Industry Perspectives on Kubernetes: A Multi-Method Empirical Analysis of Adoption Drivers, Usage Patterns, and Ecosystem Activity

by

Umme Jannat Taposhi
24166045

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
M.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
Septebmer 2025

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Umme Jannat Taposhi

24166045

Approval

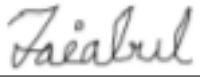
Industry Perspectives on Kubernetes: A Multi-Method Empirical Analysis of Adoption Drivers, Usage Patterns, and Ecosystem Activity

Umme Jannat Taposhi(24166045)

Of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of M.Sc. in Computer Science and Engineering on September 5, 2025.

Examining Committee:

Supervisor:
(Member)



S M Taiabul Haque, PhD

Associate Professor
Department of Computer Science and Engineering
Brac University

Examiner:
(Internal)

Farida Chowdhury, PhD

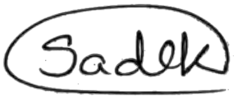
Professor
Department of Computer Science and Software Engineering
Brac University

Examiner:
(External)

Akond Rahman, PhD

Assistant Professor
Department of Computer Science Software Engineering
Auburn University

Program Coordinator:
(Member)



Md. Sadek Ferdous, PhD

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD

Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

This study was conducted with adult software professionals and through the analysis of publicly available GitHub repositories. Participation in the interviews and survey was entirely voluntary. Participants provided informed consent, could decline to answer any question, and were free to withdraw at any time. With permission, interviews were recorded only for accurate transcription; transcripts were de-identified, and any quotations reported here omit personally identifying or company-specific details. Optional contact emails for a gift-card draw were stored separately from the research data and permanently deleted after prize notifications.

The repository-mining analysis used only publicly available data, complied with platform terms of service and license requirements, and findings are reported in aggregate form. The study protocol was designed and reviewed by Dr. S. M. Taiabul Haque, and it received approval from the Institutional Review Board (IRB) of BRAC University, ensuring compliance with established ethical standards for minimal-risk, non-interventional research.

Abstract

Kubernetes has rapidly become the backbone of cloud native computing, yet we still know relatively little about how it is adopted and practiced in day-to-day software engineering work. Despite its central role in modern infrastructure, systematic empirical research that captures both practitioner experiences and open-source evidence has been largely absent. To address this gap, we conducted a mixed-method study combining 22 semi-structured practitioner interviews, survey responses from 31 professionals, and an analysis of 1,429 public repositories. Together, these sources provide a comprehensive view of Kubernetes adoption, usage, and challenges. The interviews informed the design of the survey, which in turn guided our analysis of repositories. By linking practitioner perspectives with the technical evidence from repositories, we were able to observe which practices appeared frequently, which were less common, and which showed variation between the two sources. Our findings highlight the widely shared practices as well as gaps where aspirational best practices are not consistently reflected in real-world repositories. We also highlight ongoing five key challenges that illustrate how Kubernetes remains a maturing ecosystem rather than a fully developed technology. Taken together, this work contributes to a deeper understanding of Kubernetes in practice. For practitioners, it offers lessons for navigating adoption and maintenance, while for researchers, it provides a unique empirical dataset that links qualitative insights with quantitative repository evidence.

Keywords: Kubernetes, Cloud-Native Systems, Adoption Drivers, Workloads, Infrastructure-as-Code (IaC), Scheduling Features, Logging and Monitoring Tools, Repository Mining, DevOps, Empirical Software Engineering, Practitioner Study, Triangulation.

Dedication

I dedicate this thesis to my husband, Md. Tawhid Anwar, whose steady support, patience, and constant encouragement have guided and inspired me at every step. Thank you for lifting me up, believing in me, and inspiring me always.

Acknowledgement

First and foremost, all praise is due to Allah. His blessings, guidance, and mercy sustained me throughout this journey and enabled me to complete this thesis.

I am profoundly grateful to my supervisor, Dr. S. M. Taiabul Haque, for his unwavering support, thoughtful guidance, and steady mentorship. His insightful feedback and encouragement were instrumental at every stage, and his consistent presence helped me navigate challenges with confidence.

My sincere thanks to Dr. Akond Rahman (Auburn University) for his invaluable contributions. His expertise and advice significantly strengthened the rigor and depth of this work.

I extend my heartfelt appreciation to my defense panel members, Dr. Farida Chowdhury and Dr. Md. Sadek Ferdous, for their careful review and constructive suggestions. Their feedback sharpened the final manuscript and helped bring it to completion.

Finally, my deepest gratitude goes to my parents. Their constant support, encouragement, and prayers have been the foundation of my academic journey. This achievement would not have been possible without their love and dedication.

This thesis stands on the collective effort, guidance, and generosity of all these remarkable individuals. I am deeply indebted to each of them.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Rise of Kubernetes as a Standard in Cloud-Native Infrastructure

Kubernetes plays a pivotal role in contemporary software engineering, serving as the de facto standard for managing containerized applications. Kubernetes, released as open-source by Google in 2014, has emerged as the leading platform for coordinating containerized applications across on-premise, hybrid, and multi-cloud environments [1]. As enterprises progressively adopt containerization for scalable application development and deployment, Kubernetes has emerged as the predominant orchestration platform. Orchestration is basically a system that automates the deployment, scheduling, and management of containers. Kubernetes does this orchestration across diverse infrastructure environments, for managing these containers across cloud, hybrid, and on-premise environments [1]. Kubernetes is appealing due to its promise of flexibility, scalability, and resilience; it automates deployment, scaling, and networking processes that were previously laborious and prone to errors, hence facilitating reliable service delivery by engineering teams. Supported by the Cloud Native Computing Foundation (CNCF), Kubernetes has evolved into a dynamic ecosystem of tools, extensions, and community practices that influence contemporary software development and operation [2].

This transition signifies a broader evolution in software engineering. The conventional period of monolithic programs is swiftly transitioning to microservice architectures, wherein systems are deconstructed into smaller, independently deployable components [3]. Microservices facilitate agility and expedited release cycles; nevertheless, they also increase complexity, necessitating the deployment, monitoring, and scaling of numerous services in unison. Kubernetes mitigates this difficulty by abstracting infrastructure specifics and offering integrated mechanisms for service discovery, load balancing, automated rollouts, and self-healing [4]. Kubernetes, when integrated with DevOps principles, Infrastructure-as-Code (IaC), and continuous integration/continuous deployment (CI/CD) pipelines, transcends its role as an orchestrator to serve as the operational foundation for cloud-native applications [5] [6].

However, the adoption of Kubernetes is not without challenges. Industry reports and anecdotal evidence indicate a contradiction between the potential for simplifica-

tion and the actuality of operational complexity. Although practitioners commend Kubernetes for its scalability and portability, they also highlight significant learning curves, configuration issues, and challenges related to storage, networking, and cluster update. Certain businesses encounter difficulties in continuously implementing advanced features such as role-based access control (RBAC), custom operators, or scheduling policies, despite their frequent designation as best practices. The outcome presents a complex scenario: Kubernetes is unquestionably crucial to contemporary engineering; but, its application and the issues it poses differ markedly among teams, industries, and maturity levels [7].

