Management of Self-Healing Systems for Multi-Cloud Deployments on Kubernetes

MSc Research Project
Master of Science in Cloud Computing

Vaishnavi Udayrao Deshpande
Student ID: X23183209

School of Computing
National College of Ireland

Supervisor: Sai Emani

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vaishnavi Udayrao Deshpande
Student ID:	X23183209
Programme:	Master of Science in Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Sai Emani
Submission Due Date:	12/12/2024
Project Title:	Management of Self-Healing Systems for Multi-Cloud Deployments on Kubernetes
Word Count:	5872
Page Count:	18

Signature:	Vaishnavi Deshpande
Date:	12th December 2024

Management of Self-Healing Systems for Multi-Cloud Deployments on Kubernetes

Vaishnavi Udayrao Deshpande
X23183209

Abstract

In the growing world of Cloud Computing, multi-cloud architectures have gained significant popularity as the organizations are looking to enhance the flexibility, resilience, as well as performance by leveraging the services from multiple cloud providers. However, management of system failures in such complex environments remains a crucial challenge. This research study aims to explores the Integration and management of self-healing mechanisms within multi-cloud deployments on Kubernetes and automating the failure detection and recovery processes to minimizer downtime and ensure continuous availability.
Utilizing Kubernetes Operators, Datadog for real time monitoring, and Jenkins for the CI/CD automation, the study is focused on developing a deployable framework to enhance performance, security, and resilience in multi-cloud environments. Some of the Key failure points across cloud providers including AWS, GCP and Azure are identified, and various tools are tested to evaluate their scalability and efficiency. The results show that the self-healing system significantly reduces recovery time, optimizes resource usage, and maintains high availability, even in the event of failures. This designed framework provides valuable insights for cloud DevOps Engineers, which offers practical solutions for automated failure recovery and improving the overall management of multi-cloud services. This research work also discusses potential improvements including refining failure detection and recovery workflows, and suggests future directions for advancing autonomous, cloud-native systems.

1 Introduction

Cloud technology is growing rapidly. In modern cloud computing, multi-cloud architectures are increasingly becoming popular as organizations leverage the services of multiple cloud providers to optimize their infrastructure. This approach enhances flexibility, performance as well as resilience by avoiding vendor lock-in and distributing the workloads more efficiently (Sharma, 2022). However, the complex nature of multi-cloud deployments presents several challenges, particularly around managing system failures while ensuring seamless operations and maintaining high availability at the same time. Self-healing mechanisms are crucial to addressing these challenges, which helps enabling automated detection and recovery from failures to minimize downtime (Ye, et al., 2016).

This research study aims to explore how self-healing mechanisms, enabled through Kubernetes operators along with supporting tools like Datadog, can be effectively integrated to

enhance the performance, security and resilience of multi-cloud environments. The study also aims to identify key failure points in such environments, providing practical solutions through automation and real-time monitoring which can be considered highly valuable for Cloud DevOps Engineers. Furthermore, it also consists of testing and comparing various tools, offering insights into their scalability and performance in multiple cloud architectures. The ultimate goal of the study is to develop a deployable framework for DevOps engineers, which helps enhance automated failure recovery processes and simplifying the management and deployments of multi-cloud services. In summary this study is important in cloud computing as it addresses the essential needs for resilience, automation, efficiency, security as well as sustainability in multi-cloud environments. These contributions not only solve current challenges but also set the stage for future advancements in autonomous, cloud-native systems.
This research paper is comprised of multiple sections. This section includes the introduction to the tools and technologies used in this study. Further moving on, the flow of the paper is as:
Literature review Section which delves into some of the past research work done in this field. The **Methodology and Implementation** sections are dedicated for the detailed implementation for development of the framework for management of self-healing systems for multi-cloud deployments on Kubernetes. To outline the outcomes of the study, the results and outputs derived from various metrics considered are shown and discussed in the **Results and Discussion** section. The paper also provides the conclusion of the study and further scope for research in the same area in **Conclusion and Future scope** part. Finally, **References** section provides the bibliography, and the references used to carry out this research work.

2 Literature Review

This section is the extract of the related work done in the past in this field. It is comprised of three sections which are 1) Multi-Cloud Deployment and management 2.1, 2) Self-Healing and Self-Adaptive Systems 2.2, and 3) Private Multi-Cloud and Advanced Techniques 2.3.

2.1 Multi-Cloud Deployment and management

Managing multi-cloud Deployments on Kubernetes with Istio, Prometheus and Grafana (Sharma 2022, pp. 1-10), is the key research work for this study, where the author has addressed the popularity of multi-cloud architectures and the effective management of deployments in this multi-cloud environment using Kubernetes, integrating it with Istio, Prometheus and Grafana as well. The Author identified several challenges in managing multi-cloud deployments such as Vendor lock-in, network latency, security policies across different platforms and proposed a multi-cloud architecture using Kubernetes, Istio, Grafana and Prometheus to mitigate them. Kubernetes is a widely used container orchestration platform which offers a robust solution for scaling and maintaining applications across the cloud providers. This study by Sharma (2022) emphasizes the abilities of Kubernetes to handle multi-cloud setups by providing consistent management layer. Istio is the tool that helps in managing multi-cloud traffic by providing a service mesh which decouples service, traffic management and security from application itself. Monitoring in multi-cloud environments is crucial for maintaining system health, detecting failures, and ensuring optimal performance and author has explained the role of Prometheus as a powerful management system which is suitable for Kubernetes environments Sharma (2022). The integration of Prometheus with Kubernetes provides visibility and helps in detecting anomalies. The other tool used by Sharma in study is Grafana for the metrics visualization which also offers customizable dashboards. In multi-cloud environments, Grafana's ability to collect and display data from multiple sources is important for ensuring the reliability of applications running across different clouds. Overall, the work by Sharma provides valuable insights into how integration of multiple tools including Kubernetes, Istio, Prometheus and Grafana can be leveraged to effective management of multi-cloud deployments. This study by Sharma is valuable reference study which motivated the use of Kubernetes and other tools for current study.

In the research 'Preventing Vendor lock-ins via an interoperable multi-cloud deployment approach',Pellegrini et al. (2017) have addressed an important issue in cloud computing which is Vendor lock-in. Vendor lock-in is where organizations become dependent on single cloud provider's services which make it difficult and expensive to switch between providers and to integrate with others. As a solution to this, authors proposed an interoperable multi-cloud deployment approach which would allow the applications to be deployed seamlessly across multiple clouds. The proposed interoperable approach by authors makes use of open-source cloud agnostic tools such as Kubernetes, terraform and other multi-cloud management platform tools including Juju Charms and Conjure-up. These tools allow organizations to manage resources across cloud platforms using a consistent interface which helps reduce the complexities in multi-cloud management. Moreover, in the paper, Pellegrini et al. (2017) also explain the adoption of cloud native practices which make applications more flexible. The authors also have highlighted that multi-cloud strategy not only helps preventing vendor lock-in but also has multiple other advantages in terms of cost optimization, scalability as well as disaster recovery. The paper by Pellegrini et al. (2017) contributes crucially to the discussion regarding vendor lock-ins and multi-cloud strategies cloud computing which can be further explored in detail.
The study 'Integrating Jenkins for Efficient Deployment and Orchestration across Multi-Cloud Environments' by authors Pandi et al. (2023) is focused on the Integration of Jenkins which is a popular Continuous Integration Continuous Deployment (CI/CD) tool, to enable efficient deployment and orchestration in multi-cloud environments. In the paper, authors have proposed Jenkin's role to serve as a tool for multi-cloud orchestration where it is integrated with several cloud-native services to manage different cloud providers which includes AWS, Azure as well as GCP Pandi et al. (2023). Along with this, the authors have discussed the role of Jenkins in managing containerized applications using Docker and deploying those containers across cloud-based Kubernetes clusters which enabled cloud-based deployments. The work by authors Pandi et al. highlights Jenkin's ability to define and manage deployment workflow as a code which improves the reproducibility of deployments. The authours also emphasizes Jenkin's support for Infrastructure as a Code practices (IaC) where Terraform in integrated, which enables the version controlling of resources which enhances scalability and reliability. The study by Pandi et al. (2023) discusses Jenkin's integration with orchestration tools such as Kubernetes and Service Mesh tools which for the management of traffic between services running on different clouds and integration of these tools allow applications to be managed and scaled across cloud platforms. In conclusion, Pandi et al. (2023) have highlighted

