the advantages of multi-cloud strategies are to avoid vendor lock-in scenarios, to support cloud bursting applications, and to lessen the risk of service interruption due to external security risks, such as cyber-attacks. From a technical point of view, the benefits of multi-cloud strategies are to achieve specific goals, such as latency and legal concerns, and to access mature cloud services offered globally from different vendors. New technology developments in specific areas with specific advantages are commonly adopted at different speeds, so that if a cloud service provider has latencies in offering a technology-enabled service capability, a business with a multi-cloud strategy can easily choose a different vendor instead of waiting. Moreover, about 80% of the most critical applications are affected by privacy rules or industry regulations that restrict the use of certain cloud services within a geographical location. Providing on-premises deployment options for these applications can alleviate those concerns.

A clear goal of Multi-Cloud Tools is to decouple the Enterprise from the dependent lockin on any CSP. Vendor lock-in avoiding is an essential argument against the Public Clouds Adoption. The gradually bookkeeping style of Cloud Services Providers, picking cloud services from every provider to achieve the best enterprise results is set to become a cloud computing standard. Multi-cloud computing refers to the strategy and outcome of utilizing and managing multiple Cloud Service Providers for a single business purpose. Multi-cloud usage can include private Cloud infrastructures, hybrid Clouds, and distributed public Cloud services. The cloud services comprise Tradeable, ranging from adjacent computing needs, but the services may be offered by different vendors.