Current academic and industrial research addresses aspects of this scenario but neglects significant gaps. Previous studies have investigated the advantages of microservices and containerization, organizational transformations prompted by DevOps, and the efficacy of orchestration frameworks in controlled tests. Although useful, these works frequently depict either broad trends or technical benchmarks in isolation. They overlook the practical realities of Kubernetes adoption and maintenance: the rationale for adoption choices, the daily practices utilized by teams, the indicators present in open-source ecosystems, and the ongoing challenges that endure despite the technology's advancement.

This research seeks to address that deficiency by employing a multi-method empirical strategy. We integrate three synergistic sources of evidence. Initially, we performed 22 semi-structured interviews with professionals in positions like site reliability engineers, DevOps engineers, and solution architects, documenting comprehensive accounts of adoption and operational experiences. Secondly, we gathered 31 survey responses to enhance our understanding and pinpoint common practices and issues among firms. We investigated 1,429 public GitHub repositories containing Kubernetes artifacts, employing a specialized scanner to identify the existence of objects, Infrastructure-as-Code, observability tools, and scheduling capabilities. Collectively, these elements enable us to transcend a singular viewpoint and triangulate among perceptions, self-reported actions, and repository-level evidence.

1.2 Aims and Objectives

The aim of this study is to provide a comprehensive understanding of how Kubernetes is adopted and practiced in industry, and what challenges organizations encounter in doing so. To achieve this aim, the study sets out three key objectives. First, we examine the motivations behind Kubernetes adoption, including the factors that organizations consider, the alternatives they evaluate, and the rationale guiding their decisions. Second, we explore the engineering practices and tools that are most commonly associated with Kubernetes usage. This includes Infrastructure as Code (IaC) approaches, package managers such as Helm, automation through operators, and the integration of observability stacks. Third, we investigate the obstacles that arise in real-world Kubernetes operations. These include technical and organizational challenges such as handling upgrades, managing persistent storage, ensuring compliance, and dealing with configuration complexity.
By addressing these objectives, the study draws on multiple perspectives - interview

accounts, survey responses, and evidence from 1,429 public repositories to highlight where practices align across sources and where they diverge. In doing so, it distinguishes between aspirational best practices often described by practitioners and the usage patterns that are more commonly reflected in open-source projects.

1.3 Research Questions

The rest of the paper is structured around the following research questions:

RQ1 (Adoption): Why do organizations adopt Kubernetes and what attributes do they consider?

RQ2 (Practices): Which engineering practices/tools are the most common?

RQ3 (Maintenance signals): What does open-source activity reveal about Kubernetes-related projects?

RQ4 (Challenges): What are the key operational and organizational challenges in adopting/maintaining Kubernetes?

To address these questions, we follow a mixed-methods empirical approach. We conducted 22 semi-structured interviews with practitioners, collected 31 survey responses, and analyzed 1,429 Kubernetes-related public repositories from GitHub. The interviews shaped the survey design, and the survey results guided our repository analysis, enabling us to triangulate human perspectives with technical evidence.

1.4 Thesis Contribution

We list our contributions as follows:

- A multi-perspective study of Kubernetes adoption and practices, combining interviews, surveys, and large-scale repository analysis.
- A mapping of common, less common, and unevenly distributed Kubernetes practices across practitioner reports and repository traces.
- An examination of maintenance signals through repository data, highlighting the frequency of workloads, components, IaC patterns, and scheduling features.
- A synthesis of challenges faced by practitioners in adopting and maintaining Kubernetes, contextualized with repository evidence.
- A methodological demonstration of how triangulating qualitative, survey, and repository data provides a more comprehensive understanding of cloud-native engineering practices.

1.5 Thesis Structure

We organize the rest of this thesis as follows: in chapter 2 we describe the necessary background and in chapter 3 er prepresent the related work. Chapter 4 details our methodology. Chapter 5 presents our findings, while chapter 6 discusses their implications and outlines limitations. Finally, chapter 7 concludes the thesis and suggests directions for future work.

Chapter 2

Background

In this chapter, we provide an overview of the background necessary to understand Kubernetes and its ecosystem. The discussion begins with the core architecture of Kubernetes, explaining how its main components interact to manage containerized applications. We then explore the fundamental objects that define workloads and resources within a cluster, followed by the broader tooling ecosystem that enhances automation, deployment, and monitoring. Together, these sections build the foundation for understanding how Kubernetes operates in practice.

2.1 Kubernetes Architecture

Kubernetes is an open-source technology engineered to reduce the complexities of managing containerized applications. Kubernetes implements a systematic framework that automates the initiation, cessation, and scaling of containers, eliminating the need for manual intervention [1]. Fundamentally, it comprises two primary components: the control plane, responsible for decision-making and sustaining the system's "brain," and the worker nodes, which are the machines executing programs within lightweight execution units known as pods [8].

The initial diagram (Fig. 2.1) depicts this architecture. Developers and operators typically engage with Kubernetes using the kubectl command-line interface or the Kubernetes Dashboard, transmitting their commands to the API server within the control plane. Subsequently, several components assume responsibility for executing those instructions. For instance, etcd functions as the cluster's memory, preserving its whole state. The scheduler selects the optimal worker node to execute a new pod, while the controller manager continuously monitors the real state to ensure it aligns with the user's specifications. Kubernetes fulfills its commitment to "desired state management" by automatically detecting discrepancies when a pod fails or a node becomes inoperative, then restoring equilibrium within the system [9].

The second diagram (Fig. 2.2) illustrates the process by which a request is executed on the worker side. Upon submission of a deployment by a developer, the API server processes the request, the scheduler allocates a node, and the kubelet, the agent operating on each worker, guarantees the proper initiation of the pod. The kube-proxy manages networking, ensuring that pods may communicate with one another and with other entities. A seemingly straightforward request, "run this

application three times," initiates a series of synchronized tasks across the control plane and the worker nodes [10].