the importance of automation in multi-cloud deployments with Jenkins in creating scalable and reliable CI/CD workflows via integration of multiple other tools as well which motivated to leverage Jenkins for the current study for the creation of CI/CD pipeline.

2.2 Self-Healing and Self-Adaptive Systems

Samarakoon et al. (2023) in work 'Self-Healing and Self-adaptive Management for IoT-Edge Computing Infrastructure' have proposed a self-healing and self-adaptive framework for IoT-Edge computing infrastructure. The rapid expansion of IoT devices has driven adoption of Edge computing, where the computing resources are placed closer to the data resources to reduce the latency and improve performance Samarakoon et al. (2023). The paper proposes a model-driven framework that combines self-healing and self-adaptive techniques using containerized services and orchestration tools like Kubernetes. This framework is capable of managing resources dynamically, balancing the load and recover from failures and is based on the idea that aims to detect system failures automatically, resolve them and recover from the faults and adapt resource allocation. Self-healing mechanism includes automated recovery which is based on the diagnosis where AI and Machine Learning play a significant role where these mechanisms are crucial for maintaining the health of large scale IoT-Edge systems where manual interventions would be too slow Samarakoon et al. (2023). Alongside self-healing Samarakoon, S. et al. have also emphasized the role of self-adaptive management where the system dynamically adjusts its behavior based on system changes including network congestion, resource availability and application demand. Self-adaptive systems are specifically important in multi-cloud environments where the workloads can fluctuate unpredictably. This research by Samarakoon et al. (2023) focuses on IoT-Edge environments, many of the techniques can be applied to multi-cloud environments where systems also require self-healing and self-adaptive infrastructure to maintain performance and resilience. The use of tools such as Kubernetes, Datadog in multi-cloud environments shares resemblance with the approaches explained by authors which can be further leveraged for public multi-cloud deployments of applications.
Cloud services operate on highly complex and distributed infrastructures, which makes them liable to failures that can disrupt services. Ye et al. (2016) in their work, highlighted that dependability, which includes reliability, availability and maintainability is a major issue in cloud computing. The paper has discussed challenges such as hardware failures, network outages, and software bugs which fall under common causes of downtime in cloud environments. In the work, authors proposed a self-healing mechanism as a solution to address the issue and to enhance dependability by detecting, diagnosing and recovering from the failures in real-time. Self-healing systems are designed to handle predictable as well as unpredictable faults without any human invasion which helps services to be uninterrupted. The approach for self-healing discussed by Ye et al. (2016) . is centered around Automation. Through automated monitoring systems, cloud services can continuously collect and analyze the data from system performance, resource utilization and error rates as well to detect the anomalies. Authors Ye et al. (2016) have proposed the use of machine learning algorithms to enhance failure prediction, which will help the systems to proactively address the issues before they impact the end-users. Automated recovery actions including restarting process or redistribution of workloads help maintaining service dependability and at the same time, minimize the downtime.

Overall study by Ye et al. (2016) demonstrated the value of self-healing mechanisms in improving the dependability of cloud services and that by automating the fault detection and recovery, cloud providers can achieve increased reliability, scalability and efficiency as well.

In the paper 'A framework for self-healing and self-adaptation of Cloud-hosted web-based applications' by Magalhães and Silva Magalhães and Silva (2013) have proposed the framework which will enhance the self-healing and self-adaption capabilities of cloud-hosted web-based applications. Cloud-hosted web-based applications often experience fluctuations in workload and require a high degree of reliability and availability as well Magalhães and Silva (2013). The proposed self-healing and self-adaption framework by is designed to detect and respond to the failures and changing conditions in cloud-hosted applications automatically which is built to monitor application performance, diagnose the issues and trigger automated recovery and adaption actions whenever necessary. The framework uses real-time monitoring tools to collect performance metrics which are then analyzed to detect deviations from normal behavior and through this, the system can recognize faults early and trigger pre-configured recovery processes. Authors Magalhães and Silva (2013) presented and claimed that the proposed framework's self-healing abilities notably improve application reliability and resilience by reducing the impact of faults on end-users which further helps enhancing overall service quality and meeting Service-Level Agreements (SLAs) more efficiently.

2.3 Private Multi-Cloud and Advanced Techniques

Likewise public multi-clouds, Private multi-cloud infrastructures enable organizations to combine resources from various private clouds, which helps enhance flexibility and resilience. However, maintaining service continuity and reliability across multiple private clouds can be challenging due to issues like network inconsistencies, resource fragmentation, and system failures which are addressed in the work by authors Mfula and Nurminen's (2018) Mfula and Norminen (2018) titled 'self-healing cloud services in private multi-clouds'. In their work, authors have discussed on the application of self-healing mechanisms within private multi-cloud environments, which aims to enhance the reliability, availability of cloud services. The authors have highlighted the use of self-healing mechanisms to manage failures automatically in which, self-healing involves monitoring of the services in real time, identifying faults, and taking actions to correct them and restore the system stability (Mfula & Norminen, 2018). The key techniques include failure detection, automatic resource allocation and recovery workflows, which allow services to maintain workflow without human intervention. The authors Mfula and Norminen (2018) propose a self-healing framework designed for private multi-cloud setups, highlighting automation in fault-recovery and workload management. To enhance failure prediction and resource allocation the proposed framework integrates machine learning algorithms which allow the system to adapt dynamic workloads and changing conditions within the private multi-cloud setup. The authors Mfula and Nurminen's have offered important insights into development of self-healing mechanisms as well as the importance of monitoring and automation tools in private multi-cloud environments and the strategies discussed by them can be leveraged in public multi-cloud architectures as well.

3 Methodology

This work aims to provide a comprehensive framework for the management of multi-cloud deployments which includes the integration of multiple clouds, an orchestration tool Kubernetes and monitoring tools. This methodology section follows a systematic approach which comprises of parts: (1) Requirements identification 3.1and (2) Architecture design and Workflow. 3.2

3.1 Requirements Identification:

Identifying the requirements for this research involves good understanding of the critical components needed to design, implement, and evaluate a self-healing multi-cloud framework.