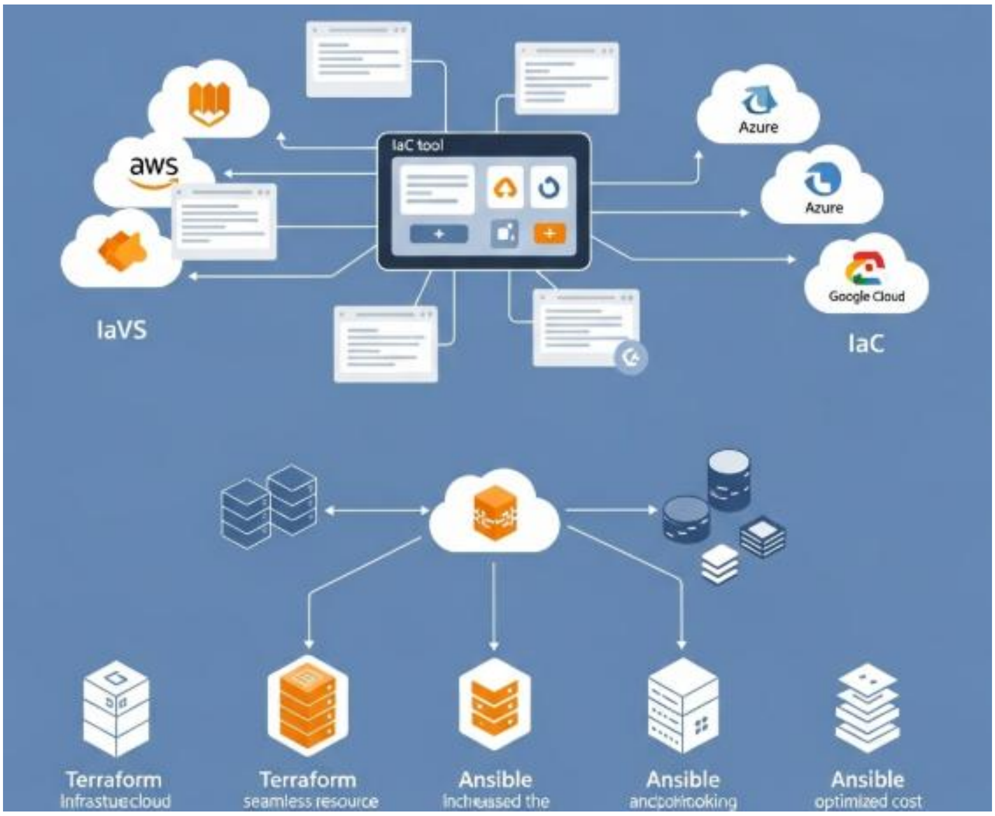

Fig 5.2: Overview of Multi-Cloud Environments

5.3.1. Definition of Multi-Cloud

Definition of Multi-Cloud A Multi-Cloud Environment is composed of Services from more than one Cloud. Some of the characteristics that usually include when we talk about Multi-Cloud are first, the integration of different Services Standards, and second, the integration of different Services Release Models. The last one is useful to identify a Multi-Cloud trigger. A Multi-Cloud environment contains private and/or Community Clouds interconnected to one or more Public Clouds at different stages of their life cycles. Moreover, a Multi-Cloud is the result of Cloud's evolution. Small Clouds Organizations, which deploy Multi-Clouds composed of a Frequency combination of Public Clouds Production, presence in Countries with no effective Public Cloud, Private Cloud that extensively use the traditional Private Cloud Services, which, because of idle variety of Resources, also Turbo Charge their Services, ideally expose Cloud Services integrated to the Multi-Cloud. Actually, one of the tingles of Multi-Cloud Appear is the increasing interest of Large Corporations in Inter-Cloud Services Adoption, where Public Cloud Providers help their Customers to integrate Services of other Public Cloud Providers. Another possible definition of Multi-Cloud is composed of a toolset of Services that allow Enterprises to import and release Cloud Services.

5.3.2. Advantages of Multi-Cloud Strategies

A multi-cloud approach will introduce some level of management complexity when orchestrating between multiple public cloud services and possibly some private cloud services. That being said, organizations opt for adding this complexity for more specialized capabilities in relation to price, geolocation, and business continuity. The capabilities of different cloud vendors are not equal in every aspect. For example, not every public cloud can offer computation services in the local region of an organization that requires very low latencies. Also, not all cloud vendors can offer optimal pricing for every single cloud service they provide. Maximizing discounts while retaining customizable cloud services capabilities is an organization advantage in a multi-cloud strategy.

Most organizations and enterprises require operational redundancy to guarantee business continuity in case one public cloud at some location goes down. A service dependency on any one cloud vendor would be high-risk exposure. During sensitive hyper-scaling operations, organizations might be dependent on the availability of more resources than those being used normally. Providers might take advantage of the situation and increase the prices of cloud resources or increase the times for resource provisioning, or even allow no new provisioning. Many organizations might require adherence to service levels agreements that can be optimized with a selection of multiple cloud vendors. During peak traffic times, the normal costs for some cloud services might be unbearable.

A business dependent on a single cloud vendor for resource provisioning could be paying extraordinary amounts for such traffic if no multi-cloud deployment is possible.

5.3.3. Challenges in Multi-Cloud Management

A multi-cloud environment, defined as the utilization of at least two public clouds from different vendors without private cloud deployment, has become increasingly popular in the last few years. Although there are benefits for organizations to adopt a multi-cloud strategy, there are still many challenges in managing multiple cloud environments. One of the most significant challenges found in a multi-cloud environment is creating and managing connectivity between the different cloud environments. Different cloud providers have different tools available to connect to other cloud environments, which is an added burden to cloud administrators. Moreover, these tools may need to be updated constantly as the interconnections between cloud environments are often patches that do not last long term, which in turn can lead to misconfigurations.

Another challenge faced by organizations that implement a multi-cloud strategy is the management of security. A multi-cloud deployment introduces multiple points of failure, with potentially differing levels of security for each service, and access to cloud resources in various cloud environments should be limited. Managing a multi-cloud environment can also be an obstacle for organizations that can experience vendor lock-in, excessive cloud expenses, and loss of visibility. They may need to reconsider their initial motives for deploying a hybrid cloud strategy as they manage their infrastructure across multiple geographic areas and services from different vendors. For instance, monitoring and managing cloud costs can be difficult as cloud providers do not have standardized pricing across all services. Either large internal teams or third-party tools need to be put in place to help organizations analyze the cloud utilization from multiple vendors.

5.4. Automation Tools for Multi-Cloud Provisioning

Automation tools facilitate the efficient management of multi-cloud environments, enabling organizations to operate across various cloud infrastructures without becoming constrained within the limitations and specificities of any single one of them. By implementing automation tools, organizations save considerable time and effort in deploying protected and secure workloads on the cloud. Moreover, since automating processes reduces the number of manual interactions, organizations boost the productivity of the teams involved in infrastructure provisioning and management because such tasks are highly repetitive and prone to human error. Multi-cloud automation software reduces complexity, cost, and risk while increasing scale, speed, and flexibility.

Automation tools perform resource provisioning either through native language processing, where the tool has its own resource provisioning language for interacting with cloud providers’ native APIs to provision cloud resources, or through agent installation on the resource. The benefit of agent installation is that provisioning tools do not have to use multi-cloud providers’ APIs, which is usually cumbersome and complex. The downside is that administration in agents is usually at a single point of failure, and provisioning tasks cannot proceed if the agent is not functioning. However, because cloud providers are now also releasing the federated API standard and its platform-independent language, which is now enabling the handling of complex requests through a single command, native language processing is being used increasingly for cloud resource provisioning. Although there are certainly more automation tools in use, we will focus on three other representative multi-cloud automation tools.