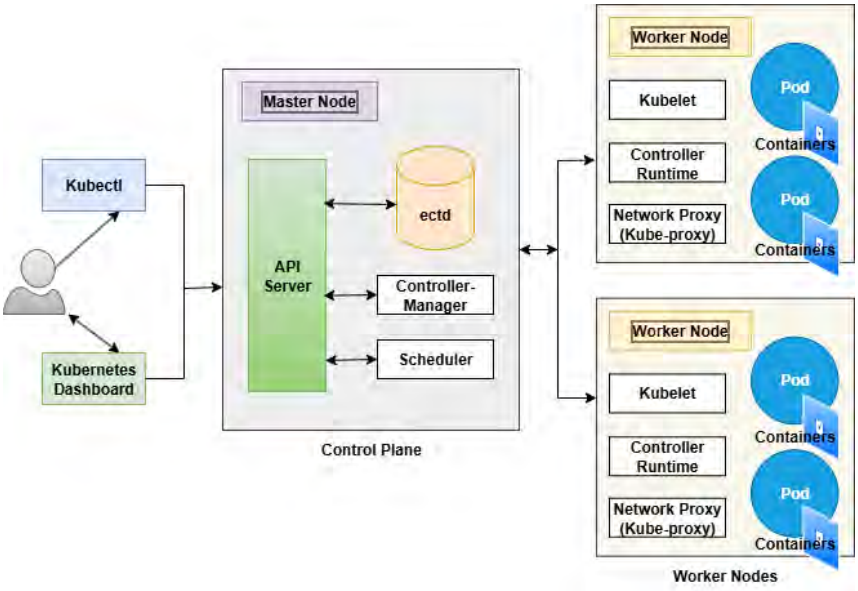


Figure 2.1: Kubernetes Architecture: Control Plane and Worker Nodes—Brief Overview of Kubernetes.

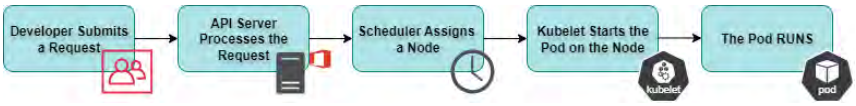


Figure 2.2: Pod Deployment Workflow in Kubernetes.

Collectively, these elements constitute a self-repairing and adaptive system. As demand increases, Kubernetes automatically scales applications upward; if a container malfunctions, it is restarted autonomously. This orchestration paradigm renders Kubernetes appealing to contemporary software teams.

Understanding this history is crucial for this study, as it enables a deeper examination of the Kubernetes objects and components that practitioners depend on in practical environments. The system provides numerous components, including deployments, services, ingress controllers, Helm charts, and operators, which various teams utilize and integrate differently according to their own contexts. Our investigation concentrates on both the technical capabilities of Kubernetes and the practical applications by industry professionals: the features they adopt, those they eschew, and how these decisions influence daily software operations. By integrating architectural frameworks with practitioner observations, we seek to obtain an empirical representation of Kubernetes utilization within the industry.

2.2 Core Kubernetes Objects

To fully understand how Kubernetes functions, it is important to look at the building blocks that make up the system. These building blocks, known as Kubernetes

objects, define the desired state of applications and workloads within a cluster. Each object serves a distinct purpose, from running applications to ensuring availability, scalability, and persistence of data. By combining these objects, developers and operators can design reliable, flexible, and fault-tolerant systems. The following subsections highlight some of the most essential Kubernetes objects and their roles in cluster management.

- **Pod** – The smallest deployable unit in Kubernetes, representing one or more containers that share networking and storage [11]. For example, a Pod might bundle a web server container alongside a logging sidecar.

- **Deployment** – A higher-level controller that manages Pods and ReplicaSets, enabling rolling updates and rollbacks [12]. For instance, a Deployment can ensure that three replicas of a web app are always available during an upgrade.

- **ReplicaSet** – Ensures that a specified number of identical Pods are running [13]. A ReplicaSet might maintain five backend service instances for load balancing.

- **StatefulSet** – Used for stateful applications, giving Pods stable identities and persistent storage [14]. A typical example is running a Cassandra or MongoDB cluster.

- **DaemonSet** – Guarantees that a copy of a Pod runs on each node in the cluster [15]. For example, a DaemonSet is commonly used to deploy monitoring agents such as node-exporter.

- **CronJob** – Schedules jobs to run at fixed times or intervals [16]. A CronJob might trigger a batch ETL process every night at midnight.

- **Service** – Provides a stable endpoint to access a group of Pods [17]. For instance, a Service can expose three replicas of a backend API behind a single cluster IP.

- **Ingress** – Manages external access to Services, usually HTTP/HTTPS [18]. For example, an Ingress rule can route traffic from api.example.com to a backend service.

- **Namespace** – Logical partitions that organize workloads within a cluster [19]. A team might deploy all their microservices into a “dev” namespace separate from “prod.”

- **HorizontalPodAutoscaler (HPA)** – Dynamically scales Pods based on resource usage [20]. For instance, an HPA can increase replicas of a web app from 2 to 10 during high CPU load.

- **PersistentVolume (PV)** – A cluster-wide storage resource provisioned independently of Pods [21]. For example, a PV could be backed by an AWS EBS volume.

- **PersistentVolumeClaim (PVC)** – A user request that binds to a PV [22]. A developer might define a PVC for 10Gi of storage, which Kubernetes maps to a suitable PV.

- **ConfigMap** – Stores non-sensitive configuration data separately from container images [23]. For instance, a ConfigMap may hold environment variables for a service.

- **Secret** – Stores sensitive information like passwords, API keys, or TLS certificates [24]. A Secret might provide a database password to an application Pod securely.

2.3 Tooling Ecosystem

While Kubernetes provides the foundation for container orchestration, its full potential is realized through the surrounding ecosystem of tools. These tools extend Kubernetes by simplifying application deployment, managing infrastructure, automating workflows, and improving monitoring. Together, they form an essential toolkit that enables teams to operate clusters efficiently and reliably at scale. By integrating these tools, organizations can customize their Kubernetes environments to match specific operational needs, from provisioning infrastructure to continuous delivery and observability.

- **Helm** – A package manager for Kubernetes that installs applications as "charts" [25]. For example, installing MySQL in a cluster often involves helm install mysql bitnami/mysql.

- **Operators** – Custom controllers that codify operational tasks [26]. For example, a PostgreSQL Operator can handle automated backups and failovers.

- **Terraform** – A declarative IaC tool for provisioning infrastructure and Kubernetes resources [27]. For instance, Terraform can define an EKS cluster and its associated node groups.

- **Ansible** – An automation tool for system provisioning and configuration [27]. Teams often use it to configure Kubernetes clusters across multiple VMs.

- **Argo CD** – A GitOps-based delivery tool for Kubernetes [28]. It continuously syncs the cluster state with manifests stored in Git.

- **Prometheus** – A monitoring toolkit widely paired with Kubernetes [29]. For example, Prometheus scrapes metrics from Pods and triggers alerts on high CPU usage.

- **Grafana** – A visualization and analytics platform [30]. It can display Prometheus data in dashboards showing cluster health.

- **GKE, EKS, AKS** – Managed Kubernetes services from Google, Amazon, and Microsoft [31] [32] [33]. Teams often adopt these for reducing operational overhead.

Chapter 3

Related Work

Research on Kubernetes and related technologies builds on broader studies in cloud computing, DevOps, and Infrastructure-as-Code (IaC). Prior work spans surveys, interviews, repository mining, and focused analyses of Kubernetes' security, performance, scheduling, observability, and maintainability. We organize the literature as follows.