- **Multi-cloud domains:** It is important for this system to support multi-cloud deployments across leading cloud providers. The popular public cloud AWS, GCP and Azure are ideal for a self-healing, multi-cloud Kubernetes due to their multiple advantages such as high availability and fault tolerance across the world, strong multi-cloud and compatibility with Kubernetes and service mesh solutions, native monitoring tools which ensure secure and reliable performance. Along with these, these clouds provide robust CI/CD tools like AWS CodePipeline, Google Cloud Build and Azure DevOps and also support external tools like Jenkins, Bitbucket, etc.

- **Orchestration and Self-healing:** Kubernetes is the best choice for orchestration and self-healing in this study. First of all, it is an open-source tool which integrates seamlessly with all of the clouds. It provides automation, scalability, and fault-tolerant capabilities. It also simplifies multi-cloud deployments by automating workload distribution, scaling, and resource optimization across AWS, GCP and Azure.
  Kubernetes' native self-healing mechanisms such as health monitoring, automated pod restarts, and fault isolation, ensure system stability and resilience. Its integration with monitoring tools like Datadog enhances inter-cloud communication and real-time performance tracking. In addition to this, its declarative configuration model and strong ecosystem support make it ideal for managing complex, reliable, and scalable multi-cloud architectures.

- **CI/CD for Deployment:** Implementing CI/CD pipelines and automating the deployment processes are crucial for the DevOps Engineers. In this framework design, Jenkins and Git are used. Jenkins Project (2024) and Git SCM (2024) are ideal CI/CD tools for this study as these provide flexibility, automation capabilities, and seamless integration with Kubernetes and all cloud environments.
  Jenkins enables highly customizable pipelines, automated deployments and dynamic scalability through Kubernetes integration. Whereas Git provides reliable version control, collaborative workflows and branching strategies to ensure code consistency and minimize errors. Jenkins and Git together support cross-platform compatibility across AWS. GCP and Azure, while integrating effectively with the and monitoring tools as well. These tools are cost-effective and adaptable which ensures efficient and reliable CI/CD workflows for this project.

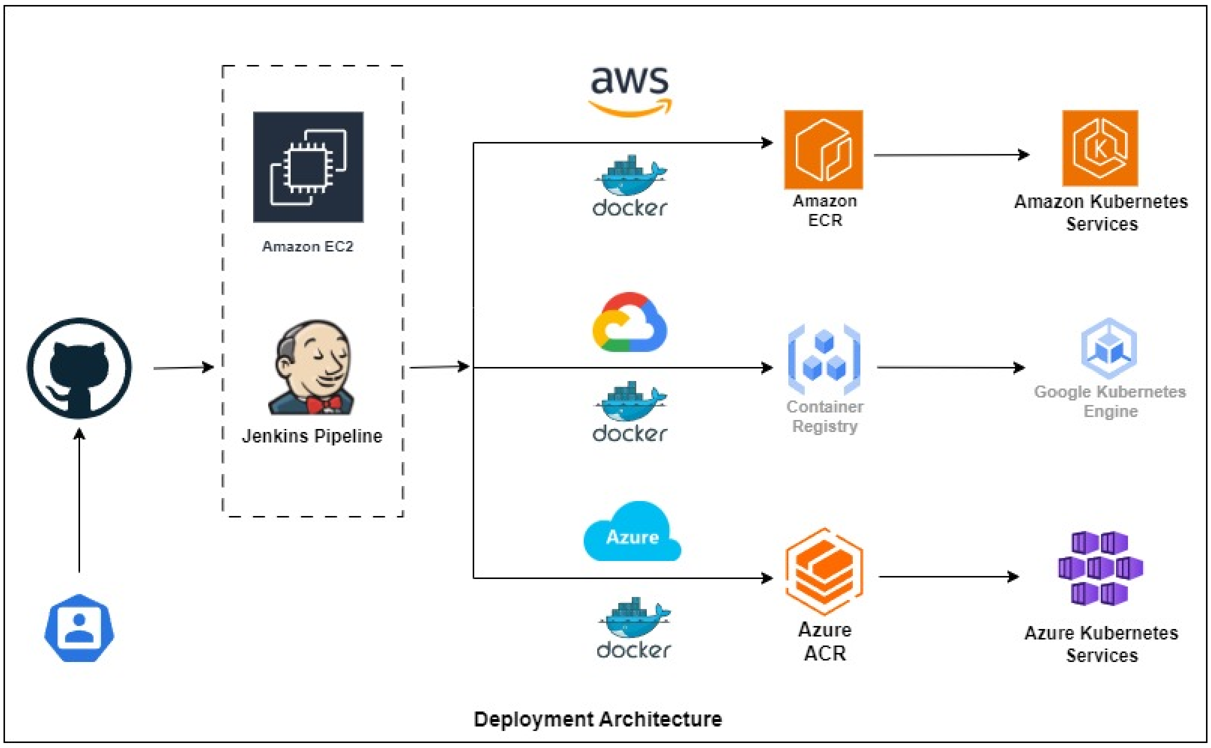


Figure 1: CI/CD Pipeline

- **Monitoring:** Monitoring is critical for this system to work as expected, as monitoring will provide the continuous feedback if there is any failure in the system which can be recovered in time. Datadog Monitoring (2024) is one of the most popular monitoring tools which provides deep insights into the system health, application performance, and resource utilization. It integrates seamlessly with multi-cloud environments and Kubernetes as well. Datadog's ability to automate failure detection, trigger alerts and offer anomaly detection ensures that self-healing systems can respond quickly to issues, which helps minimizing down-times. Its scalability, security monitoring, and collaboration tools further enhance its suitability for complex, large-scale cloud architectures, which makes it a powerful tool for maintaining resilient, efficient, and secure multi-cloud systems.

3.2 Architecture Design and Workflow:

Figure Figure 2 Shows the high-level overview of proposed system architecture diagram. As shown in figure Figure 1, draw.io (2024) Application is deployed using CI/CD tools like GitHub and Jenkins. GitHub provides easier way to version control the source code and Jenkins is used to create a pipeline which consists of the steps like containerizing the application using the dockerfile provided along with an application in GitHub and creating an image for the same. Once the image is created, that is pushed to the container registries of all the clouds where the application image can be saved and pulled from by Kubernetes clusters. Using that image, the application is then deployed on the Kubernetes clusters in each cloud platform.

