International Journal of Leading Research Publication (IJLRP)

E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

Kubernetes: Advanced Deployment Strategies- *Technical Perspective

Bhanuprakash Madupati

Cision, NC

Abstract

Container orchestration is a foundational element of Kubernetes, the pre-eminent container orchestration platform responsible for automating the deployment, number of processes, and scaling up and down containers. Advanced Deployment Strategies official Kubernetes Concepts Doc at Kubernetes.org This paper covers some advanced deployment strategies in Kubernetes like Blue-Green Deployment Rolling updates Auto-scaling mechanisms. It is necessary to keep high availability and spare resources in cloud environments. Besides, it comes with continuous service-delivering functionality, particularly in complex Environments. This paper offers an inclusive analysis of the existing research work, which illustrates that all such strategies are combined using Kubernetes to manage resources dynamically, automate micro-services deployment, and provide continuous operations across hybrid/multi-cloud infrastructures. The discussion is exclusively based on the related work from studies regarding Kubernetes deployment techniques and resource management, offering perspectives on difficulties in Kubernetes deployment practices and future directions to be enhanced.

Keywords: Kubernetes, blue-green deployment, rolling updates, auto-scaling, microservices, cloud orchestration

1. Introduction

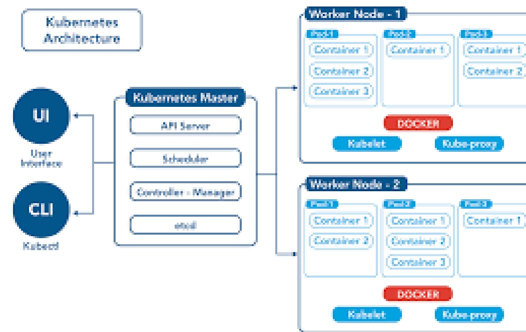
Kubernetes is widely used to orchestrate container-based applications and is a key tool in automating applications' deployment, scaling, and management across different environments. It supports intricate application architectures and is thus very common in keeping cloud environments highly available and scalable. This includes Kubernetes' ability to provide more advanced deployment strategies, a key feature enterprise customers seek to have seamless updates, save on resources, and have continuous service delivery at the highest cadence.

Blue-green deployments, rolling updates, and auto-scaling are examples of the advanced deployment strategies that Kubernetes actively supports, making it a powerful tool for managing operational efficiency. These are methods that enable organizations to stay caught up with updating an application without any disruption to ensure that your services can adjust dynamically to your workloads, as you have discussed. I am sure we did talk about maximizing the utilization of resources. In addition to tools like Prometheus and horizontal pod auto scalers, the health checking and resource provisioning in Kubernetes makes it easy to adjust for a wide range of infrastructure needs [1], [2], [5], [7].

This paper will study some of those sophisticated deployment strategies by reviewing the recent research and implementations quoted in the materials. The paper discusses the way Kubernetes can be used in delivering stable, scalable, and cost-effective cloud-native services through the examples of blue-green

deployment (this artifact transition follows gates exposing defined features), rolling updates example (implement cutover one instance update while others serve requests) and auto-scaling [1], [2], [4], [5].

Figure 1: Kubernetes Architecture Overview

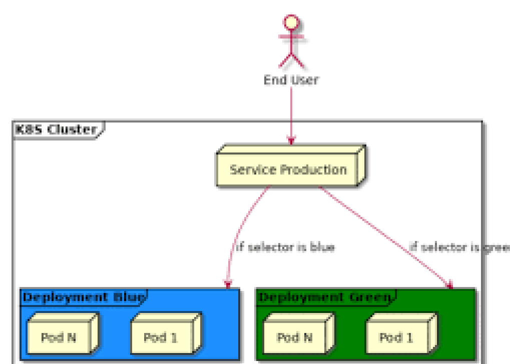


2. A Blue-Green Deployment Strategy

One common strategy in Kubernetes is Blue-green deployment, which allows deploying an application's updated version with zero downtime. This means leaving two environments in place: blue (the current version of the app) and green, where you deploy the new version. At the outset, all production traffic goes to the blue environment. The traffic is switched to the green climate after deploying and testing the latest version in the green environment. The smooth redirection is invisible to end users, enabling service availability to continue without interruption.

