SISTEKINFO
JURNAL SISTEM DAN TEKNOLOGI INFORMASI
e-ISSN : 3109 6522

Volume 2 • Number 2 • September 2025 • 17 - 28

Design of a High Availability Infrastructure Based on Microservices Using an Infrastructure as Code (IaC) Approach with Terraform and Kubernetes

Nasywaa Putricia Ananda[1], Ilham Caesar Dwianto Prakoso[2], Isa Faqihuddin[3]

[123]Information Systems and Technology Study Program, Faculty of Industrial Technology and Informatics, Muhammadiyah University Prof. Dr. HAMKA

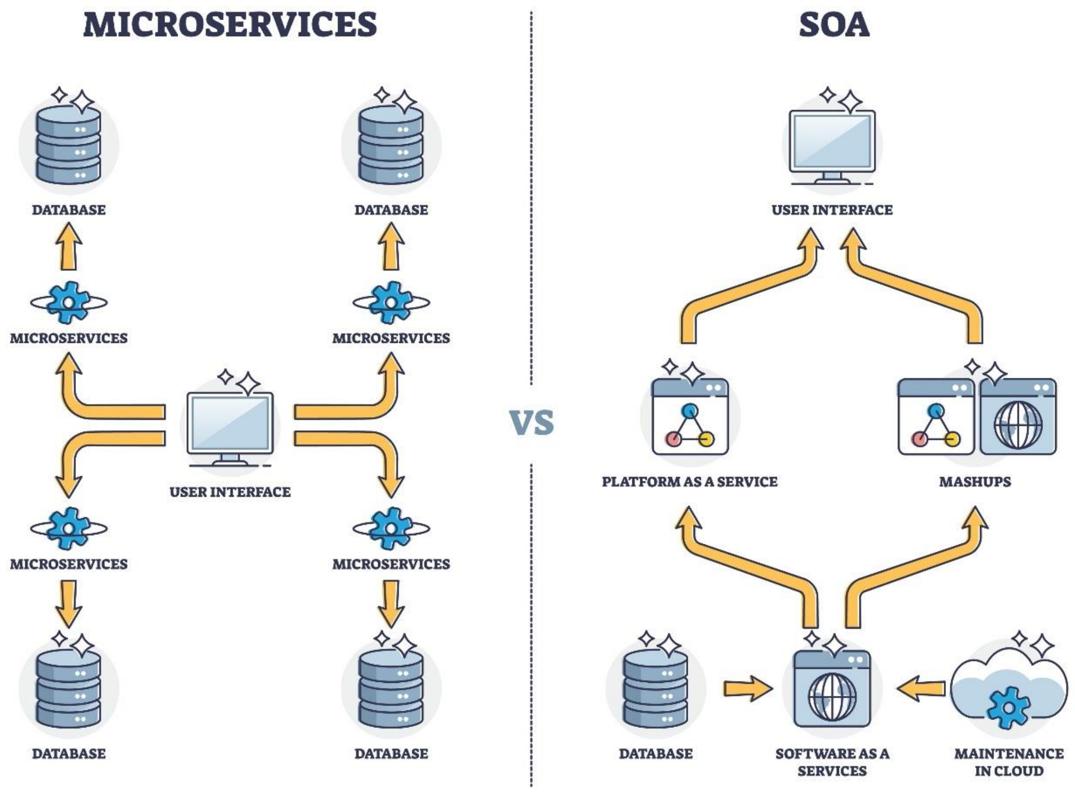
Received: December 15, 2025 Accepted: December 15, 2025 Published: December 15, 2025

Abstract

The development of modern software architecture has shown a significant shift from monolithic approaches to microservices architecture in order to support scalability and flexibility of information systems (Newman, 2021). However, the adoption of microservices increases the complexity of infrastructure management due to the distributed nature of the components, thereby requiring automation and guarantees of service availability, commonly referred to as High Availability (Burns et al., 2019). Without proper management, this complexity can lead to single points of failure and configuration drift. This study aims to theoretically examine the design of High Availability infrastructure using the Infrastructure as Code (IaC) approach with Terraform and container orchestration using Kubernetes (HashiCorp, 2024; Kubernetes Authors, 2024). The research method employed is a systematic literature study with a descriptive-analytical approach, based on scientific publications, official documentation, and relevant reference books. The findings indicate that the combination of Terraform and Kubernetes conceptually supports configuration consistency through immutability principles, automated infrastructure provisioning across multi-availability zones, and precise self-healing mechanisms (Morris, 2021). Therefore, the IaC approach in microservices architecture has the potential to provide a reliable, efficient, and resilient infrastructure solution capable of meeting high service availability requirements (above 99.9%)..