The architecture leverages the Kubernetes-native self-healing mechanisms across multiple cloud providers. Figure Figure 3, shows the Components of self-healing system

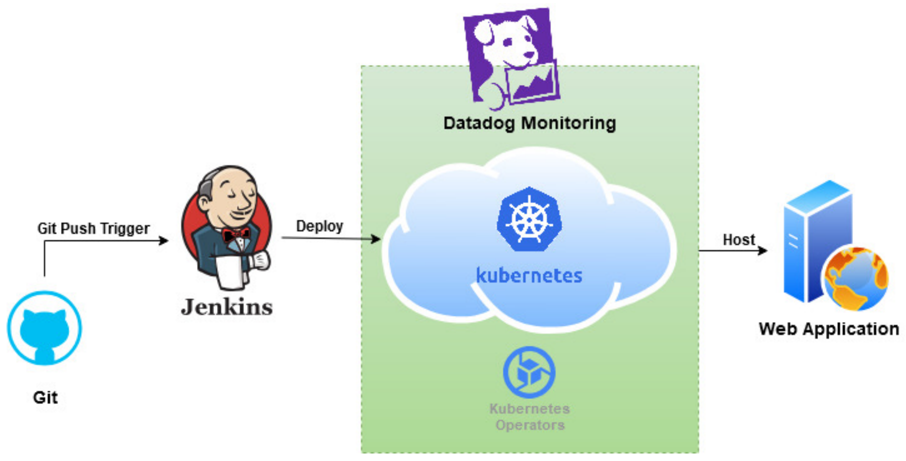


Figure 2: Architecture Overview

architecture which are as follows:

- **Application pods:** Application pods are managed by Kubernetes Deployment that ensures a specific number of replicas are always running which includes Liveness probe to check if pods are alive. Whereas Readiness Probes ensure only healthy pods serve traffic.

- **Service:** Service mechanism abstracts the pods and provide a stable endpoint for external as well as internal traffic. Service routes the traffic only to pods in 'Ready' state.

- **Horizontal Pod Autoscaler (HPA):** HPA mechanism in Kubernetes Native dynamically scales the number of pods based on resource usage such as CPU utilization.

- **Pod Disruption Budget (PDB):** PDB endures that at least a minimum number of pods are always available during maintenance or any kind of disruption for service availability.

For this proposed system to work as expected, monitoring plays a very crucial role. Datadog is an open-source platform which makes it possible to monitor as well as visualize the metrics and gives out alerts in real time if any failure occurs. It is very useful in monitoring and managing Kubernetes clusters, applications and cloud infrastructure. Datadog's integration with Kubernetes provides insights into health, performance, and resource usage of the clusters on multi-cloud. In this design, as shown in Figure Figure 3. datadog is set up to monitor node health in terms of CPU, Memory and Storage usage of nodes, tracking pod status during deployments, and provides cluster metrics like total nodes, pods, and deployments as well.

4 Implementation

The Implementation of management of self-healing framework for multi-cloud deployments on Kubernetes involves multiple stages, focusing on integration of various tools and technologies to achieve better management of automated failure recovery, resilience

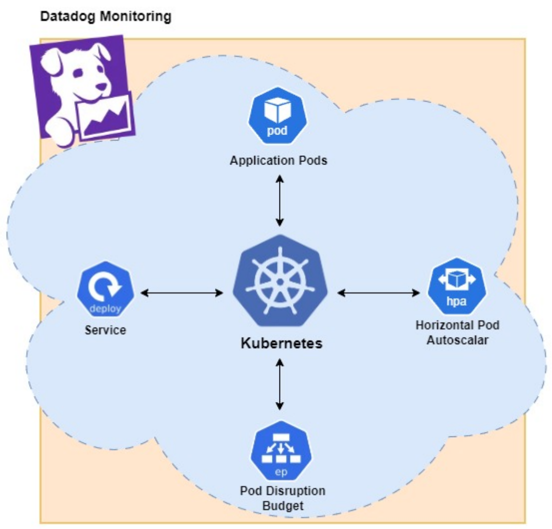


Figure 3: Self-Healing Mechanisms

and real-time monitoring. The following steps outline the process considered to develop this proposed framework:

1. **Multi-Cloud Deployment Set-Up:** The proposed self-healing system is developed and deployed on a multi-cloud Infrastructure including AWS, GCP and Azure. The deployment is configured using Kubernetes clusters in each providers, which ensures high availability and fault tolerance which included following steps:

   - Cluster Configuration: Each of the cloud provider's Kubernetes service i.e AWS EKS (AWS), GCP GKE (GCP), Azure EKS (AWS) are configured for this design. These clusters are set up for efficient routing and management of traffic between cloud environments. These Kubernetes clusters are created in AWS EKS, GCP GKE, and Azure AKS using console of respective cloud platforms which can be done using CLI as well. While creating the clusters, the minimum nodes are set up to 2. Using the CI/CD pipeline via Jenkins, the application is deployed
   - CI/CD Pipeline Integration: For the Continuous Integration and Continuous Deployment, Jenkins is configured. This helps to automate the application deployment process across all three clouds. To host the Jenkins, AWS EC2 instance is used For more information on cloud services, see the official AWS documentation (AWS). This EC2 instance is configured with t2-medium machine type and enough volume mounts of memory, which helps Jenkins to work faster and seamless. Once the Jenkins is set up, the pipeline is created which includes the steps as follows:
     - Authentication: In this step, Jenkins try to authenticate all three cloud platforms and checks whether all the required permissions and security keys are in place to deploy the application.
     - Code Check-Out: Once the authentication is done, it goes on and checks out the code from GIT repository, and grabs the required files for application deployment. (eg. Deployment.yaml, service.yaml, dockerfile)

  - Docker Image: Once the files are in place, it goes on and build the docker image for the application using the dockerfile in repository.
  - Image push: The created docker image is then pushed to the container registeries of all the respective clouds which include AWS ECR, GCP GCR (Artifact registry) and Azure ACR, from where Kubernetes pulls the image.
  - Deployment: The image is pulled by Kubernetes cluster, and deployed as well as exposed externally on the NodePort which is mentioned in service.yaml file. (This can be changed as per the requirement Eg. LoadBalancer, Ingress)

  Once the steps are configured, this pipeline is set up to trigger using the github push action i.e when any changes are pushed in Git Repository related to the pipeline, it will trigger the pipeline and latest version of the application will be deployed. As shown in Figure,Figure 4 this Jenkins pipeline is set up to deploy the 'simple-app-application' in each cloud's Kubernetes cluster.