5.4.1. Terraform

Terraform is an open-source infrastructure provisioning tool that focuses on Infrastructure as Code. It is mainly developed by a company, and its popularity has been steadily climbing, especially with regard to its use in multi-cloud provisioning. Today, there are many companies and developers contributing to the development of Terraform. The advantage of Terraform is that users can take advantage of a wide variety of provider plug-ins, allowing the provisioning of a variety of resources not only on supported cloud services but also on other platforms as well. Some examples of Terraform's supported clouds include various major cloud providers.

As of version 0.12, Terraform utilizes a domain-specific language allowing infrastructure to be described as code. Terraform enables automated cloud provisioning through a series of manifest files, known as Terraform configuration files, describing a desired cloud infrastructure. The Terraform engine then parses these manifest files and makes sequential API calls to a desired cloud service provider. Using the domain-specific language, users define cloud services utilized in their infrastructure, alongside associated properties and values. The unique aspect of Terraform is how it manages an infrastructure lifecycle through state files. Once an infrastructure is provisioned, Terraform downloads the state file and periodically compares the defined state with the current cloud state represented in the state file. Not only does this aspect allow the automatic generation of an on-demand blueprint of an infrastructure, but it also allows seamless automation and management of its lifecycle. This includes the ability to destroy the complete infrastructure with a single command or command sequence.

5.4.2. Ansible

Ansible is an automation tool which can run anywhere and provide provisioning of resources in any environment. Because of its agentless architecture, it enables automation of multiple resources in different environments including locally, or remotely on the cloud, especially orchestration, configuration management, application deployment, and continuous delivery tasks. Although Ansible's primary focus is configuration management and application deployment, it can also provision cloud resources. Though it is not as efficient in provisioning as Terraform, it can also accomplish the same task. Ansible uses the YAML format along with playbook files for writing the automation code. It allows Infrastructure as Code (IaC) and codes for moving the needed part to a defined or to be defined state. The modules of Ansible allow cloud infrastructure resources to be defined, created, or manipulated. The playbooks can be validated by linters or the command is available to check these.

The modules provided by Ansible for cloud support most of the services provided by cloud providers. The API is imperfect and peculiar in its approach to the cloud, but it is manageable. Ansible provides good support for managing specific resources. In summary, Ansible is an excellent choice for managing and configuring an existing in-place or minimal multi-cloud infrastructure as it is easier and involves simple workflows compared to Terraform. It can also provision temporary resources, such as those for CI/CD, as code, which can be called from Terraform.

5.4.3. CloudFormation

Amazon Web Services CloudFormation takes a similar approach as Terraform. It enables you to implement infrastructure as code for your AWS resources. But CloudFormation is not designed for multi-cloud usage, unlike Terraform. You can use it for AWS provisioning only. CloudFormation does not only allow you to deploy existing resources. It supports updating and deleting existing resources as well. It can take an existing AWS configuration, snapshot it, and re-deploy it anywhere you want. You can express the infrastructure you want using instantiation templates written in JSON or YAML. It is recommended to use YAML instead of JSON. JSON is very verbose, making it hard to read and understand for complex scenarios.

YAML is more readable and concise. However because of the poor error feedback in CloudFormation, this is less of an issue if you validate your templates with the command-line interface. You can model your infrastructure using either standard, user-defined macros or modules in YAML. Macros allow you to customize your templates at a high detail level. Modules help you share complex configurations across your organization using a single line of code. CloudFormation also integrates with AWS

Service Catalog, enabling you to control IT costs and the cloud provisioning process. Common users deploy pre-approved services via a service catalog. There are also IAM policies that enable you to restrict who can use the CloudFormation service for resource provisioning in what way. Different access policies apply to stack sets and stack instances.

5.5. Integrating IaC with CI/CD Pipelines

Infrastructure as Code (IaC) outlines each available machine's specific configuration within the provider's API. Therefore, it only makes sense that IaC should be integrated with Continuous Integration of the application and Continuous Deployment to the production environment of the work that is provisioned. However, this is relatively rare.