The company uses this deployment style primarily to minimize the update risk of critical services. If something goes wrong in the green environment, you can effortlessly flip everything back to the blue environment, limiting the impact of deployment mistakes on production services [1].

Figure 2: Blue-Green Deployment Flow



2.1 Benefits of Blue-Green Deployment

No Downtime: Users do not experience downtime while updating to a newer version, as traffic is only switched to the new environment after deployment and testing.

Rollback capability: If the new version in the green environment does not work as expected, the system can be easily rolled back to the blue environment, ensuring service continuity [1], [5].

Pre-Production Testing: The green environment can be completely tested before it is brought into production. Can catch the issues well in advance.

This can be achieved in Kubernetes using Kubernetes services with label selectors to point traffic to either blue or green based on a deployment's status.

2.2 Blue-Green Deployment on Kubernetes

With blue-green deployment, Kubernetes can automatically switch traffic between containerized applications running in different environments through service objects. They work as a routing layer in Kubernetes Services, which routes the traffic to the proper set of pods (blue or green) without disrupting services. Modifying the label selectors inside the service configuration makes traffic go into the rust version of the application running in a green environment, thereby leading to a shift between one environment and another. This guarantees continual, making rollbacks easy by sending traffic back to the blue environment.[1], [2]

Table 1: Comparison of Blue-Green Deployment and Rolling Updates

Feature	Blue-Green Deployment	Rolling Updates
Downtime	No downtime during the switch to the new version	Minimal downtime during the update process
Rollback	Immediate rollback by switching back to the blue environment	Gradual rollback by pausing the update process
Resource Usage	Requires double the resources during the update process	Uses fewer resources as pods are updated incrementally
Testing	Full testing of the green environment before it goes live	Testing occurs as pods are gradually updated
Use Case	Suitable for critical applications needing zero downtime	Suitable for services requiring continuous availability

3. Rolling Updates and Recreate Strategies

With recreate, there is zero downtime as all pods are replaced once, but you lose the current state of the connection or token store, if any.

Kubernetes provides several deployment strategies to keep serving requests without downtime and consistently handle the complexities of releasing new application versions. A standard way to do this is to use rolling updates or the recreate Strategy to update pods.

3.1 Rolling Updates

In Kubernetes, a rolling update is a deploy strategy that slowly replaces all pods of an application with the new version instead of replacing them at once. It meant new versions of the application could be incrementally deployed, so old pods (of the same app) continued running in coincidence with new ones

being started. When a deployment is performed using a rolling update, the same amount of time for the different containers to be deployed launches and stops just soon enough that there will always be a subset with an application running.

This method informs Kubernetes about the state of these pods and how to automatically handle the switch to a new version. This approach is very useful when I need to roll out changes without impacting the service or when I want to make my deployment process more resilient/ in case we need our application time to be high. Rolling an update is safer compared with full redeployment because it allows the update to be stopped or paused if something does not work in this new version [2].

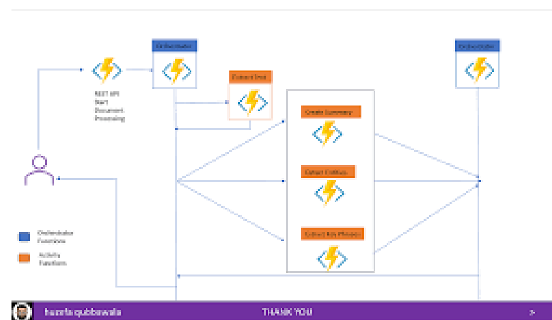
One of the biggest advantages of performing rolling updates is to avoid shear between the version from being updated and the next one running, as long as there is no breaking error in the new version. maxUnavailable and maxSurge tell Kubernetes how many old pods can be unavailable, how many new pods can be created during deployment, etc. These settings balance concerns for web performance and fine-granular control over the update process, making it suitable for a broad spectrum of application environments.