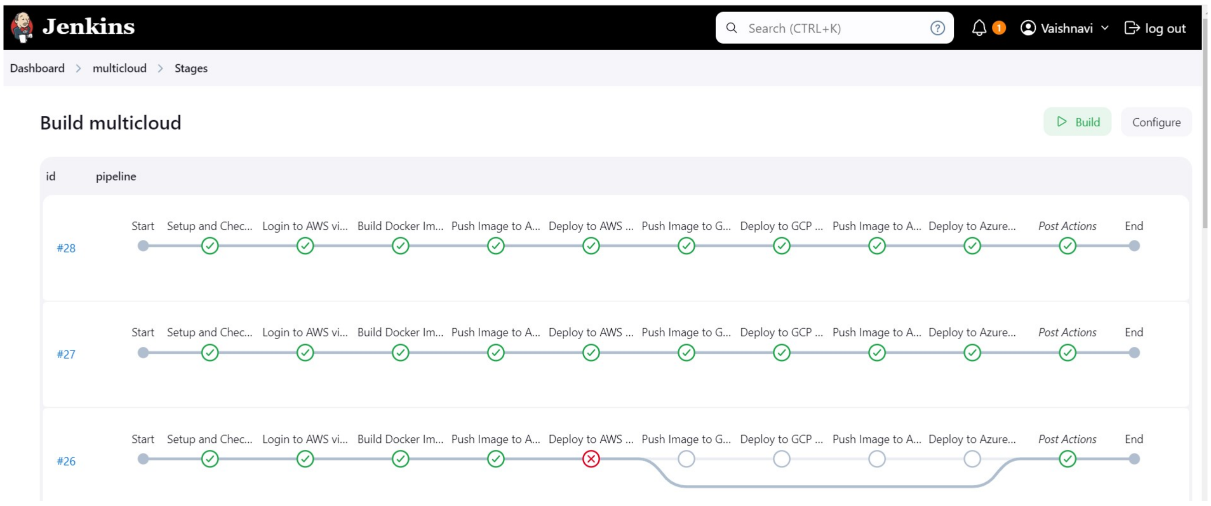


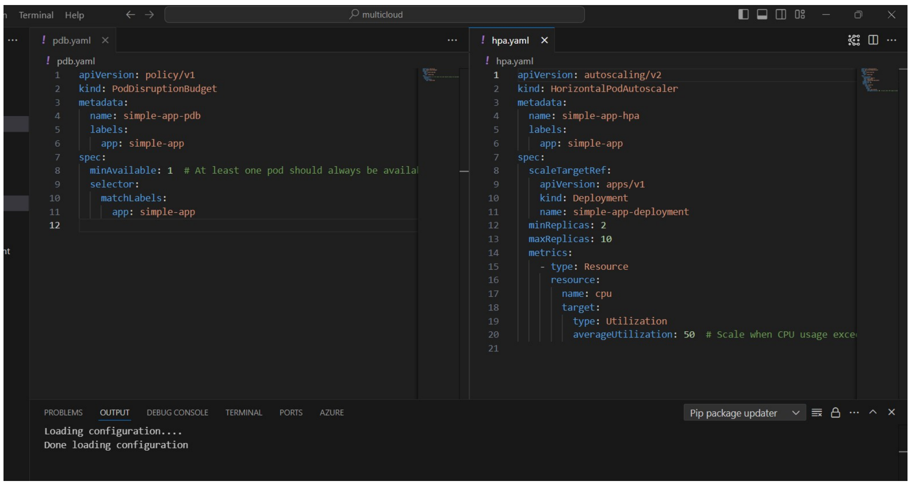
Figure 4: Jenkins Pipeline

- NodePort Configuration: Considering the scope of this work, external exposure of the application was not required, hence NodePort services are used to expose the application internally within clusters. This approach helped to minimize security risks and also ensured that the services are only accessible within the private cloud networks.

2. **Kubernetes Clusters and Operators Setup:** To automate the failure recovery within the multi-cloud environment, multiple Kubernetes operators are used. These operators extends the Kubernetes API to manage the life-cycle applications and resources automatically. The operators were configured to monitor the application states, detect anomalies and initiate the recovery process as well when required. These operators are set up as .yaml files along with the deployment.yaml and service.yaml files in the repository. Following mentioned steps were taken to implement the operators:

   - Custom Resource Definitions (CRDs): To manage the health of the deployed application, Custom resource was defined. This provides a declarative way to specify the desired state and life-cycle of the application. In this setup, PDB.yaml and HPA.yaml are used as CRDs.

- PDB.yaml: Pod Distribution Budget is the Kubernetes operator which makes sure that there will be at least two pods always running even if other pods crash or fail due to any issues.Figure 5



pdb.yaml

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: simple-app-pdb
  labels:
    app: simple-app
spec:
  minAvailable: 1  # At least one pod should always be availal
  selector:
    matchLabels:
      app: simple-app
```

hpa.yaml

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: simple-app-hpa
  labels:
    app: simple-app
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: simple-app-deployment
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50  # Scale when CPU usage exce
```

```
Loading configuration....
Done loading configuration
```

Figure 5: PDB.yaml & HPA.yaml

- HPA.yaml: Horizontal Pod Autoscaler is the operator used to scale up and scale down the number of pods according to the application demand and workload.Figure 5

• Operator Logic: Using the Go Programming Language, operators logic is implemented. This helps to watch for the pod failure events, resource exhaustion or even connectivity issues and take automated actions to recover from these. Considering, if one pod or node is deleted or crashed, the new pod is created automatically and takes up the workload.

• Deployment Strategy:These operators are deployed as a pod in all of the Kubernetes clusters and configured to work using shared Helm-charts for consistency across multi-cloud clusters.

3. **Datadog Integration:** To monitor the system health, detect failures in real-time, and provide observability across the multi-cloud deployment, Datadog is integrated into the framework which are explained in following steps:

   • Datadog Agent Setup: The Datadog Agent is deployed on each of the Kubernetes clusters. While dealing with multi-cloud clusters in a single instance, contexts are switched and datadog is installed in each of the Kubernetes clusters. To be precise, Datadog's API key and Application key are used to integrate clusters and datadog with each other. These agents collect metrics, logs and traces from the applications and infrastructure components and send them to the Datadog Platform. Figure 6

   • Custom Metrics Dashboards: Datadog has advantage that we can create our own metrics dashboards and choose our own Custom Metrics. Some of such metrics are created to track specific application and infrastructure health indicators. These metrics include pod health, CPU utilization, pod restarts count,