Keywords: High Availability, Microservices, Infrastructure as Code, Terraform, Kubernetes, DevOps..

INTRODUCTION



Shutterstock

The need for information systems that are always available (always-on systems) has become a primary demand in the era of digital transformation, where even minutes of downtime can result in significant financial losses and reputational damage (Armbrust et al., 2010). Traditionally, applications were built using a monolithic architecture in which the user interface, business logic, and data access were combined into a single executable unit. Although simple in the early stages, centralized monolithic architectures often face limitations in scalability and system resilience, where the failure of a small module (for example, a memory leak in a reporting feature) can bring down the entire application (Richards, 2020).

As an alternative, microservices architecture was introduced with the concept of decomposing applications into small, independent services that are loosely coupled and integrated through APIs (Newman, 2021). However, the transition to microservices introduces new challenges, commonly referred to as "The Distributed System Fallacy." Managing dozens to hundreds of services running across multiple servers (nodes) manually is impractical and highly prone to human error. Burns et al. (2019) emphasize that the greatest challenge in microservices lies not in writing application code, but in managing the underlying infrastructure to ensure configuration consistency and service availability (High Availability).

A classic problem that frequently arises is the "Snowflake Server," where servers are manually and uniquely configured, making them difficult to replicate in the event of a disaster. Therefore, the Infrastructure as Code (IaC) approach becomes highly relevant. IaC treats infrastructure components—such as networks, servers, load balancers, and topology—as software code that can be tested, documented, and version-controlled (Morris, 2021).

Research Objectives

This study aims to formulate a theoretical foundation for designing reliable infrastructure by integrating two industry-standard technologies: Terraform for physical infrastructure provisioning and Kubernetes for application orchestration. The primary focus is on how these two technologies can collaborate to create an environment that supports High Availability principles without requiring manual intervention when system failures occur.

Theoretical Framework

Microservices Architecture

Microservices is an architectural style that structures an application as a collection of services that are:

Independently deployable

Organized around business capabilities

Owned by a small team

Highly observable

In the context of infrastructure, microservices require a dynamic networking layer because service locations (IP addresses) can change at any time (Bass et al., 2021). This condition necessitates the presence of a reliable Service Discovery mechanism.

High Availability (HA) Concept

High Availability (HA) refers to a system's ability to remain operational continuously over a long period compared to its normal operating time. HA is typically measured as a percentage of uptime (e.g., 99.99% or "four nines"), which corresponds to a downtime tolerance of less than 52 minutes per year (Mell & Grance, 2011).

The main principles for achieving HA include:

Redundancy: Eliminating Single Points of Failure (SPOF) by duplicating critical components (N+1 or N+M redundancy).

Automatic Failover: The system's ability to redirect traffic from failed components to healthy ones without human intervention.

Geographic Distribution: Distributing services across multiple availability zones to withstand physical disasters affecting a single data center location.

Infrastructure as Code (IaC) and Terraform

Infrastructure as Code (IaC) is the practice of managing IT infrastructure using machine-readable configuration files rather than physical hardware configuration or interactive configuration tools.

Terraform, developed by HashiCorp, is a cloud-agnostic IaC tool that adopts a declarative approach.

Declarative vs. Imperative: In an imperative approach, users specify the steps required to achieve a goal. In a declarative approach (as used by Terraform), users define the desired end state (Desired State), and the tool determines how to achieve it (HashiCorp, 2024).

State Management: Terraform stores the infrastructure state in a state file (.tfstate), enabling the mapping between real-world resources and code configurations, as well as detecting configuration changes (drift detection).

Kubernetes (K8s)

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. The Kubernetes architecture consists of:

Control Plane (Master Node): Manages the cluster, makes global decisions (such as scheduling), and detects and responds to cluster events. Its main components include kube-apiserver, etcd (a key-value store for cluster data), kube-scheduler, and kube-controller-manager.

Worker Node: Machines that run applications. These consist of the kubelet (an agent that communicates with the master), kube-proxy (networking rules), and a container runtime (such as Docker or containerd) (Kubernetes Authors, 2024).

A key Kubernetes feature for HA is Self-Healing: if a Pod (the smallest unit in Kubernetes) fails, Kubernetes will immediately reschedule a new Pod to replace it in order to maintain the Desired State.

