International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Framework for Kubernetes High Availability and Resilience in Public Cloud

Harshavardhan Nerella[1], Prasanna Sai Puvvada[2], Praveen Borra[3]

[1]Independent Researcher,
Austin, Texas, USA
nerellaharshavardhan[at]outlook.com

[2]Independent Researcher,
Austin, Texas, USA
Prasannasaipuvvada[at]outlook.com

[3]Computer Science, Florida Atlantic University,
Boca Raton, Florida, USA
pborra[at]fau.edu

Abstract: ***High Availability (HA) and Resilience is crucial for applications to minimize downtime, ensure business continuity, and maintain consistent reliability. Kubernetes, as the de facto container orchestration platform, has been widely adopted by startups, enterprises, and government agencies. While Kubernetes does not offer HA and Resilience for clusters and workloads by default, it provides several mechanisms that can be used in tandem to achieve them. Managed Kubernetes services (e.g., AWS EKS, Azure AKS, Google GKE) offload the responsibility of HA and resiliency for the control plane, which serves as the "brain" of Kubernetes. However, the responsibility for the data plane and workloads still lies with cluster administrators and developers. This paper proposes a multi-layered approach to achieving HA and Resilience in managed Kubernetes clusters. We categorize the approach into two layers: workload and infrastructure layer. For each layer, we outline a series of mechanisms that must be adopted to ensure HA and Resilience. By applying these techniques sequentially at each layer, Kubernetes clusters can handle failures gracefully, scale efficiently, and maintain availability under varying workloads and infrastructure disruptions.***

Keywords: Kubernetes, High availability, Cloud Computing, Resilience

1. Introduction

In the rapidly evolving landscape of cloud-native technologies, Kubernetes has emerged as the leading platform for orchestrating containerized applications. Developed by Google and released as an open-source project in 2014, Kubernetes was inspired by Google's internal systems like Borg and Omega, which managed large-scale workloads across vast data centers [1]. Its name, derived from the Greek word for "helmsman" or "pilot," reflects its role in steering application deployment and scaling. Over the past decade, Kubernetes has been widely adopted by startups, enterprises, and government agencies, becoming the cornerstone of modern infrastructure management [2].

High Availability (HA) refers to a system's ability to remain operational and accessible even in the face of failures. Achieving HA involves eliminating single points of failure, implementing redundancy, and ensuring rapid failover mechanisms. The significance of HA cannot be overstated, as system downtimes can lead to substantial financial losses, damage to reputation, and erosion of customer trust. The absence of HA can have severe business repercussions. It is estimated that IT outages can cost organizations over $700 billion lost revenue and year in North America [3].

The importance of High Availability (HA) and Resilience becomes evident when examining significant outages that caused severe financial and operational disruptions. In 2021, Facebook (now Meta) faced a 5.5-hour outage, disrupting 3.5 billion users across Facebook, WhatsApp, and Instagram, resulting in $60 million in lost advertising revenue due to a misconfiguration error [4]. Similarly, Fastly's outage in 2021 caused major websites like Amazon, Reddit, and BBC to go offline when a configuration change triggered a system-wide bug, impacting 85% of its network [4]. These incidents underscore the devastating financial and reputational consequences of downtime, emphasizing the critical need for robust HA strategies to mitigate failures and ensure business continuity.

Kubernetes operates through a split architecture as shown in the figure 1, comprising the control plane and the data plane. The control plane serves as the cluster's "brain," managing the overall state, scheduling workloads, and orchestrating communication between components [5]. In managed Kubernetes services-such as Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), and Google Kubernetes Engine (GKE)-cloud providers assume responsibility for the control plane's HA and resiliency, ensuring its robustness and scalability [1].