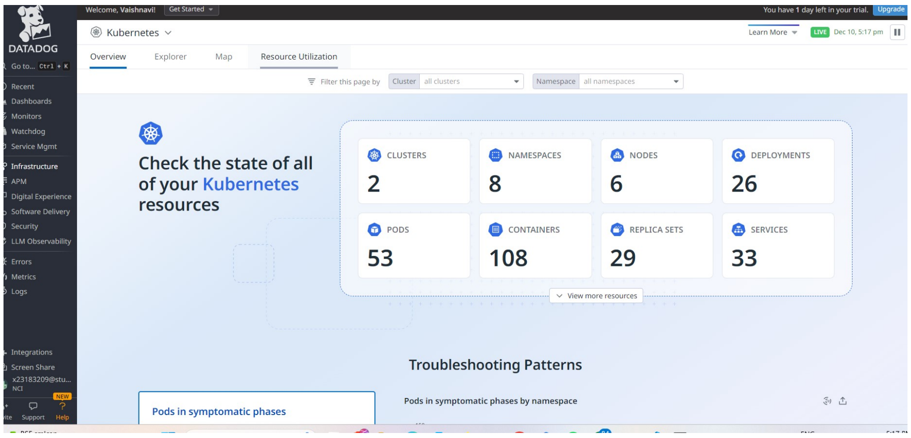


Figure 6: Datadog Dashboard

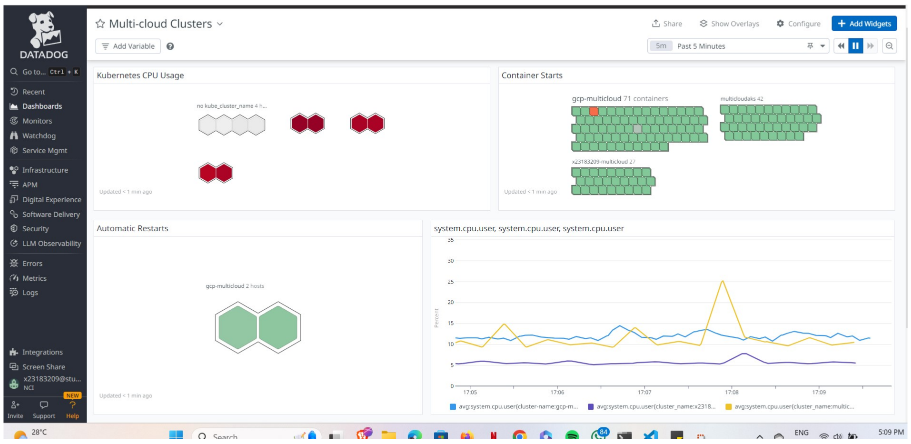


Figure 7: Datadog Metrics Dashboard

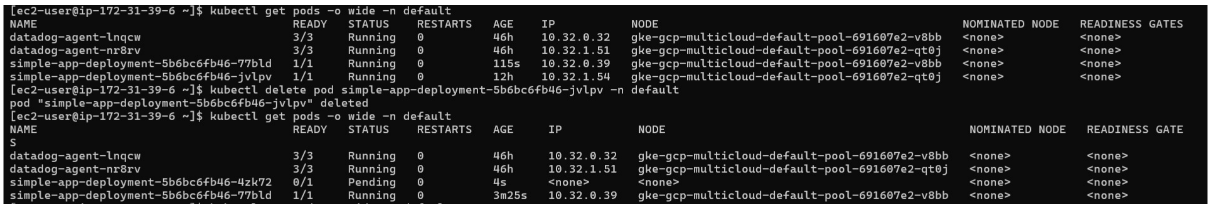
running containers and etc. This Datadog dashboard is configured to provide a real-time insights into these mentioned metrics, which allows DevOps team to identify potential failures. Figure 7

4. **Automated Failure Recovery Mechanisms:** The core component of this framework is the automated failure recovery system, Following are the failure recovery methods implemented:

   - Pod Auto-Scaling: Kubernetes' Horizontal Pod Autoscaler (HPA) is configured to scale up and down the application pods based on resource utilization metrics including CPY and Memory usage. This ensures that the system adapts to the changing workloads.In case of any pod failures, Kubernetes operator automatically triggers the pod restarts to restore the service
   - Pod Restarts: In case of any pod failures, Kubernetes operator automatically triggers the pod restarts to restore the service. Figure 8
   - Self-Healing workflow: If a critical failure such as a Node failure or a Network issue occurs and detected by Datadog, Kubernetes operators automatically initiate a sequence of actions, which includes scaling up additional pods, restarting the failed nodes and services or shifting workloads between different clusters to ensure the continuous availability.

5. **Testing and Validation:** Once the system is deployed, comprehensive testing was conducted to validate its functionality:

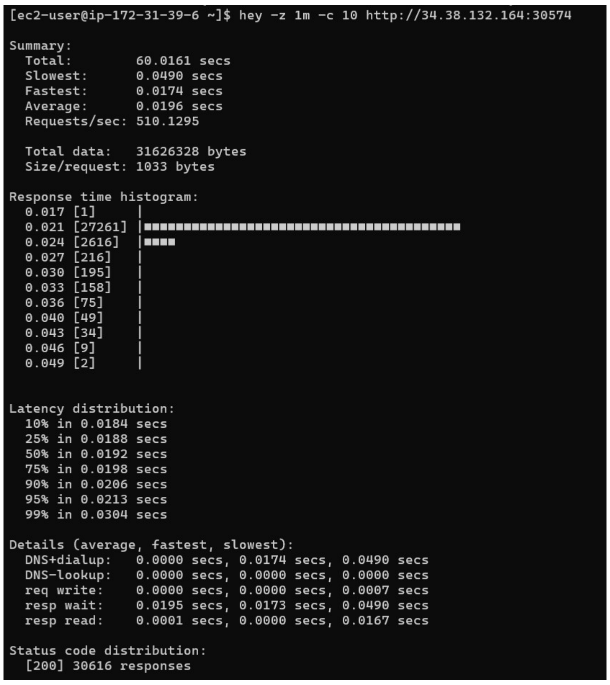
   - Failure simulation: To ensure the self-healing mechanisms work as expected, controlled failure scenarios were simulated, such as node shutdowns, Pod crashes, and network issues. The system's ability to detect from failures is measured. Figure 8



```
[ec2-user@ip-172-31-39-6 ~]$ kubectl get pods -o wide -n default
NAME                                     READY   STATUS    RESTARTS   AGE    IP           NODE                                             NOMINATED NODE   READINESS GATES
datadog-agent-lnqcw                      3/3     Running   0          46h    10.32.0.32   gke-gcp-multicloud-default-pool-691607e2-v8bb   <none>           <none>
datadog-agent-nr8rv                      3/3     Running   0          46h    10.32.1.51   gke-gcp-multicloud-default-pool-691607e2-qt0j   <none>           <none>
simple-app-deployment-5b6bc6fb46-77bld   1/1     Running   0          115s   10.32.0.39   gke-gcp-multicloud-default-pool-691607e2-v8bb   <none>           <none>
simple-app-deployment-5b6bc6fb46-jvlpv   1/1     Running   0          12h    10.32.1.54   gke-gcp-multicloud-default-pool-691607e2-qt0j   <none>           <none>
[ec2-user@ip-172-31-39-6 ~]$ kubectl delete pod simple-app-deployment-5b6bc6fb46-jvlpv -n default
pod "simple-app-deployment-5b6bc6fb46-jvlpv" deleted
[ec2-user@ip-172-31-39-6 ~]$ kubectl get pods -o wide -n default
NAME                                     READY   STATUS    RESTARTS   AGE     IP           NODE                                             NOMINATED NODE   READINESS GATE
S
datadog-agent-lnqcw                      3/3     Running   0          46h     10.32.0.32   gke-gcp-multicloud-default-pool-691607e2-v8bb   <none>           <none>
datadog-agent-nr8rv                      3/3     Running   0          46h     10.32.1.51   gke-gcp-multicloud-default-pool-691607e2-qt0j   <none>           <none>
simple-app-deployment-5b6bc6fb46-4zk72   0/1     Pending   0          4s      <none>       <none>                                           <none>           <none>
simple-app-deployment-5b6bc6fb46-77bld   1/1     Running   0          3m25s   10.32.0.39   gke-gcp-multicloud-default-pool-691607e2-v8bb   <none>           <none>
```

Figure 8: Pods Recreation

   - Performance Metrics: The performance of the multi-cloud system was monitored using the custom metrics created in the Datadog. Metrics such as resource utilization, downtime, are analyzed to assess the system's resilience and efficiency as well.
   - Scalability Testing: The system's ability to scale automatically in response to increased traffic or workload was tested by simulating a high load scenario across all three clouds. Figure 9



```
[ec2-user@ip-172-31-39-6 ~]$ hey -z 1m -c 10 http://34.38.132.164:30574

Summary:
  Total:        60.0161 secs
  Slowest:      0.0490 secs
  Fastest:      0.0174 secs
  Average:      0.0196 secs
  Requests/sec: 510.1295

  Total data:   31626328 bytes
  Size/request: 1033 bytes

Response time histogram:
  0.017 [1]     |
  0.021 [27261] |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.024 [2616]  |■■■■
  0.027 [216]   |
  0.030 [195]   |
  0.033 [158]   |
  0.036 [75]    |
  0.040 [49]    |
  0.043 [34]    |
  0.046 [9]     |
  0.049 [2]     |


Latency distribution:
  10% in 0.0184 secs
  25% in 0.0188 secs
  50% in 0.0192 secs
  75% in 0.0198 secs
  90% in 0.0206 secs
  95% in 0.0213 secs
  99% in 0.0304 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0000 secs, 0.0174 secs, 0.0490 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0000 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0007 secs
  resp wait:    0.0195 secs, 0.0173 secs, 0.0490 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0167 secs

Status code distribution:
  [200] 30616 responses
```