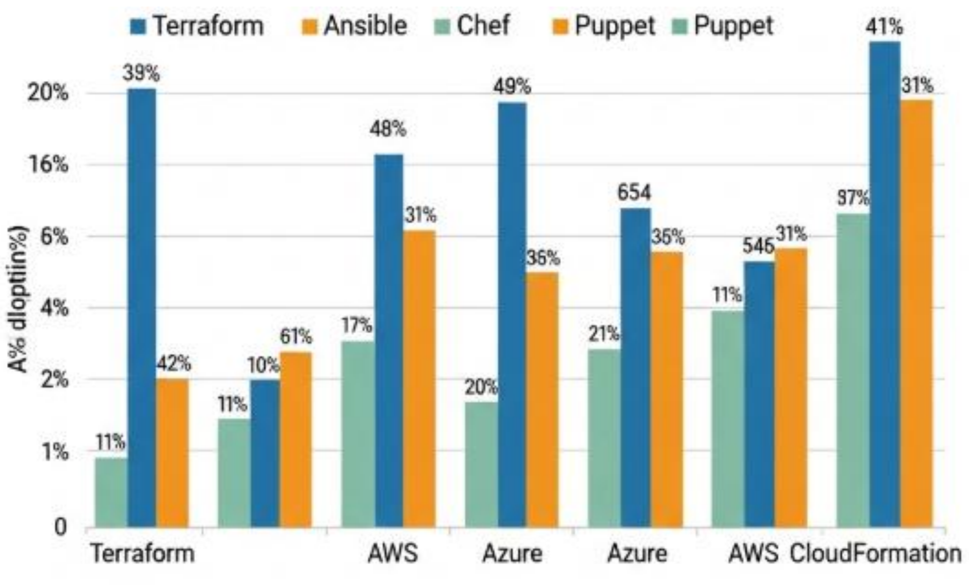

Fig : Infrastructure as Code and Automation Tools for Efficient Multi-Cloud Resource Provisioning

Often, Continuous Deployment just means deploying to a pre-staged environment with an automated upload. Typically, the IaC tooling is deployed more manually, and the investment put into continuous automation actually stands for a few commands that happen to be run by automated pipeline servers rather than the IaC tooling itself. Modern IaC tools make automation as simple as possible, enabling a truly continuous deployment process. In this section, we will provide a methodical process of successful integration of IaC with Continuous Integration/Continuous Deployment (CI/CD).

Continuous Integration (CI) and Continuous Delivery (CD) are significant practices that describe a software release process in which dynamic visual notices indicate the status of the deployed application concerning code runs. Continuous Integration (CI) and Continuous Deployment (CD) have become tenets of successful application development, allowing complex applications to be updated on a regular basis without downtime. Job queues are used to link microservices together that include the various steps of the pipeline, with services processing CI/CD pipelines into distinct steps. Triggering commits to the repository results in compilation and testing of the application, which then proceeds through different environments until being tested in Production.

5.5.1. Overview of CI/CD

CI/CD are parts of the agile development system that enhance the software deployment process by allowing developers to ship and receive updates to applications and services more frequently. Continuous integration provides for a software ecosystem in which code changes are automatically built and shipped, along with verification via tests. Continuous delivery, an extent of continuous deployment, extends on CI by having automated deployment processes. This allows for packaged applications to be pushed to production or other environments with a click or automated through a job. The CI/CD process relies heavily on automation to build, test, and deploy code.

CI/CD sandboxes allow for hardened or production-like environments to be updated out-of-band and auto-configured by the CI/CD pipelines. This can speed up the traditional waterfall method by enhancing the instant gratification theme, as an organization is able to release features, fixes, or updates in days and weeks instead of months and years. It also limits integration hell; with CI/CD, an organization does not postpone system updates that integrate with existing processes related to business needs. CI/CD pipelines allow ever-changing demands of business to be implemented efficiently and with low friction.

5.5.2. Best Practices for Integration

When designing CI/CD pipelines that will involve direct execution of IaC scripts, some best practices should be observed to reduce the associated risks and avoid movement down the DevOps maturity ladder during side effects or brute errors. A first easy and effective best practice is to include unit testing of the IaC scripts as part of the CI processes for those tools that include a dry-run, unit test libraries, or other code checkers. These tests can quickly expose syntax problems, user errors or semantic problems with few penalties and in many script languages and tools, these integrated tests tools are consistently available.