3.1 Survey-Based Studies on DevOps, IaC, and Kubernetes

Initial research on DevOps implementation predominantly utilized surveys to gather practitioners' perspectives. Lenarduzzi et al. (2020) executed extensive surveys to elucidate the drivers, advantages, and obstacles of DevOps, indicating that enterprises frequently have difficulties with cultural transformations, automation pipelines, and quality assurance [34]. Rahman et al. (2019) investigated IaC adoption using surveys, revealing that while IaC provides reproducibility and scalability, it also presents issues including technological debt, script maintainability, and the absence of recognized best practices [35].
In Kubernetes contexts, polls regularly identify scalability and automation as primary adoption drivers, while highlighting steep learning curves, cluster complexity, and security issues as obstacles [36]. These studies underscore the significance of surveys in encompassing a wide range of industry practitioners.

3.2 Interview-Based Qualitative Studies

Surveys offer broad insights, whereas qualitative studies based on interviews enable researchers to grasp organizational intricacies. Wiedemann et al. (2019) conducted interviews with practitioners regarding microservices adoption, uncovering tensions between architectural flexibility and heightened operational demands [36]. Bogner et al. (2021) investigated continuous delivery strategies via interviews, showing that effective implementation frequently depends more on teamwork, information dissemination, and cultural change than on tools.
In the realm of containerization, qualitative research has highlighted challenges such as frequent version upgrades, troubleshooting distributed systems, and integrating Kubernetes with legacy systems [37]. These studies demonstrate that interviews

reveal human and organizational narratives underlying technology adoption, which surveys alone cannot disclose.

3.3 Repository Mining and Empirical Analyses

A significant body of work employs repository mining to uncover practices, misconfigurations, and maintenance signals. Rahman et al. (2019) analyzed numerous IaC repositories to discern trends and anti-patterns, illustrating the evolution of real-world scripts over time [35]. Jiang and Adams (2015) examined configuration scripts to monitor usage trends and development practices, illustrating how repository data uncovers insights not accessible through surveys or interviews [38].
Recent research has focused on Kubernetes artifacts, including Helm charts, Operators, and YAML manifests, to identify usage trends, persistent misconfigurations, and the adoption of scheduling features [39]. Shamim et al. (2020), in a grey literature study, confirmed that repository analysis can validate community-reported practices (e.g., secret management, RBAC policies) in open-source projects [40]. Repository-based evidence is also valuable for analyzing maintenance indicators such as commit frequency, issue resolution, and release cadence, which act as proxies for project health and sustainability [41].

3.4 Containers and Microservices Beyond Kubernetes

Beyond Kubernetes-specific work, containerization and microservices studies provide important context. Koskinen et al. (2019) reviewed container-based research, advocating for in-depth examinations of performance, monitoring, and isolation [42]. Casalicchio and Iannucci (2020) conducted a systematic evaluation of container research, identifying challenges in performance prediction and multi-layer monitoring [43]. Tyresson (2020) examined Docker security, emphasizing static and dynamic analysis for container fortification [44].
Li et al. (2020) systematically studied microservice designs, highlighting scalability and maintainability as crucial yet inadequately addressed [45]. These works establish the basis for examining how Kubernetes, as the predominant orchestrator, inherits and amplifies issues from prior containerization models.

3.5 Adoption Drivers and Organizational Challenges

Research indicates that organizations implement Kubernetes to gain scalability, portability, and automation. Survey research on DevOps adoption (Erich et al., 2017) indicates that practitioners desire expedited release cycles and adaptable infrastructure [46]. Comparable results arise from studies on cloud migration, where agility and cost-effectiveness are frequently identified as drivers (Jamshidi et al., 2016).
However, adoption is not always straightforward. Interview-based studies highlight organizational inertia, insufficient skills, and challenges in integrating Kubernetes

with legacy systems [47]. Alternatives like Docker Swarm, Mesos, or managed cloud orchestrators have been evaluated, but Kubernetes' expanding ecosystem has positioned it as the preeminent option [43].

3.6 Practices, IaC, and Tooling

Kubernetes adoption brings new processes and tools. Research on Infrastructure-as-Code (IaC) indicates that while teams automate deployments with scripts and frameworks, they face challenges in maintainability and technical debt. Grey literature reviews emphasize that IaC, while powerful, is prone to errors without standardization (Kumara et al., 2021). Helm charts, Operators, and CI/CD pipelines are central to practice but underexplored academically [48].
Interview studies (Soldani et al., 2018) also highlight that microservice adoption is closely tied to Kubernetes usage, with observability and monitoring tools (e.g., Prometheus, Grafana) playing indispensable roles [49].

3.7 Maintenance and Sustainability

Empirical research increasingly examines repositories to monitor maintenance indicators. Analyses of open-source projects show how commit activity, issue resolution, and release frequency reflect project vitality [50].
In Kubernetes, studies of Helm charts and Operators reveal inconsistent maintenance, with many projects stagnating soon after release [51]. Research on IaC repositories highlights common misconfigurations and vulnerabilities, underscoring the need for ongoing maintenance [52]. These findings align with our own repository research, which measures both the presence of Kubernetes elements and the vitality of projects.

3.8 Kubernetes-Specific Research: Security, Performance, Scheduling, and Observability

Kubernetes research spans multiple domains:

- **Security.** Rahman et al. (2023) showed insecure defaults (e.g., privileged containers, weak RBAC policies) in Kubernetes manifests [53]. Zerouali et al. (2023) found outdated and vulnerable images in Helm charts [54]. Minna et al. (2024) evaluated LLMs for detecting/remediating misconfigurations, showing both promise and limitations [55]. Zhang et al. (2021) tracked Helm chart evolution, revealing widespread abandonment [56].

- **Performance and Scalability.** Nguyen et al. (2019) benchmarked orchestrators, finding Kubernetes scalable but with higher scheduling overhead than alternatives [57]. Kim et al. (2020) studied auto-scaling, showing limitations under bursty workloads [58]. Rossi et al. (2020) simulated large-scale clusters, noting trade-offs between cluster size and latency [59].

- **Scheduling and Resource Allocation.** Bauer et al. (2019) explored scheduler extensions for energy efficiency and GPU allocation [60]. Bhat et al. (2021) proposed fairness-oriented schedulers [61]. Gruner et al. (2020) evaluated Kubernetes' default scheduler, noting inefficiencies for NUMA workloads [62].

- **Observability and Monitoring.** Al-Doghman et al. (2021) proposed anomaly detection pipelines using Prometheus/Grafana [63]. Boukhechem et al. (2022) examined logging and tracing, noting persistent complexity for practitioners [64].

- **Reliability and Maintainability.** Casale et al. (2021) studied chaos engineering in Kubernetes [65]. Peinl et al. (2019) analyzed enterprise adoption, highlighting operational struggles [66]. Lyu et al. (2022) mined operator repositories, revealing bottlenecks [67].