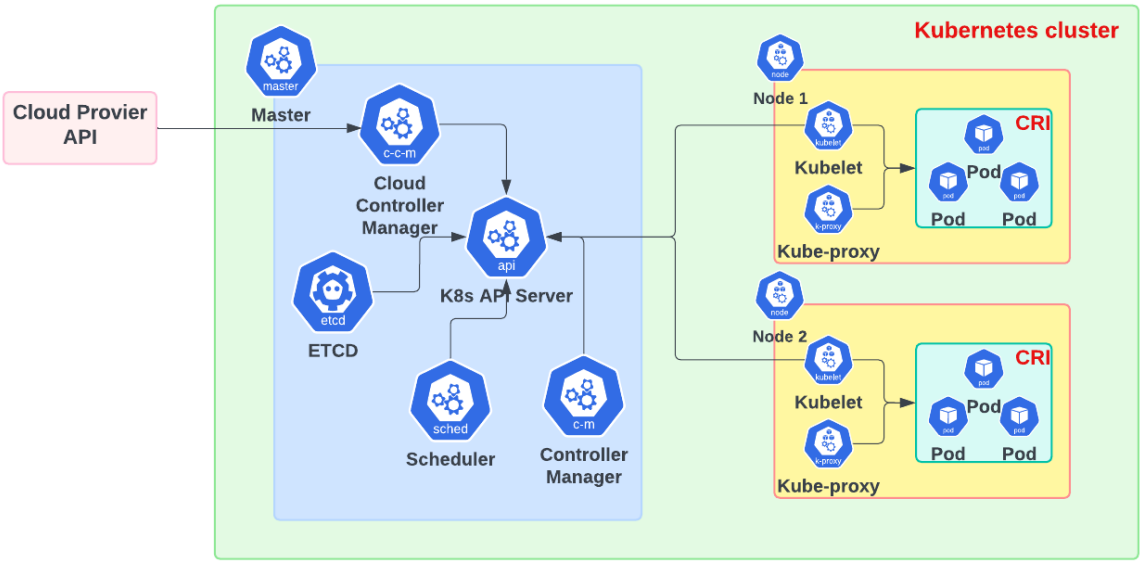

Figure 1: Kubernetes Architecture

Conversely, the data plane, encompassing the worker nodes and the workloads, remains under the responsibility of cluster administrators and developers. This delineation necessitates implementation of effective HA strategies at both the workload and infrastructure layers to ensure seamless application performance.

This paper proposes a multi-layered approach to achieving HA in managed Kubernetes clusters. We categorize the approach into two layers: the workload layer and the infrastructure layer. For each layer, we outline a series of mechanisms that must be adopted to ensure HA and Resilience. By applying these techniques sequentially at each layer, Kubernetes clusters can handle failures gracefully, scale efficiently, and maintain availability under varying workloads and infrastructure disruptions.

2. Literature Review

According to Burns (2018), Kubernetes is fundamentally designed for scalability and fault tolerance by allowing multiple replicas of workloads to be distributed across nodes [6]. However, Burns emphasizes that Kubernetes' default configurations alone do not guarantee comprehensive high availability, requiring additional strategies at workload and infrastructure layers.

Luksa (2017), in "Kubernetes in Action," emphasizes the inherent resiliency provided by managed control planes but notes that responsibility for ensuring the HA of workloads rests with cluster administrators [7].

Anemogiannis et al. (2025) proposed a novel anomaly detection and prediction framework to enhance Kubernetes resilience, leveraging graph-based representation and machine learning techniques to dynamically identify and manage anomalies within Kubernetes clusters. Their approach integrates unsupervised models to define normal operational states and supervised models for effective anomaly detection, thereby offering actionable insights for maintaining cluster availability [8].

While the aforementioned literature provides valuable insights, a gap remains in synthesizing and clearly outlining an integrated, multi-layered HA and Resilience strategy specifically for managed Kubernetes clusters. Existing studies often separately address workload or infrastructure HA aspects, leaving administrators without comprehensive guidance. This paper aims to bridge that gap by proposing a structured, cohesive framework encompassing both workload and infrastructure layers, designed specifically for managed Kubernetes environments in public cloud platforms.