Figure 9: Workload Testing

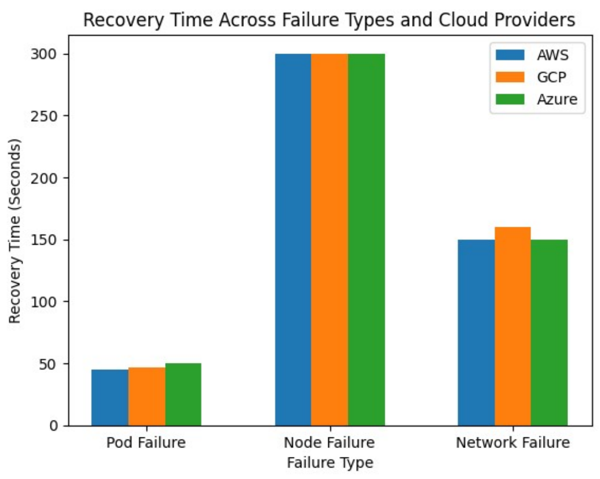


Figure 10: Recovery Time across Cloud Providers

5 Results

The implementation of self-healing framework for multi-cloud deployments on Kubernetes is evaluated based on several key factors which includes system availability, failure recovery time, scalability, and resource utilization. This section discusses the outcomes of the study, and explores the implications of the findings in the context of multi-cloud environments and self-healing systems.

- In all simulated failure scenarios, the system demonstrated the high availability. The Kubernetes operators effectively detect failures and trigger recovery actions, such as restarting failed pods or scaling services.
- The automated recovery mechanisms significantly minimizes downtime, with most of the failures being recovered automatically within few seconds depending on the complexity of failures.
- The average recovery time for failures was between 30 Seconds to 1 Minute, depending on the cloud provider and failure type.
- For node failures, recovery time was longer, taking up to 5 minutes to initiate the recovery by scaling application to a healthy node.
- The system successfully scaled the application based on the CPU and Memory utilization metrics, using Kubernetes Horizontal Pod Autoscaler (HPA)
- When system is under high load, it automatically adds up more replicas of application pods. Similarly, during the periods of low demand, unnecessary replicas are scaled down to save the costs and resources.
- The automated CI/CD pipeline successfully deploys application on multi-cloud Kubernetes clusters at once without any manual intervention except the authentication process.
- The system is able to detect the issues during deployment and automatically roll back changes when its necessary.

- The management of complex multi-cloud systems is easier when this framework is in place as the deployment of an application is handled through a single platform and the monitoring provides better insights into the system which makes it easy for DevOps engineers to work on the failures quickly.

In Figure Figure 10, Bar graph describes the recovery times for different failure types. This shows a quick comparison between pod failures, node failures, and network failures across included cloud providers.

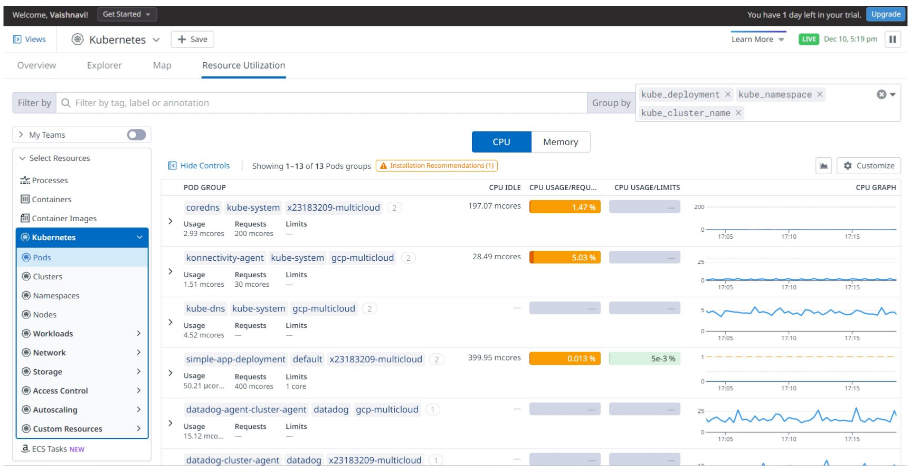
6 Evaluation and Discussion

The Evaluation of Self-Healing management Framework for Multi-Cloud deployments on Kubernetes primarily focuses on its effectiveness in achieving high availability, resilience, and automated failure recovery in a multi-cloud environment. The Evaluation process involves several key metrics:

1. System Availability: The framework designed maintains high availability across all of the simulated failures. Automatic recovery ensures that the system maintains the operational state with minimal downtime.

2. Failure Recover Time: The system demonstrates solid scalability, and it automatically adjusts to increasing demand by adding replicas of the application pods. However, network delays and communication issues between clouds were observed during scaling process which can be addressed by improving the cross-cloud communication strategies.

3. Resource Utilization: The system efficiently utilizes resources, with only temporary increases during recovery phases. Datadog's monitoring allows for better management of resources, which ensures that the system do not over-position resources unnecessarily. Although, during the large-scale recovery, resource optimization can still be improved.

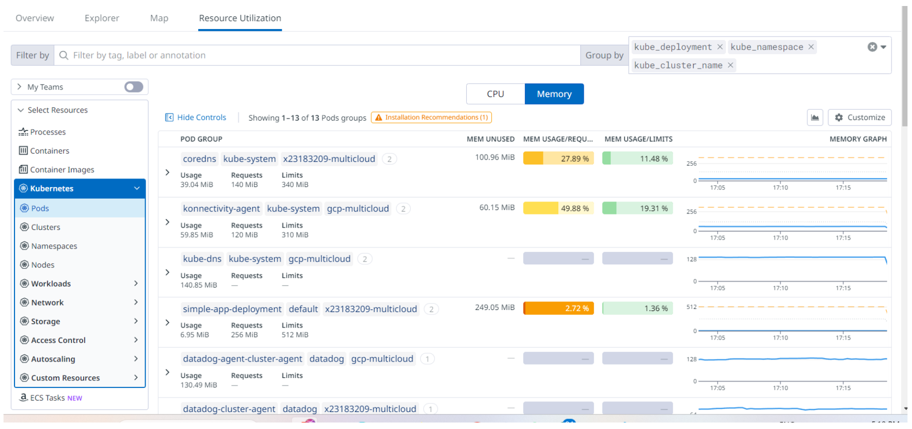


Group by: kube_deployment × kube_namespace × kube_cluster_name ×

Showing 1–13 of 13 Pods groups

POD GROUP	CPU IDLE	CPU USAGE/REQU...	CPU USAGE/LIMITS
coredns kube-system x23183209-multicloud (2) — Usage 2.93 mcores, Requests 200 mcores, Limits —	197.07 mcores	1.47 %	—
konnectivity-agent kube-system gcp-multicloud (2) — Usage 1.51 mcores, Requests 30 mcores, Limits —	28.49 mcores	5.03 %	—
kube-dns kube-system gcp-multicloud (2) — Usage 4.52 mcores, Requests —, Limits —	—	—	—
simple-app-deployment default x23183209-multicloud (2) — Usage 50.21 µcor..., Requests 400 mcores, Limits 1 core	399.95 mcores	0.013 %	5e-3 %
datadog-agent-cluster-agent datadog gcp-multicloud (1) — Usage 15.12 mco..., Requests —, Limits —	—	—	—
datadog-cluster-agent datadog x23183209-multicloud (1)	—	—	—