3.2 Recreate Strategy

Unlike the rolling updates, recreate takes a more brute-force approach: it stops (i.e., kills) all the currently running Pods and only then starts new ones with a new version. This method is safer as it does not allow pods from the old version to be part of deploying the latest version. Out of the box, this means downtime since we will not have any instances of our app available while we perform the deployment.

Recreate. This Strategy is implemented in scenarios where the application does not need to be available during the update or the update involves breaking changes, and both versions cannot run side by side. The recreate Strategy automatically stops old instances before creating new ones. Hence, it may not be recommended for services that require high availability [2].

Figure 3: Rolling Update Process



3.3 Comparison between Rolling Updates and Recreate Strategy

Feature	Rolling Updates	Recreate Strategy
Downtime	No downtime (if properly configured)	Downtime is inevitable during deployment
Rollback	Gradual rollback if issues are detected	No rollback without restarting the entire

		process
Resource Utilization	Pods from both old and new versions run temporarily	All resources were freed before deploying new pods
Use Case	Suitable for high-availability services	Ideal for simple deployments or breaking changes

Rolling updates are more frequently utilized in production settings when maintaining service continuity is crucial. Although it is more straightforward, the recreate approach is usually used when a brief outage is acceptable or when modifications conflict with the current version of the program [2], [5].

4. Auto-scaling mechanisms with Kubernetes

The Kubernetes system is built to dynamically handle workloads by adjusting resources based on immediate urgency. These auto-scaling mechanisms are designed to help applications maintain a certain level of performance by allocating more or less resources as needed. With less predictable workloads, auto-scaling is important in maintaining optimal resource utilization and availability of your applications.

Kubernetes mainly provides two auto-scaling mechanisms: Horizontal Pod Autoscaling (HPA) and Vertical Pod Autoscaling (VPA).

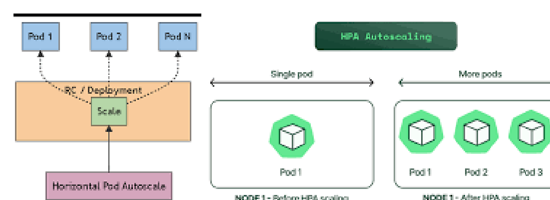
4.1 HPA (Horizontal Pod Autoscaling)

Horizontal Pod Autoscaler (HPA) is one of the most widely used auto-scaling strategies in Kubernetes. It scales the number of running pods based on the application load. It manages scaling a certain resource metric (CPU and memory) by creating (or deleting) pods depending on the predefined thresholds.

For instance, if an application's CPU utilization exceeds a predefined threshold, HPA will scale out pods automatically to balance the load. On the other hand, if demand drops and resource consumption goes below this threshold, Kubernetes scales down the number of pods to save those resources [2].

HPA is best suited for applications that have requests per minute with very high variance, like web services that experience huge spikes in traffic. Kubernetes manages this by manipulating the number of pods (and capacity) to keep the application responsive while not wasting resources when demand is low [2].

Figure 4: Horizontal Pod Autoscaling (HPA)



4.2 Vertical Pod Autoscaling (VPA)

On the other hand, Vertical Pod Autoscaling (VPA) scales pod resource limits(CPU, memory) size and not the number of instances running. The VPA watches the actual resource utilization of Pods on the

cluster and adjusts the number of resources allocated to them up or down if required. This is useful for applications where the number of pods is steady, but the resource needs might change over time.

This field is deeper escalation than splitting an application over many pods (many:->replicas) yet not superficial enough to justify the need for additional pods of a specific application and taking more or less the resources in percentage-wise concerning the recommendation least IBM suggests this as part of VPA illustration. By changing the resource limits of the pods on a per-pod or user/namespace basis, VPA ensures that each pod gets the right amount of resources for its workload to run smoothly and not be over- or under-provisioned [2].