Once we have validated the IaC scripts inputs for successful execution when merging to the repository, we now can execute the IaC scripts from the CD pipelines. In this case, a second best practice is using modules to abstract entire IaaS resources that use external logical dependencies so that every IaC execution where we affect those modules can be treated as destructive. Then, when executing the steps defined in those modules, the execution tasks are independent of each other, minimizing unintended side effects. With this approach, the worst possible impact of the execution will be the entire environment error, even if the IaC execution is defined as non-destructive and the change is only in one of the module tasks.

5.5.3. Case Studies of Successful Integrations

Infrastructure code and automation services have become critical components of multi-cloud resource provisioning. Integrating IaC into a CI/CD pipeline is an effective methodology for practicing complete and efficient automation. Its successful application is a powerful reference for others. Research on the practice and integration of IaC within DAOs, such as agile software development, proposed a refined version coordinate structured or cyber phase model that highlights three interrelated assets: economic resources, human capabilities, and tooling support. The research was designed to answer two questions: how do you design and implement DevOps pipelines using IaC best practices, and what are the tools available to help you refine or iterate this design and implementation.

Real-world experience designing and implementing IaC for a variety of clients and use cases: container orchestration with Kubernetes, application microservices development and deployment, systems provisioning, and hybrid application hosting was leveraged. Each project initially used different toolchains: AWS CloudFormation, Ansible, Terraform, Packer, and Jenkins. After repeated successes using a mixture of Packer, Terraform, and Jenkins for multiple clients and use cases, the research revealed a minimal DevOps IaC blueprint toolchain that leveraged Jenkins, Packer, and Terraform.

5.6. Conclusion

Cloud computing has become a disruptive technology for all organizations, offering cost-effective, scalable, and reliable solutions to host and manage existing and new workloads. Resource provisioning in the cloud is facilitated by several automation tools and practices, such as Infrastructure as Code or DevOps. However, the automation of processes also imposes new challenges, especially in the case of large-scale, multi-cloud, and hybrid-cloud deployments. Based on the systematic analysis of selected papers, we present in this chapter the landscape of existing research on IAC and automation tools for efficient multi-cloud resource provisioning, highlighting the most important results. The proposed classification is composed of 16 classes described in this chapter, providing a guideline for researchers and professionals in the area of multi-cloud resource provisioning and offering insight into future opportunities for research in the IAC and automation tools' area.

Due to the large number and diversity of existing automation tools and technologies, we divide the options into horizontal or vertical solutions. The first category refers to ideas, concepts, and technologies that are generic and can be applied to several clouds or workloads, such as IAC concepts, tools, and technologies. The second category includes optimizations or solutions that are specific to vertical providers or cloud workloads. Such a classification is not absolute or exclusionary, and we expect our landscape to evolve in the future as new research topics and questions arise.

5.6.1. Emerging Trends

The work's focus has been on demonstrating that there are, at present, no essential difficulties in moving multi-cloud operations to an efficient script-based paradigm. The case of a project has been used as a template example, describing how the ideas and principles of decoupling cloud infrastructures from their code have been successfully applied and made widely available to the research community. Here, some emerging trends in the area are given for consideration, which will be addressed in future steps undertaken in both the work described and herein.

The first trend towards true cloud is to operate, develop, and expose code ecosystem convergence. Traditional infrastructure development and exposure and operation environments and ecosystems are today still largely disjoint from true cloud provider space and tooling ecosystems. Although the proportion of code in what amount to true cloud offerings as services is today very small indeed compared to the amount of code contained in the services directly offered by providers, it remains true that exposure not only is the last step of most providers, and certainly that of the smallest, but also is that where they have the most potential control and power. A concerted effort to make both worlds better integrated would lead to a tremendous increase in the variety of enabled services available in any given cloud environment if the traditional access-ID/keys already in use were to be extended into true cloud tool supporting cloud access through native environment protocols and service-centric APIs such as candidate Operating Frameworks and their derived templates that offer an easy, script-centric exposure path for local services of current offering.

References

Lakshman, A., & Malik, P. (2010). Cassandra: A decentralized structured storage system. ACM SIGOPS Operating Systems Review, 44(2), 35–40.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. HotCloud, 10(10-10), 95.

Tsai, C. W., Bai, Y., Huang, M. C., & Tsai, Y. C. (2019). Edge computing and AI in cloud ecosystems. Future Generation Computer Systems, 96, 417–428.

Sharma, T., & Lamba, H. (2020). AI-driven cloud orchestration using Kubernetes and TensorFlow. Journal of Cloud Computing, 9(1), 1–14.

Liu, Y., & Shao, Z. (2021). End-to-End MLOps systems: A review. IEEE Access, 9, 102347–102362.