3.9 Gaps and Motivation

Despite this diversity, the literature remains fragmented. Surveys provide breadth but lack validation, interviews offer depth but limited scope, and repositories provide technical evidence but little organizational context. Few studies integrate these perspectives. Notably, works such as Almeida et al. (2021) and Cito et al. (2020) acknowledge Kubernetes' socio-technical complexity but isolate single perspectives [68], [69]. Kratzke (2018) highlights Kubernetes' role in shaping the future of cloud-native platforms but does not systematically connect adoption, practices, and ecosystem health [70].

This gap motivates our study. By triangulating surveys, interviews, and repository mining, our work extends prior research to provide a more comprehensive account of Kubernetes adoption, practices, challenges, and sustainability.

Chapter 4

Methodology

The study employs a multi-method empirical framework to investigate Kubernetes adoption and usage trends within the industry. This study employs qualitative interviews, a quantitative survey, and open-source repository mining to completely address the research questions (RQ1–RQ4). Each phase builds on the preceding one, facilitating the triangulation of findings from practitioner views and ecosystem-level evidence. Figure 4.1 presents a summary.

To obtain a comprehensive perspective, we utilized three distinct yet complementary data sources. Initially, we conducted semi-structured interviews with industry professionals to comprehend how firms determine the adoption of Kubernetes, the kind of workloads they execute, and the operational approaches they utilize. Secondly, we developed and distributed an online survey to quantitatively assess these trends and to determine the prevalence of specific adoption reasons and deployment tactics among teams. We performed a comprehensive examination of public GitHub repositories pertaining to Kubernetes, enabling us to identify trends in tooling preferences, project engagement, and the general maturity of the Kubernetes open-source ecosystem.

The interviews provided comprehensive, firsthand insights into the reasons enterprises select Kubernetes, emphasizing motives such as scalability, portability, and automation, along with the architectural choices they implement. The survey facilitated the assessment of the prevalence of various practices, documenting the frequency of different deployment models, workload types, and the integration of Kubernetes with IaC.

The repository study provides an additional perspective by demonstrating how the open-source ecosystem has evolved in response to these tendencies. Through the analysis of metadata from numerous public projects, we discerned the tools that are most vigorously created, the predominant programming languages, and the maintenance patterns of projects over time. These indications illuminate the larger community's response to industry adoption.

The three phases collectively form a more comprehensive representation. The report integrates qualitative views from practitioners with quantitative data from surveys and extensive repository analysis, encapsulating both individual organizational ex-

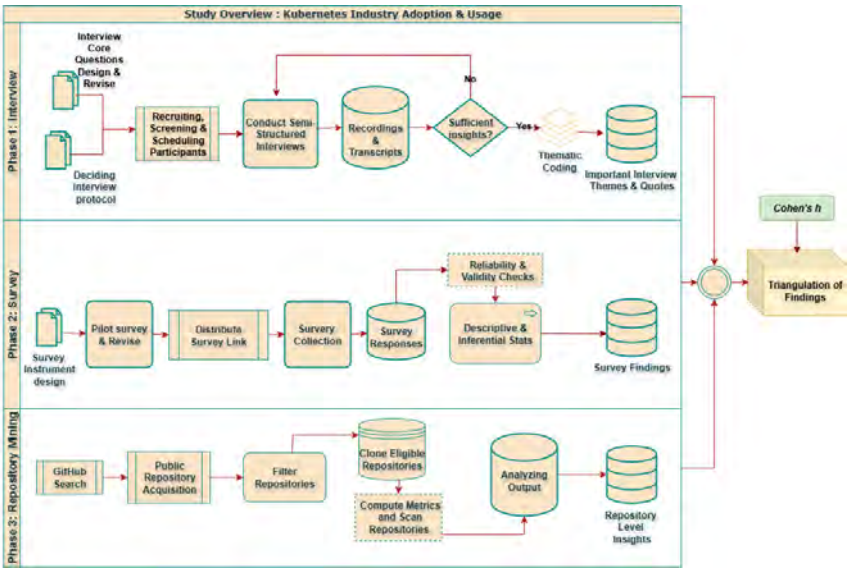


Figure 4.1: Multi-Methods Pipeline & Triangulation of the study for Investigating Kubernetes Adoption and Usage Patterns

periences and the overarching development of the Kubernetes ecosystem.

4.1 Phase 1: Semi-Structured Interviews (Qualitative Method)

In the initial phase of the study, we executed 22 semi-structured interviews with software industry professionals with a minimum of one year of Kubernetes expertise. Among these, 12 participants are recruited from Bangladesh, while 10 were international practitioners located in the United States, United Kingdom, Japan, Australia, India, and Pakistan. This combination enabled us to encompass both regional adoption contexts, such as resource-limited settings in Bangladesh and South Asia, and global viewpoints influenced by established cloud-native ecosystems.
All interviews were done anonymously, safeguarding the confidentiality of personal and corporate identities. Participants were presented with a voluntary honorarium: 1,000 BDT through bKash for Bangladeshi interviews and a $10 Amazon Gift Card for international participants. Figure 4.2 delineates the sequential operations of this phase.

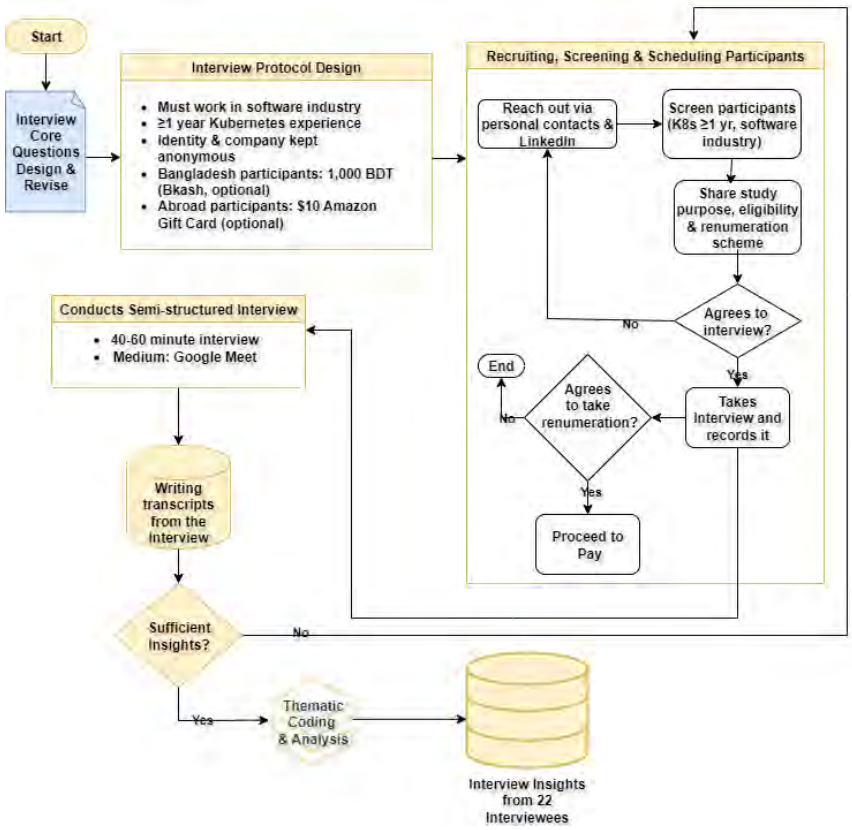


Figure 4.2: Qualitative Method: Interviews (Design → Collection → Analysis)