Table: Horizontal vs. Vertical Autoscaling

Scaling Mechanism	Horizontal Pod Autoscaling (HPA)	Vertical Pod Autoscaling (VPA)
Scaling Type	Adjusts the number of pods	Adjusts CPU and memory limits per pod
Response to Workload	Scales out/in based on aggregate resource usage	Resizes pod resource requests based on individual pod needs
Use Case	Applications with fluctuating traffic (e.g., web services)	Applications with static pod numbers but variable resource needs
Resource Efficiency	Avoids under- and over-provisioning by adjusting pod count	Optimizes resource utilization within existing pods

4.3 Benefits of auto-scaling in Kubernetes

Resource usage: Scaling up and down according to the requirement allows Kubernetes to be the best-suited scheduling solution, providing most of the compute resources.

Cost Efficiency: Auto-scaling eliminates manual intervention, making it easier to manage resources more efficiently and reducing operational costs.

Better Performance: Applications never go down, remaining highly available and responsive as Kubernetes dynamically responds to changing resource demands [2].

According to a study, Kubernetes can improve CPU utilization by 28.99%. The study showed dynamic scaling capabilities, especially when using auto-scaling mechanisms [2]. This enhancement improves resource efficiency and the performance of Kubernetes-managed clusters in general.

5. Kubernetes Resource Provisioning and Monitoring

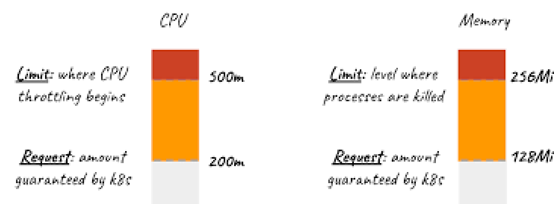
Deploying applications in Kubernetes is important because Kubernetes provisioning and monitoring resources becomes crucial, especially when working on dynamic cloud infrastructures. The Kubernetes resource monitoring implements allocation optimization and real-time scaling features very robustly. With these features, Kubernetes can provide the bare minimum that an application needs to work effectively, creating minimal waste.

5.1 Resource Provisioning

Resource provisioning in Kubernetes is all about providing the right amount of computational resources (CPU, memory) to needed containers or pods. Kubernetes manages resource allocation by using both requests and limits. A resource request is the minimum amount of CPU or memory guaranteed for a pod, while a resource limit is the maximum amount it can consume.

Resource provisioning is important for avoiding contention of resources where pods compete for resources and can potentially degrade the performance of applications. Kubernetes manages and dynamically allocates resources depending on the current load to ensure high availability and performance for cloud environments. Provisioning resources on demand also works well with modern cloud billing models that are typically charged per millisecond second. Hence, it is essential to reduce unused resource time[7].

Figure 5: Kubernetes Resource Requests and Limits



5.2 Monitoring in Kubernetes

In the end, monitoring is very important for Kubernetes to properly manage and allocate resources. Kubernetes is defined by some of the tools and mechanisms used to ensure that applications are in a state of continuous health, performance, and resource consumption. Heapster and InfluxDB are good monitoring tools for Kubernetes.

Performance metrics are collected using Hipster, which collects performance data on the CPU and memory usage of each pod in the Kubernetes cluster. These stats are very important for making decisions about auto-scaling and resource allocation.

Heapster collects metrics and stores them as Logs in a time-series database called InfluxDB; you can monitor the resource usage pattern in real-time. With these tools, Kubernetes can scale the number of pods or partition allocations based on real-time usage data so that there is no under- or over-provisioning of resources[7]

These monitoring features also reach further than resource allocation, as Kubernetes can monitor the health and performance of various pods, nodes, etc. With this information, potential issues can be detected early, and corrective action may be taken automatically.