Figure 11: Resource Utilization (CPU)



Showing 1–13 of 13 Pods groups

POD GROUP	MEM UNUSED	MEM USAGE/REQU...	MEM USAGE/LIMITS	MEMORY GRAPH
coredns kube-system x23183209-multicloud 2 Usage 39.04 MiB, Requests 140 MiB, Limits 340 MiB	100.96 MiB	27.89 %	11.48 %	
konnectivity-agent kube-system gcp-multicloud 2 Usage 59.85 MiB, Requests 120 MiB, Limits 310 MiB	60.15 MiB	49.88 %	19.31 %	
kube-dns kube-system gcp-multicloud 2 Usage 140.85 MiB, Requests —, Limits —	—	—	—	
simple-app-deployment default x23183209-multicloud 2 Usage 6.95 MiB, Requests 256 MiB, Limits 512 MiB	249.05 MiB	2.72 %	1.36 %	
datadog-agent-cluster-agent datadog gcp-multicloud 1 Usage 130.49 MiB, Requests —, Limits —	—	—	—	
datadog-cluster-agent datadog x23183209-multicloud 1	—	—	—	

Figure 12: Resource Utilization (Memory)

4. Integration and Automation: Integration of Kubernetes Operators, Datadog and Jenkins streamlines the deployment process, automates the failure recovery, as well as provides the real-time monitoring. This enhances the overall reliability of the system. In future, improvements can include automating more granular failure response.

6.1 Challenges and Limitations:

- The system's recovery can sometimes be slower during large-scale infrastructures such as node failures shows longer recovery times.
- The complexity of managing self-healing mechanisms across multiple cloud environments increases with scale, which suggests the need for more efficient orchestration in larger deployments.
- Cross-cloud communication can be challenging and needs to be tested thoroughly.

7 Conclusion and Future Work

7.1 Conclusion

This study is focused on developing and management of the developed simple and deployable framework for the applications deployed on kubernetes in multiple cloud clusters. Considering the motive and objectives of this study, following conclusions can be derived:

- The implementation of the self-healing techniques in a multi-cloud environment using kubernetes operators, Datadog, and Jenkins is successfully done. The system proposed demonstrated high availability, automated failure recovery as well as the ability to scale dynamically based on the workloads of the application demands. The integration of these various tools allows for enhanced reliability, performance and resource optimization as well.
- However, while handling the complex failure scenarios and optimizing recovery times for node failures. Despite these challenges, the system performed very well and proved to be a robust solution for maintaining high availability

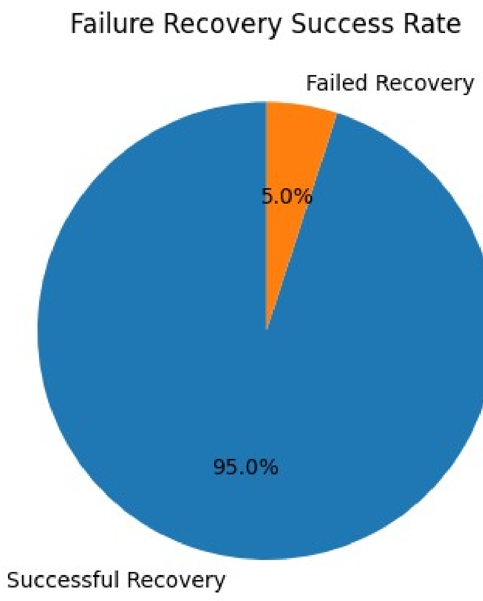


Figure 13: Failure Recovery Success Rate

- Management of multi-cloud deployments is easier when this framework is used as deployment processes are handled in a single framework and monitoring which is crucial is handled in separate and easy to use platform.

7.2 Future Work

This research has a lot of scope for future work for multiple enhancements. To improve the recovery times for node failures, future research can be focused on refining failure detection mechanisms and exploring more advanced failure recovery strategies. Further research into optimizing communication between clouds in a multi-cloud environment can address latency and network issues during scaling and recovery operations.
Implementing machine learning models to predict potential failures and automate response strategies can significantly improve the effectiveness of the self-healing framework. Exploring the integration of additional service mesh technologies, such as Istio can further enhance cross-cloud communications and improve system resilience.
The IaC i.e Infrastructure as a Code technology can be used while creating the cloud infrastructure using Terraform and other IaC tools which is considered as one of the best practices in DevOps Automation.

References

(AWS), A. W. S. (2024a). Amazon web services (aws). Accessed: 2024-12-12.
URL: *https://aws.amazon.com*

(AWS), M. A. C. (2024b). Azure cloud. Accessed: 2024-12-12.
URL: *https://portal.azure.com*

draw.io (2024). Online diagram created using draw.io. Accessed: 2024-12-12.
URL: *https://www.draw.io*

(GCP), G. C. P. (2024). Google cloud platform (gcp). Accessed: 2024-12-12.
URL: *https://cloud.google.com*

Magalhães, J. P. and Silva, M. L. (2013). A framework for self-healing and self-adaptation of cloud-hosted web-based applications, *IEEE International Conference on Cloud Computing Technology and Science* .

Mfula, H. and Norminen, K. J. (2018). Self-healing cloud services in private multi-clouds, *2018 International Conference on High Performance Computing Simulation* .

Monitoring, D. (2024). Datadog. Accessed: 2024-12-12.
URL: *https:datadoghq.eu*

Pandi, S. S., Kumar, P. and Suchindhar, R. M. (2023). Integrating jenkins for efficient deployment and orchestration across multi-cloud environments, *International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)* .

Pellegrini, R., Rottmann, P. and Strieder, G. (2017). Preventing vendor lock-ins via an interoperable multi-cloud deployment approach, *The 12th International Conference for Internet Technology and Secured Transactions (ICITST-2017)* .

Project, J. (2024). Jenkins: An open-source automation server. Accessed: 2024-12-12.
URL: *https://www.jenkins.io*

Samarakoon, S., Bandara, S., Jayasanka, N. and Hettiarachchi, C. (2023). Self-healing and self-adaptive management for iot-edge computing infrastructure, *Moratuwa Engineering Research Conference (MERCon)* .

SCM, G. (2024). Git: Distributed version control system. Accessed: 2024-12-12.
URL: *https://git-scm.com*

Sharma, V. (2022). Managing multi-cloud deployments on kubernetes with istio, prometheus and grafana, *8th International Conference on Advanced Computing and Communication Systems (ICACCS)* p. 10.

Ye, F., Wu, S., Huang, Q. and Wang, X. A. (2016). The research of enhancing the dependability of cloud services using a self-healing mechanism, *International Conference on Intelligent Networking and Collaborative Systems* .