4.1.1 Interview Protocol Design

The interview protocol was designed to balance consistency across participants with the flexibility to explore unique experiences. The semi-structured format allowed us to follow a predefined guide while adapting follow-up questions based on participants' expertise. The guide was structured into five thematic areas aligned with the study's research questions:

1. **Drivers of Adoption** – motivations such as scalability, portability, modernization, and automation.

2. **Workloads and Deployment Patterns** – applications deployed, cluster usage, and operational models.

3. **Operational Complexity** – day-to-day challenges, incident recovery, performance bottlenecks.

4. **Tooling and Automation** – use of Infrastructure as Code (IaC), Helm, Objects, Operators, observability stacks.

5. **Security and Governance** – compliance practices, access controls, and organizational governance strategies.

Prior to the interviews, participants were briefed on the purpose of the study, the

intended use of the data, and their rights regarding participation. Informed consent was obtained for both participation and recording. All interviews were audio-recorded, subsequently transcribed verbatim, and anonymized to protect participant identity. The Bangladeshi interviewees received a 1,000 Taka through Bkash for participation and the foreign Interviewees received a $10 Amazon gift card.

4.1.2 Participant Recruitment, Screening and Scheduling

The study involved 22 industry professionals via semi-structured interviews, all of whom had direct, practical experience with the deployment, operation, and maintenance of Kubernetes in actual production settings. The recruitment employed a deliberate sampling technique to guarantee variety in geography, organizational size, and job. The recruitment process was intentionally designed to encompass a wide array of perspectives, guaranteeing that the dataset represented both the diversity of adoption scenarios and the profundity of practitioner knowledge.

1. Geographical Distribution

We interviewed 22 professionals with direct experience in Kubernetes production environments. Twelve participants were from Bangladesh, while ten represented international regions including the United States, Australia, Japan, India, Pakistan, and the United Kingdom. This global spread ensured coverage of both developed cloud markets and resource-constrained contexts.

Table 4.1: Geographical Distribution of Interview Participants

Location	**N**
Bangladesh	12
United States	3
Australia	2
Japan	1
India	1
Pakistan	1
United Kingdom	2
Total (Abroad)	**10**
Total	**22**

2. Organizational Scale and Cloud Platforms

The participants came from organizations of varying sizes, ranging from startups operating a handful of nodes to large enterprises managing multi-region, multi-cluster infrastructures. Their environments included AWS, GCP, Azure, hybrid, and on-premises deployments, offering a range of insights on vendor-specific challenges, cost trade-offs, and multi-cloud strategies. Kubernetes experience levels spanned from under two years to over seven years, capturing both early adopter lessons and perspectives shaped by today's mature ecosystem.

3. Recruitment Channels

Participants were recruited through LinkedIn searches and personal contacts reached via emails and LinkedIn direct messages. These approaches ensured the inclusion of practitioners with substantial Kubernetes experience, often yielding senior-level professionals with proven operational expertise.

4. Professional Roles Represented
The participants represented a wide spectrum of technical and managerial positions across diverse industries. The cohort included DevOps-focused professionals such as Senior DevOps Engineers, DevOps Leads, Platform Engineers, and Kubernetes Administrators who oversaw infrastructure automation, reliability, and day-to-day cluster operations. Site Reliability Engineers (SREs), Platform Reliability Engineers, and Infrastructure Engineers contributed perspectives on maintaining large-scale, production-grade environments.

Several participants worked in software development roles, including Senior Software Engineers, Back-end Developers, and Systems Engineers, focusing on application-level deployment, service integration, and scalability. In addition, the sample included Software and Solutions Architects as well as a Cloud Infrastructure Lead, reflecting strategic roles in system design and cloud-native adoption. Senior Engineers from AI/ML startups, research organizations, and large enterprises added insights into sector-specific Kubernetes practices.

This diversity of roles ensured coverage of both hands-on technical expertise and higher-level architectural and strategic decision-making in Kubernetes adoption and operation.

4.1.3 Conducting Semi-Structured Interviews

The interviews were conducted in a semi-structured manner, following a framework that concentrated on five domains: adoption drivers, workload and deployment patterns, operational problems, tooling and automation methods, and governance/security considerations. This framework facilitated comparability while allowing participants to elaborate on their unique experiences.
Before commencing the full set of interviews, a pilot interview was conducted to test the clarity of questions and refine the flow of the discussion guide. Feedback from this session informed minor adjustments, ensuring the interviews were both structured and flexible.
Each interview spanned 40 to 60 minutes and was conducted remotely over Google Meet. Participants were free to communicate in either English or Bengali, depending on their comfort level. All sessions were audio-recorded with prior consent to ensure accuracy.
The participant group exhibited diverse deployment contexts:

- A South Asian ride-hailing platform managing over 70 million daily requests, utilizing advanced autoscaling and high-availability techniques.

- A regional retail organization operating AWS-based Kubernetes clusters in over 10 Southeast Asian countries, with a strong emphasis on compliance and workload distribution.

- Several cloud-native firms leveraging managed Kubernetes services to achieve rapid scalability while minimizing operational overhead.

This design yielded a comprehensive dataset that combined technical detail with strategic management perspectives. The carefully selected participant base, together with the pilot interview, provided a robust empirical foundation for identifying patterns in Kubernetes adoption and usage across the global industry landscape.

4.1.4 Thematic Analysis of Interview Data

All interviews were recorded fully and anonymized to safeguard participants' confidentiality. We subsequently employed thematic analysis (Braun Clarke, 2006), providing a systematic yet adaptable method for recognizing recurring patterns within the data [71].

1. Familiarization - Each transcript was examined repeatedly to cultivate a profound comprehension of both the explicit content and the implicit context. At this time, we saw persistent issues like the significant learning curve associated with Kubernetes, prevalent tooling techniques, and daily operational difficulties.

2. Preliminary Coding — Pertinent segments of text were subsequently annotated with descriptive labels. These codes encompassed both technological and organizational dimensions. For instance:

- **Learning and skills** – Difficulties in explaining containers and the shortage of skilled Kubernetes administrators.

- **Cluster operations** — Challenges with recurrent version upgrades, CrashLoopBackOff faults, node malfunctions, and ingress-related complications.