5.3 Importance of Monitoring in Dynamic Scaling

The system can adjust according to the monitored data by integrating monitoring tools with Kubernetes auto-scaling mechanisms. Kubernetes can scale pods horizontally by increasing the number of instances or vertically by adjusting resource limits, and it does so automatically based on constantly collecting metrics to maintain application performance while avoiding waste of resources. This policy enhances the

overall efficiency of cloud resource management by minimizing the operational cost and ensuring QoS (quality of service) [7]

6. Kubernetes in Cross-Cloud and Hybrid Cloud Deployments: Challenges and Future Directions

Enterprises increasingly deploy their applications across several cloud providers in public and private clouds. This is usually called the hybrid cloud or multi-cloud deployment that minimizes resource wastage from one side, locks in from the other, and improves application reliability. *Kubernetes* is the tool that helps make deployments like this possible. It acts as an abstraction layer over bare metal servers. It provides a unified container orchestration solution on top of them with facilities for explicitly defining how applications should be deployed.

6.1 Cross-Cloud and Hybrid Cloud Deployments with Kubernetes

One of the benefits Kubernetes offers is that it allows organizations to deploy applications on any cloud platform like Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, or any private infrastructure. Because it is vendor-agnostic, workloads can move natively among clouds or span multiple clouds simultaneously, making it extremely flexible and adaptable. This is accomplished by Kubernetes being able to orchestrate containers anywhere; it provides a common and neutral interface for managing containerized applications.

Multi-cloud architecture allows organizations to distribute workloads using multiple service providers. Kubernetes makes this easy by allowing the deployment process to be consistent regardless of the underlying infrastructure. This means businesses can leverage the best qualities of various cloud providers—cost, performance, or geographic availability—while reducing the risks of vendor lock-in [4]. It is particularly important to avoid vendor lock-in when the organization becomes so dependent on a particular hosting provider that it cannot easily or economically switch to other providers.

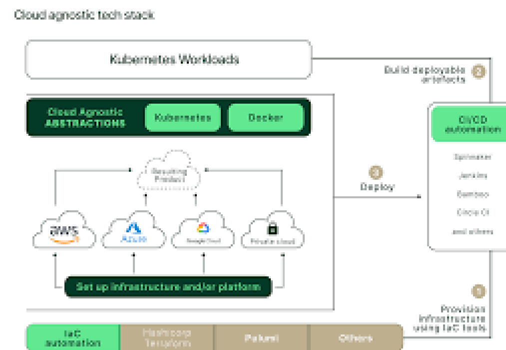
When private clouds (on-premises data centers) are used with public cloud infrastructure, hybrid cloud deployments are used. Kubernetes provides easy-to-provide common functionality and abstracts the differences between private and public Kubernetes, thus ensuring that applications can be deployed in any environment without changing anything. Enterprises can use Kubernetes as a common management layer to keep private-cloud control of the data but scale-out workloads on a public cloud during peak times [4].

A way to manage your hybrid infrastructure with Kubernetes For this article, we will use Terraform. This tool allows you to define and provision more or less any infrastructure in different cloud environments (Kubernetes is one of them!) Going forward, Terraform integration for provisioning both cloud resources and creating Kubernetes clusters will facilitate this automation in a hybrid cloud deployment of applications. Then, Kubernetes orchestrates the deployment of these applications and standardizes configurations everywhere they are deployed [4].

One of the significant upcoming developments for Kubernetes in multi-cloud and hybrid cloud environments is the nascent trend of osmotic computing. Osmotic computing enables applications to move between networks and locations, offering dynamic resource provisioning so that the required resource can be rented/leased on demand for mission-critical jobs. Kubernetes is well-suited for it

because of its container orchestrating features, such as auto-scaling and horizontal scaling, which makes it perfect to be deployed as cross-cloud deployments [1].

Figure 6: Kubernetes in Multi-Cloud Deployment



6.2 Challenges in Cross-Cloud and Hybrid Cloud Deployments

Just like virtualization, Kubernetes may seem promising and perfect in theory, except working with the tool itself in the real world of multi-clouds and hybrid clouds comes with some hurdles. Organizations choose to run workloads on multiple cloud infrastructures for various technical, operational, and security reasons.