3. Workload Layer

Achieving High Availability (HA) at the workload layer within managed Kubernetes clusters involves a comprehensive set of strategies to ensure continuous application availability and resilience. Below, we detail critical Kubernetes mechanisms that collectively provide robust workload resiliency:

3.1 Deployment

Deployments in Kubernetes manage stateless applications through replica sets, which specify the desired number of pod instances running concurrently. Applications should be managed as part of deployments rather than running as stand-alone pods. If an application supports multiple instances, they should be deployed at least two or more replicas to achieve high availability, redundancy and resilience. Utilizing parameters like maxSurge and maxUnavailable, Kubernetes ensures seamless rolling updates. maxSurge defines how many pods can be created above the desired count during updates, allowing workloads to scale temporarily, while maxUnavailable limits the number of pods that can be unavailable during updates, thereby guaranteeing minimum operational capacity.

3.2 Resource request and limits

Resource requests and limits in Kubernetes define the resource allocation boundaries for pods, ensuring

predictable performance and effective resource utilization within the cluster. Resource requests specify the minimum guaranteed resources (CPU and memory) a pod requires, guiding the scheduler in pod placement decisions. Conversely, resource limits define the maximum amount of resources a pod can consume, preventing excessive resource usage and draining the resources from other application pods. By configuring the limits and requests, Kubernetes automatically assigns pods to one of three Quality of Service (QoS) classes:

- **Guaranteed QoS**: Pods are assigned to this class when their resource requests equal their resource limits. These types of pods have the highest resource availability assurances and lowest eviction risk under resource pressure.
- **Burstable QoS**: Pods fall under this class when resource requests are specified but are lower than resource limits, allowing pods to temporarily consume additional resources if available. These pods have moderate resource assurances and are more susceptible to eviction compared to Guaranteed QoS pods.
- **Best-Effort QoS**: Assigned to pods without specified resource requests or limits, offering minimal resource availability guarantees. Pods in this class are the first candidates for eviction during resource contention, presenting the highest risk of disruption.

Configuring resource requests, limits is critical to ensuring workload and cluster stability and maximizing availability within Kubernetes clusters.

3.3 Health Check Probes

Kubernetes offers three essential health checks-Readiness, Liveness, and Startup probes-to monitor and maintain workload health:

- Readiness Probes verify if pods are ready to serve requests. Pods that fail readiness checks are removed from service endpoints, preventing downtime or service disruption.
- Liveness Probes determine if pods are responsive and functioning correctly. If a pod fails the liveness check, Kubernetes restarts it automatically, thus minimizing downtime.
- Startup Probes manage the startup time of pods, ensuring slow-starting pods aren't prematurely restarted before becoming fully operational.

Together, these probes proactively monitor pod health, automatically remediating issues, thereby ensuring continuous workload availability. In the following deployment manifest, the deployment is configured with a replica count of 3, a rolling update strategy with maxSurge and maxUnavailable set to 1 pod each, resource requests and limits, and readiness and liveness probes.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: workload-layer-deployment
  labels:
    app: workload-layer
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: workload-layer
  template:
    metadata:
      labels:
        app: workload-layer
    spec:
      containers:
      - name: workload-layer
        image: my-image:latest
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        readinessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 10
          periodSeconds: 5
          timeoutSeconds: 2
          successThreshold: 1
          failureThreshold: 3
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 15
          periodSeconds: 10
          timeoutSeconds: 2
          successThreshold: 1
          failureThreshold: 3
```

3.4 Tables Horizontal Pod Autoscaler (HPA)

Deployment helps to maintain a certain number of replicas at all times. Often, applications experience increases in load that existing replicas cannot handle adequately. Hence, applications deployed as deployments should be configured with Horizontal Pod Autoscaler (HPA). HPA dynamically adjusts the number of pod replicas based on real-time resource utilization metrics such as CPU or memory usage.

This elasticity allows Kubernetes workloads to respond efficiently to variable demand, reducing the risk of downtime during sudden spikes in traffic. In the below code snippet, the HPA is configured on the deployment to scale from 3 to 10 replicas if average cpu utilization is over 75.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: workload-layer-hpa
  labels:
    app: workload-layer
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: workload-layer-deployment
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 75
```