- **Toolchains and automation** — Dependence on Helm, Terraform, Ansible, and Prometheus, coupled with deficiencies in GitOps methodologies.

- **Data and state management** — Challenges with StatefulSets, persistent volumes, and the removal of PVCs.

- **Security protocols** — Implementation of Role-Based Access Control (RBAC), management of confidential information, multi-layered security, and vulnerabilities arising from misconfigurations.

- **Adoption Methods -** Encompass options about managed versus self-managed clusters, compliance-oriented choices in FinTechs, and trade-offs associated with multi-cloud environments.

- **Community and ecosystem** – the role of local meetups and the growing Kubernetes practitioner base in Bangladesh.

3. Theme Development — The interview findings were coded and then synthesized into themes that closely corresponded to the study's research questions. For RQ1 (adoption drivers), themes reflected concerns around scalability, adaptability, and the range of workloads organizations aimed to support with Kubernetes. In relation to RQ2 (engineering practices), recurring codes focused on the use of different Kubernetes objects, cluster components, and associated toolchains, underscoring how teams structure and extend their environments. While RQ3 (maintenance signals) was primarily explored through repository mining, interview narratives about upgrade pain points and sustainability issues offered valuable context. Finally, RQ4 (operational challenges) emerged most strongly from the interviews, where practitioners highlighted struggles with configuration complexity, persistent storage, cluster upgrades, security and compliance, and the steep learning curve.

Table 4.2: Theme Development: Areas and Inclusion Criteria

Theme Area	**Inclusion Criteria**
Objects	References to Pods, Deployments, Services, Ingress, ConfigMaps, Secrets, CRDs, etc.
Components	Mentions of Helm, Terraform, ArgoCD, Flux, Ansible, Prometheus, Grafana, Datadog, etc.
Workloads	Discussions about web apps, databases, CI/CD pipelines, batch jobs, or ML workloads.
IaC (Infrastructure-as-Code)	Statements on automation, GitOps pipelines, IaC practices, or configuration management.
Logging/Monitoring	Mentions of observability tools, monitoring dashboards, log collection, or alerting practices.
Scheduling Features	References to HPA/VPA, affinity, tolerations, taints, or custom scheduling policies.
Challenges	Any mention of steep learning curve, operational overhead, security issues, or organizational barriers.

4. Cross-Case Comparison — Subsequently, we analyzed themes across various participant groups. In Bangladesh, practitioners frequently emphasized the deficiency of experienced professionals and the community's involvement in addressing this shortfall, while overseas participants underscored the complexities of upgrades and governance issues. Startups generally characterized lean implementations reliant on managed services, whereas larger corporations addressed intricate multi-cluster, multi-region deployments with a primary focus on compliance.

The final outcome was a collection of well articulated and experimentally substantiated themes. These topics not only influenced the design of the survey used in Phase 2, but also provided a qualitative lens for interpreting insights from the repository mining in Phase 3.

4.2 Phase 2: Online Survey (Mixed Method)

The online survey complemented the interviews by extending our insights to a broader population of practitioners. While the interviews provided rich, contextual narratives, the survey enabled us to quantify the prevalence of adoption drivers, workload patterns, tool selections, and operational challenges across a more diverse set of participants. Importantly, all survey respondents were distinct from interviewees, ensuring independent perspectives and eliminating redundancy.

4.2.1 Survey Design

The survey was developed to capture both measurable patterns and practitioner perspectives, while keeping the design grounded in the experiences reported during Phase 1 interviews. Most of the questions were structured, asking participants about Kubernetes adoption drivers, the types of workloads they run, the objects and components they use, and the infrastructure that supports their clusters. A small number of open-ended questions gave participants the opportunity to elaborate on challenges or describe practices in their own words.

The questionnaire covered organizational background, motivations for adopting Kubernetes, usage characteristics, and operational difficulties. By drawing directly on interview findings such as issues with persistent storage, Helm security concerns, or the complexity of ingress—the survey ensured that recurring themes were systematically tested with a wider group of practitioners.

4.2.2 Distribution and Responses

Before launching the main study, a pilot survey was conducted with one practitioner to test clarity, timing, and flow. Feedback from this pilot was used to refine wording and improve the overall instrument.
The final survey was then shared through targeted outreach on LinkedIn, focusing on practitioners with direct Kubernetes experience across different industries and regions. Each invitation explained the purpose of the study, the expected time commitment, and emphasized anonymity and confidentiality. To preserve independence between datasets, none of the interview participants were invited to take part in the survey.

In total, 31 complete responses were collected. Respondents represented a mix of organizational contexts, ranging from startups with lightweight deployments to large enterprises operating Kubernetes at scale, often across multiple clusters and geographies. This diversity provided a balanced view of how Kubernetes is used in practice.

4.2.3 Survey Data Analysis

Responses were analyzed using a mixed-method approach. The structured questions were summarized with descriptive statistics to reveal adoption trends, commonly used components, and preferred infrastructure setups. The limited open-ended input was coded thematically, using the same framework as the interviews, to ensure consistency while allowing participants' own words to enrich the findings.

The survey confirmed several issues raised in interviews, such as the challenges of managing stateful workloads and the burden of frequent version upgrades. It also quantified the widespread reliance on tools like Helm, Terraform, and Prometheus. Together with interviews and repository analysis, the survey added breadth to the study and strengthened the overall validity of the results.

4.3 Phase 3: Repository Mining (Computational Method)

The final part of the study included a comprehensive investigation of Kubernetes-related repositories on GitHub. The prior two phases documented the perspectives and experiences of practitioners via interviews and surveys, but this stage provided an alternative viewpoint by enabling us to examine ecosystem-level activities embedded within open-source projects. Through the analysis of source code, configuration files, and repository activity, we gained insight into the practical application of Kubernetes, offering a significant contrast to practitioner perceptions. Figure 4.3 presents a comprehensive account of the repository analysis.

4.3.1 Repository Acquisition

Repositories were obtained through a combination of an advanced browser-based query and the GitHub REST API. The browser query was designed as an initial reference to represent authentic, production-grade Kubernetes utilization. A total of 17,369 repositories were obtained via the GitHub Search API. Due to the API's limitation of 1,000 items per query, we employed a star-range paging approach to ensure comprehensive dataset coverage. This method was essential to guarantee the inclusion of both prominent flagship projects and smaller, community-driven repositories.