Cost Optimization & Resource Management

Resource management across cloud providers is a hassle with cross-cloud deployments. These cloud providers have different pricing models and resources, complicating managing costs simultaneously with performance consistency. In the case of AWS, GCP and A-Zure have different pricing for compute storage or network services. While Kubernetes abstracts some of this complexity, organizations must balance their spending across clouds to ensure they pay attention and applications get enough concurrency and resource utilization [4]. Different cloud providers will have different performance characteristics. Unless these are specifically managed, this can lead to some resources that require more IOPS (Input/output operations per second) being heavily used while others are not.

Security and Compliance:

The challenges of managing security across multiple clouds include security, identity, and compliance. Each cloud provider has its own set of security protocols and an identity management system. While Kubernetes features role-based access control (RBAC) and integrates with IAM systems for identity management, it can be hard to have a consistent security strategy that unifies across variances in cloud-based environments [4]. Ensuring compliance with the GDPR, HIPAA, and other regulatory requirements, especially when data is being spread across multiple jurisdictions, also poses a challenge to organizations in multi-cloud deployments.

Networking and Latency:

Cloud providers do not agree on how data transfer works, and networking (network latency, bandwidth. Per GigaByte) can be difficult for different cloud providers. Data center networking architecture must be carefully designed and tuned to provide low-latency communication for applications that run in different

clouds. The service mesh capabilities of Kubernetes (like Istio) tackle this complexity by offering observability, security, and traffic management among services. However, from a network perspective, cross-cloud networking will always be the hardest aspect of multi-cloud deployments, especially when running high-performance and low-latency applications where every microsecond counts [4].

Operational Complexity:

Spreading the required functionality across multiple clusters allows for fine-grained access to resources. This has a natural advantage as many managed services are always in the cloud land, not the planet Kubernetes that you reside on. Although Kubernetes tries to simplify this by providing a single platform, cloud providers have different ways of managing their resources, so you would still need to learn how they offer tools and services to do so. Multiple clouds manage updates, patching, and troubleshooting, making it worse for the ops team, especially in large-scale deployments.

6.3 Future Directions for Kubernetes in Multi-Cloud and Hybrid Cloud Environments

Therefore, the future work of Kubernetes development will center on multi-cloud orchestration solutions, the security challenges for K8s, and automation—hopefully! This will lead to simplified application deployment and life-cycle management across hybrid, multi-cloud environments, and cloud service providers, which means organizations can maximize the benefits of cloud computing.

Enhanced Multi-Cloud Orchestration:

The future of Kubernetes is multi-cloud orchestration—the ability to deploy, scale, and manage applications across different cloud environments as naturally as on a single cloud. Future developments in Kubernetes will increasingly enable convenient consumption by more than one cloud provider, improve locality, and optimize workloads based on real-time environmental variables such as cost, resource availability, and performance [4]. This approach should provide the necessary portability that businesses in mainstream industries require to migrate applications freely between clouds.

Enhanced Security Models:

Security is one of the highest priority improvements to Kubernetes, especially in multi-cloud deployments. These new updates will also lead to advancements like enhanced multi-cloud identity management capabilities, more secure encryption standards, and increased integrations with cloud-native security tools. Interoperability with security solutions across a range of cloud providers should also enable companies to maintain a common security profile, reducing the risk that operating over multiple environments would normally represent [4].

Automation and Tooling:

A major addition to the deployment capabilities of mainstream orchestration systems, Kubernetes automates more and more tasks. Helm (Kubernetes Package Manager) and Terraform are great tools for process management. These solutions can simplify deploying applications on different environments, reducing operational overhead when managing a hybrid cloud infrastructure [4]. With the evolution of Kubernetes, these automation tools will mature, which will help you with things like self-healing applications and automatically scale across cloud environments.

7. Conclusion

Below are the key takeaways from my detailed backup on Kubernetes' advanced deployment strategies and practices in hybrid-cloud and multi-cloud architectures.

1. Advanced Deployment Techniques:

With blue-green deployments and rolling updates, Kubernetes supports app deployment by ensuring there is no downtime during a new deployment while allowing the app to be released safely.

These methods allow flexible deployment to keep the services highly available and efficient while upgrading or changing services [1], [2].