METHODS

This study employs a Systematic Literature Review (SLR) method with a qualitative–descriptive approach. This method is chosen to synthesize a comprehensive understanding from various trusted sources without conducting laboratory experiments, given that the focus of the research is on conceptual architecture design.

Research Stages

Problem Identification: Analyzing the gap between High Availability (HA) requirements in microservices architectures and traditional infrastructure management methods.

Literature Search: Collecting data from international journal databases (IEEE Xplore, ACM Digital Library), official technology documentation (HashiCorp, Kubernetes.io), and reference books (O'Reilly, Addison-Wesley) published between 2018 and 2024.

Source Selection: Sources are selected based on relevance to the following keywords: "Microservices Infrastructure," "Terraform Best Practices," "Kubernetes High Availability Architecture," and "DevOps Automation."

Analysis and Synthesis: Comparing concepts, identifying architectural patterns, and synthesizing a theoretical model for integrating Terraform and Kubernetes.

Conceptual Design: Formulating an architectural design based on best practices identified in the literature (Kitchenham et al., 2009).

FINDINGS AND DISCUSSION

GENERAL OVERVIEW OF RESPONDENT DATA

This section discusses the theoretical design for integrating Terraform and Kubernetes to achieve a resilient infrastructure.

Core Infrastructure Design with Terraform

To support High Availability, the core infrastructure must be built on a redundant networking foundation. Terraform plays a role in providing the physical infrastructure layer (provisioning layer). Based on the literature (Brikman, 2019), the recommended design includes:

Virtual Private Cloud (VPC): Logical network isolation.

Multi-Availability Zones (Multi-AZ): Terraform is configured to create subnets in at least two or three physically separate availability zones. If Zone A experiences a power outage, Zones B and C remain operational.

Load Balancer: Provisioning an Application Load Balancer (ALB) or Network Load Balancer (NLB) to distribute incoming traffic across all availability zones.

Managed Kubernetes Service: Using Terraform to provision a managed Kubernetes service (such as Amazon EKS, Google GKE, or Azure AKS). Managed services are recommended because the provider guarantees control plane availability, allowing users to focus solely on worker nodes.

Terraform Code Modularity: Terraform code should be modular (using Terraform Modules) to ensure reusability and standardization across multiple environments (Development, Staging, Production).

High Availability Orchestration Strategies with Kubernetes

Once Terraform provisions the cluster, Kubernetes assumes responsibility for application orchestration. High Availability mechanisms within Kubernetes are achieved through multiple layers (Luksa, 2017):

Pod Replication (ReplicaSets)

Each microservice should not run as a single instance. Deployments must be configured with replicas: N (where N > 1). This ensures that if one container fails, others continue to serve requests.

Pod Anti-Affinity

To prevent scenarios where all Pod replicas run on the same node (server), Pod Anti-Affinity rules are applied. This configuration forces the Kubernetes scheduler to distribute Pods across different nodes, ideally across different availability zones.

YAML

```
affinity:
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
            - key: app
              operator: In
              values:
              - frontend
        topologyKey: "topology.kubernetes.io/zone"
```

(Contoh konseptual aturan Anti-Affinity)

Liveness and Readiness Probes

Kubernetes monitors application health using probes.

Liveness Probe: Checks whether the application is running. If the probe fails, Kubernetes terminates and restarts the container (self-healing).

Readiness Probe: Checks whether the application is ready to receive traffic. If it fails, the Pod IP is removed from the Service endpoint, ensuring that users are not routed to an unready Pod.

Horizontal Pod Autoscaler (HPA)

High Availability also means maintaining availability during traffic spikes. The HPA automatically increases the number of Pods based on CPU or memory utilization, ensuring that services do not go down due to overload.

Terraform and Kubernetes Workflow Integration

The synergy between Terraform and Kubernetes creates a DevOps workflow commonly known as GitOps or an Infrastructure as Code pipeline.

Provisioning Phase (Terraform)

Administrators define cluster infrastructure, networking, and security in a Git repository. A CI/CD pipeline executes terraform plan and terraform apply. The result is a ready-to-use Kubernetes cluster with nodes distributed across multiple availability zones.

Deployment Phase (Kubernetes/Helm)

Once the infrastructure is ready, Kubernetes manifests (YAML) or Helm charts are applied to the cluster. Terraform can be used to manage the Helm provider, or a separate continuous delivery tool such as ArgoCD can be employed.

Maintenance Phase (State Reconciliation)

If manual changes occur (for example, someone deletes a firewall rule manually), running terraform plan again allows Terraform to detect configuration drift and restore the infrastructure to the state defined in code. This guarantees long-term infrastructure integrity.