3.5 Pod Disruption Budget (PDB):

Pod Disruption Budgets (PDB) play a crucial role in maintaining application availability during voluntary disruptions, such as routine maintenance operations, cluster upgrades, or planned evictions. Technically, a PDB defines rules around how many pods of a particular deployment or StatefulSet can be safely disrupted simultaneously. Configure PDB by specifying either the minimum number of pods that must remain available (minAvailable) or the maximum number of pods that can be simultaneously unavailable (maxUnavailable) during disruptions. PDBs inform Kubernetes' eviction logic to respect these constraints, ensuring that critical workloads maintain a minimum operational capacity at all times. Without appropriately defined PDBs, routine maintenance or upgrades could inadvertently disrupt too many pods concurrently, leading to service degradation or downtime.

4. Infrastructure Layer

Achieving High Availability (HA) and resilience at the infrastructure layer within managed Kubernetes clusters requires meticulous cluster configuration, strategic node placement, appropriate use of third-party software, and a carefully designed cloud environment. Below, we detail essential infrastructure-level strategies and mechanisms that collectively ensure workload stability, cluster robustness, and continuous service availability:

4.1 Taints and Toleration

Kubernetes employs taints and tolerations as mechanisms to precisely control pod scheduling on designated nodes, thereby optimizing resource allocation and maintaining high cluster availability. Taints are applied to nodes, signifying specific conditions or dedicated roles, and preventing pods without corresponding tolerations from being scheduled on those nodes. Pods must explicitly declare tolerations matching node taints to be eligible for scheduling onto tainted nodes.

This feature is particularly critical in managing nodes reserved for specialized workloads-such as GPU-intensive tasks, security-sensitive applications, or infrastructure-level components like monitoring and logging services. By using taints and tolerations, administrators can isolate critical or resource-intensive workloads onto dedicated infrastructure, significantly reducing the risk of performance degradation due to resource contention.

4.2 Namespace Quotas

Namespace quotas set constraints on resource consumption (CPU, memory, etc.) within individual namespaces. Implementing quotas prevents a workload in a single namespace from monopolizing cluster resources, which could compromise the availability of other applications-e.g., evicting other pods and, in certain situations, potentially bringing down the entire cluster. Quotas enable predictable resource allocation and capacity planning, ensuring balanced and stable cluster performance.

4.3 Node Autoscaling

Node autoscaling is not an inherent feature of Kubernetes itself. However, open-source solutions like Cluster Autoscaler [10] or Karpenter facilitate automatic node scaling based on workload demand or capacity constraints within the cluster. These autoscalers dynamically provision additional nodes when existing resources are insufficient for scheduling new workloads or when workloads experience increased demand. Conversely, they also scale down nodes during periods of reduced application load, optimizing resource utilization and minimizing unnecessary operational costs.

In particular, advanced autoscaling tools such as Karpenter [9] individually assess workload resource requirements, proactively provisioning nodes tailored precisely to the needs of pending workloads. This targeted approach significantly reduces scheduling latency, enhances workload placement efficiency, and improves overall resource allocation. Node autoscaling thereby ensures consistent resource availability, maintaining optimal performance during fluctuating workloads and effectively preventing resource starvation, ultimately contributing to the robustness and availability of Kubernetes-managed applications.

4.4 Cloud Multi-AZ:

Deploying Kubernetes nodes across multiple availability zones (AZs) is foundational to achieving high availability at the infrastructure level. Multi-AZ deployments ensure that Kubernetes clusters are resilient against localized infrastructure failures, such as power outages, network disruptions, or hardware malfunctions within a single AZ.