2. Auto-scaling for Efficiency:

Horizontal and Vertical Pod Autoscaling in Kubernetes: Along with the probability of availability, Kubernetes uses mechanisms such as Horizontal Pod Autoscaling (HPA) and Vertical Pod Autoscaling (VPA) to ensure resources are allocated on demand only where they are needed.

Dynamic scaling has also improved CPU utilization and reduced overall resource usage [2].

3. System Monitoring and Resource Provisioning:

How Kubernetes uses integrated tools like Heapster and InfluxDB to monitor resources across a cloud and dynamically allocate additional or free unused resources to maintain quality of service.

These tools provide near real-time insights into application performance, keeping applications responsive at various workloads [7].

4. Cross-Cloud and Hybrid Deployments

Kubernetes has proven very successful in managing multi-cloud and hybrid cloud setups by abstracting away the arduous infrastructure operations task with vendor-neutral deployment options.

A key benefit of Kubernetes is that it enables deployment on different cloud providers, helping prevent vendor lock-in and enabling automated deployments across private and public clouds [4].

5. Challenges in Multi-Cloud Environments:

Multi-cloud deployments also pose significant resource management, networking, and security challenges.

Running Kubernetes in the field is operationally complex, considering multiple environments must be coordinated for efficiency, low-latency communication, and security compliance [4].

6. Future Directions:

With the maturation of multi-cloud orchestration, automation, and security in the future, since Kubernetes will be a tool to easily manage complex cloud environments on public clouds where users choose to provision resources using any one or more of their accounts, its invaluable role in cross-cloud productions is increasingly being appreciated.

This helped us to show why tools like Terraform and Helm are crucial in automating multi-cloud workflows, reducing operational overhead, and enabling self-healing and auto-scaling capabilities [1], [4].

Reference

[1]. Buzachis, A. Galletta, A. Celesti, L. Carnevale, and M. Villari, "Towards Osmotic Computing: a Blue-Green Strategy for the Fast Re-Deployment of Microservices," in Proc. of the 2019 IEEE Symposium on Computers and Communications (ISCC), Jun. 2019, pp. 1-6. doi: https://doi.org/10.1109/iscc47284.2019.8969621.

[2]. H. Rajavaram, V. Rajula, and B. Thangaraju, "Automation of Microservices Application Deployment Made Easy By Rundeck and Kubernetes," in Proc. of the 2019 IEEE International Conference on Electronics, Computing and Communication Technologies (CONNECT), Jul. 2019, pp. 1-5. doi: https://doi.org/10.1109/conecct47791.2019.9012811.

[3]. Q. Wu, J. Yu, L. Lü, S. Qian, and G. Xue, "Dynamically Adjusting Scale of a Kubernetes Cluster under QoS Guarantee," in Proc. of the 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), Dec. 2019, pp. 1-8. doi: https://doi.org/10.1109/icpads47876.2019.00037.

[4]. C.-C. Chang, S.-R. Yang, E.-H. Yeh, P. Lin, and J.-Y. Jeng, "A Kubernetes-Based Monitoring Platform for Dynamic Cloud Resource Provisioning," in Proc. of the 2017 IEEE Global Communications Conference (GLOBECOM), Dec. 2017, pp. 1-6. doi: https://doi.org/10.1109/glocom.2017.8254046.

[5]. M. Orzechowski, B. Balis, K. Pawlik, M. Pawlik, and M. Malawski, "Transparent Deployment of Scientific Workflows across Clouds - Kubernetes Approach," in Proc. of the 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Dec. 2018, pp. 1-8. doi: https://doi.org/10.1109/ucc-companion.2018.00020.

[6]. M. Beltre, P. Saha, M. Govindaraju, A. Younge, and R. E. Grant, "Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms," in Proc. of the 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), Nov. 2019, pp. 1-7. doi: https://doi.org/10.1109/canopie-hpc49598.2019.00007.

[7]. D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," IEEE Cloud Computing, vol. 1, no. 3, pp. 81–84, Sep. 2014, doi: https://doi.org/10.1109/mcc.2014.51.