Advantages of the Integrated Approach

Based on a comparative analysis by Morris (2021) and Sayfan (2020), this approach offers:

Idempotency: The same code can be executed repeatedly with consistent results, eliminating environmental variability.

Audit Trail: Since all infrastructure is defined as code (in Git), every change is recorded (who, what, when), enhancing security and compliance.

Disaster Recovery (DR): In a total disaster scenario, the entire infrastructure can be rebuilt in a new region within minutes by executing the same Terraform scripts.

Challenges and Mitigation Strategies

Despite offering significant High Availability benefits, implementing this approach involves theoretical challenges that must be anticipated:

State Management Complexity

Terraform relies on a sensitive state file. If this file is corrupted or lost, synchronization with the actual infrastructure is disrupted.

Mitigation: Use a remote backend (such as AWS S3 with DynamoDB locking or Terraform Cloud) that supports encryption and state locking to prevent conflicts when multiple engineers work simultaneously.

Learning Curve

The combination of Terraform (HCL language) and Kubernetes (YAML manifests) introduces high cognitive complexity. Misconfigurations in Kubernetes (for example, incorrect resource limits) can actually lead to downtime.

Mitigation: Implement modular standards and automated code validation (linting) within the CI/CD pipeline before deploying code to production.

Security

Storing infrastructure definitions as code poses the risk of exposing credentials (API keys, database passwords) if they are hardcoded.

Mitigation: Integrate with secret management tools such as HashiCorp Vault or AWS Secrets Manager, where Terraform references only the secret locations rather than the secret values themselves.

CONCLUSION

Based on an in-depth theoretical review, it can be concluded that the design of modern microservices-based infrastructure can no longer rely on manual configuration. The integration of Terraform as an Infrastructure as Code (IaC) provisioning tool and Kubernetes as a container orchestrator represents a strategic combination for achieving High Availability.

Terraform ensures a consistent physical foundation that is geographically distributed (Multi-AZ) and rapidly recoverable (Disaster Recovery). Meanwhile, Kubernetes enhances the application layer with self-healing, auto-scaling, and intelligent load-balancing capabilities. The synergy between the two creates a resilient, transparent, and manageable infrastructure ecosystem that is theoretically capable of meeting service availability Service Level Agreements (SLAs) above 99.9%.

Recommendations

For future research, it is recommended to:

Conduct a technical implementation in the form of a prototype to measure performance metrics such as Recovery Time Objective (RTO) and Recovery Point Objective (RPO) during simulated failure scenarios.

Examine the integration of observability approaches (monitoring, logging, and tracing) using Prometheus and Grafana into Terraform scripts to complete the infrastructure lifecycle.

Analyze cost efficiency aspects (cost optimization) of implementing High Availability infrastructure across different cloud service providers.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A View of Cloud Computing. Communications of the ACM, 53(4), 50-58.

Arundel, J., & Domingus, J. (2019). Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud. O'Reilly Media.

Bass, L., Weber, I., & Zhu, L. (2021). DevOps: A Software Architect's Perspective. Addison-Wesley Professional.

Brikman, Y. (2019). Terraform: Up & Running: Writing Infrastructure as Code. O'Reilly Media.

Burns, B., Beda, J., & Hightower, K. (2019). Kubernetes: Up and Running: Dive into the Future of Infrastructure. O'Reilly Media.

Davis, J. (2020). Cloud Native Patterns: Designing Change-tolerant Software. Manning Publications.

Fowler, M. (2018). Microservices: A definition of this new architectural term. martinfowler.com.

Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley.

Kim, G., Behr, K., & Spafford, G. (2018). The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win. IT Revolution Press.

Luksa, M. (2017). Kubernetes in Action. Manning Publications.

Morris, K. (2021). Infrastructure as Code: Dynamic Systems for the Cloud Age. O'Reilly Media.

Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems. 2nd Edition. O'Reilly Media.

Richards, M., & Ford, N. (2020). Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media.

Sayfan, G. (2020). Mastering Kubernetes. Packt Publishing.

HashiCorp. (2024). Terraform Documentation: Infrastructure as Code. Tersedia di: https://www.terraform.io/docs

Kubernetes Authors. (2024). Kubernetes Documentation. The Linux Foundation. Tersedia di: https://kubernetes.io/docs

Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. National Institute of Standards and Technology (NIST), Special Publication 800-145.

CNCF (Cloud Native Computing Foundation). (2023). Cloud Native Landscape. Tersedia di: https://landscape.cncf.io/