Multi-AZ deployment involves evenly distributing nodes across distinct AZs within the cloud provider's region, thus ensuring that replicas of application pods are not co-located within a single failure domain. Kubernetes scheduling mechanisms like topology spread constraints help facilitate balanced distribution of pods across these nodes. Furthermore, we recommend using a min of three AZ in production. This approach mitigates the risk that a single AZ failure could affect all instances of critical workloads simultaneously.

4.5 Larger Instance types

Creating a cluster with larger instance types reduces cluster overhead, particularly in relation to interactions with the Kubernetes API server. Additionally, larger nodes accommodate more pods per node, thereby minimizing node-to-API server communication and reducing overall cluster management overhead. Consequently, this approach decreases the API server load, enhancing the stability and responsiveness of cluster management operations.

5. Conclusion

Achieving High Availability (HA) and resilience in managed Kubernetes clusters deployed in public cloud environments requires a systematic, multi-layered strategy. Throughout this paper, we have emphasized that Kubernetes, despite its robust architecture and widespread adoption, does not inherently guarantee high availability at either the workload or infrastructure layers. Responsibility for ensuring HA at these layers rests with cluster administrators and developers.

In addressing both workload and infrastructure layers, critical Kubernetes mechanisms such as deployments, Horizontal Pod Autoscaler (HPA), Pod Disruption Budgets (PDBs), Pod Topology Spread Constraints, Pod Priority and Preemption, resource requests and limits, health check probes, Taints and Tolerations, Namespace Quotas, Node Autoscaling, Multi-AZ node distribution, and optimal node sizing collectively ensure application stability, operational continuity, and enhanced cluster resilience. Proper implementation of these strategies enables workloads to dynamically scale, gracefully handle failures, maintain performance, and mitigate infrastructure-level risks.

Ultimately, the effective integration and rigorous implementation of these workload and infrastructure strategies form a robust framework for achieving comprehensive HA. As Kubernetes continues to dominate container orchestration, the importance of proactively addressing potential failure points through these strategies cannot be overstated. Adopting this holistic approach will empower organizations to sustain business continuity, minimize downtime, and reliably deliver resilient cloud-native services.

References

[1] IBM, “The History of Kubernetes.” [Online]. Available: https://www.ibm.com/think/topics/kubernetes-history
[2] Kubernetes Blog, “10 Years of Kubernetes,” Jun. 6, 2024. [Online]. Available: https://kubernetes.io/blog/2024/06/06/10-years-of-kubernetes/
[3] Insights for Professionals, “The Real Cost of IT Outages & Disruptions.” [Online]. Available: https://www.insightsforprofessionals.com/it/leadership/the-real-cost-of-it-outages-disruptions
[4] Insights for Professionals, “Famous Business Outages.” [Online]. Available: https://www.insightsforprofessionals.com/it/leadership/famous-business-outages
[5] Wikipedia, “Kubernetes.” [Online]. Available: https://en.wikipedia.org/wiki/Kubernetes
[6] B. Burns, Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. Sebastopol, CA: O’Reilly Media, 2018. [Online]. Available: https://www.amazon.com/Designing-Distributed-Systems-Paradigms-Scalable/dp/1491983647
[7] M. Luksa, Kubernetes in Action. Shelter Island, NY: Manning Publications Co., 2017. [Online]. Available: https://www.manning.com/books/kubernetes-in-action
[8] V. Anemogiannis, B. Andreou, K. Myrtollari, K. Panagidi, and S. Hadjiefthymiades, “Enhancing Kubernetes Resilience through Anomaly Detection and Prediction,” arXiv preprint, arXiv:2503.14114, 2025. [Online]. Available: https://arxiv.org/abs/2503.14114
[9] Karpenter Documentation. [Online]. Available: https://karpenter.sh/docs/
[10] Kubernetes Autoscaler GitHub Repository. [Online]. Available: https://github.com/kubernetes/autoscaler
[11] Learnk8s, “How to Choose the Right Node Size for Kubernetes.” [Online]. Available: https://learnk8s.io/kubernetes-node-size