BROWSER QUERY:
"topic:kubernetes archived:false mirror:false created:<2025-07-01pushed:>2015-07-01"

This query guaranteed the inclusion of only repositories explicitly labeled with the Kubernetes topic, while omitting mirrors and archived projects that are no longer under active development. The specified date range encompasses projects initiated from 2015 to 2025, corresponding to the timeframe when Kubernetes became suitable for industrial production deployment, subsequent to its initial release in 2014. Repositories were required to possess a minimum of one star to exclude trivial or

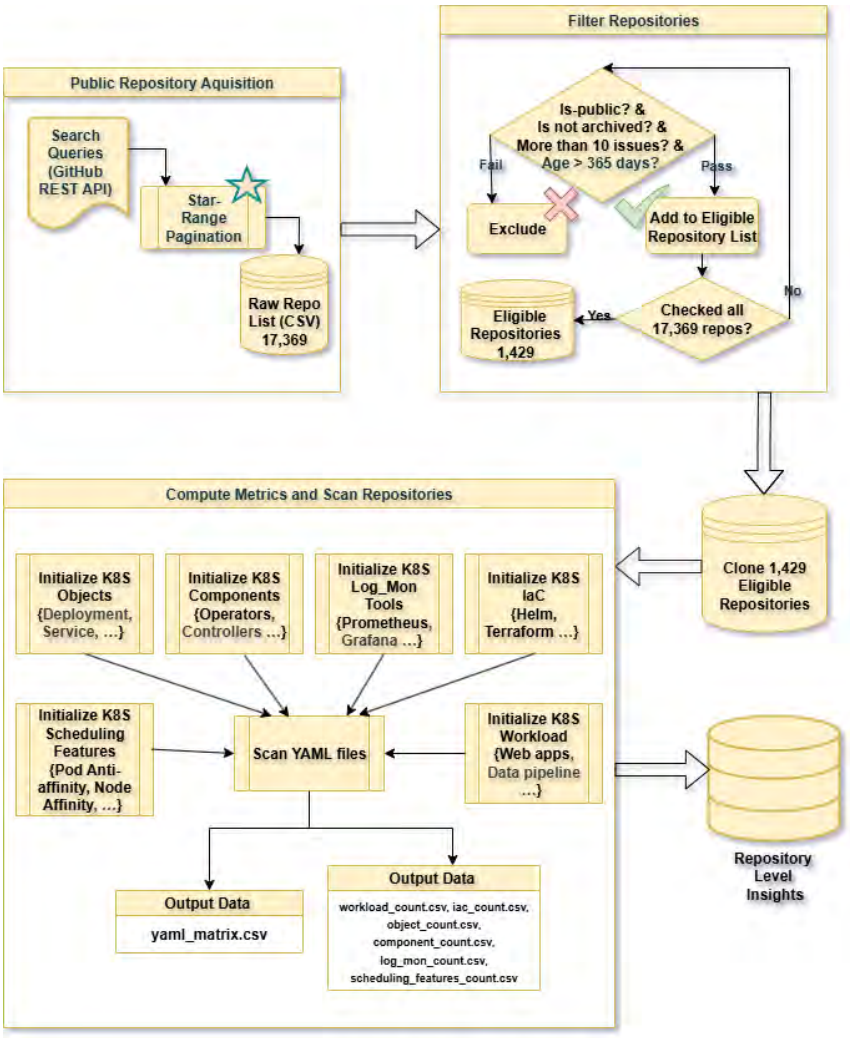


Figure 4.3: Repository Mining (Discovery → Filtering → Analysis)

vacant codebases. The initial search yielded a raw dataset including 17,369 repositories.

To address GitHub's API constraint of providing a maximum of 1,000 repositories per query, we enhanced the browser search by implementing star-range pagination via the REST API (e.g., requests for repositories with stars: 1..50, 51..200, 201..1000). This guaranteed the inclusion of projects across the spectrum of popularity, ranging from minor community tools to prominent Kubernetes operators. Metadata including repository name, owner, URL, star and fork counts, creation date, last push date, license, topics, and primary language was compiled into a unified CSV file.

The procedure encompassed multiple technical stages:

1. Repository Segmentation by Popularity Repositories were divided into star-count buckets (e.g., 20,000–50,000; 10,000–19,999; 5,000–9,999; . . . ; 1–4). This ensured that each query stayed within the API's 1,000-item limit while covering the entire spectrum of repository popularity.

2. Fetching in Descending Star Order Within each bucket, repositories were retrieved in descending order of stars using the sort=starsorder=descAPI parameters. This guaranteed that the most prominent repositories—often those with high visibility and active communities—were prioritized in the results.

3. Rate-Limit Management To comply with GitHub's rate limits, a 3.2-second delay was introduced between successive API calls. Additionally, HTTP 403 responses (indicating rate limit exceeded) were automatically handled by pausing the script until the reset time indicated by the X-RateLimit-Reset header. This ensured a complete dataset while maintaining compliance with GitHub's Terms of Service.

4. Automation and Storage The entire collection process was automated using Python scripts, which:

- Queried the API and handled pagination.
- Stored responses in JSON format for reproducibility.
- Logged error messages and reset times for auditability.

This automation minimized human error, ensured consistent retrieval across multiple queries, and enabled replication of the procedure for future studies.

5. Validation of Query Accuracy Prior to full-scale collection, browser-based validation queries were executed directly on GitHub's search interface. This step ensured that the API results aligned with the browser results for equivalent queries, thereby verifying the accuracy and consistency of the search parameters.

4.3.2 Repository Filtering

After collecting the raw dataset, we applied a multi-stage filtering process to focus on repositories that were relevant, active, and reflective of real-world Kubernetes usage. These thresholds were chosen to balance inclusiveness with the need for industry-relevant signals.

1. Public Availability — Only repositories marked as `is:public` were retained (17,369 repos). Public repositories ensure transparency, reproducibility, and open community access, which is essential for empirical research.

2. Not Archived — Archived repositories were excluded (`archived:false`), as these are no longer actively maintained (17,368 repos). This step ensures that the dataset reflects living projects rather than abandoned codebases. By doing so, we capture practices that are relevant to current engineering work.

3. Open Issues > 10 — Repositories with fewer than 10 issues were removed (1,557 repos). This threshold was selected to filter out toy projects, forks, and trivial demos, which typically lack sustained community engagement. A minimum of 10 issues establishes a baseline for active collaboration and maintenance, aligning with prior repository-based studies that used issue activity as a proxy for vitality.

4. Project Duration > 1 Year — Only repositories with at least one year between creation and last update (`created_at` vs. `pushed_at`) were included (1,429 repos final set). This ensures projects have sustained activity over time, enabling meaningful measurement of commit cadence, release frequency, and issue/PR dynamics. Short-lived projects were excluded because they rarely reach production relevance.

Taken together, these filters reduce noise and highlight repositories that are community-engaged, maintained over time, and closer to production-grade Kubernetes usage. The thresholds reflect a balance between inclusiveness and reliability, ensuring that the analysis captures meaningful engineering practices rather than ephemeral or trivial projects.

4.3.3 Repository Cloning

For extensive study, 1,429 repositories were chosen using the previously specified filtering criteria (C1–C4). The repositories were cloned locally utilizing a bespoke bulk cloning script that automated the procedure via GitHub API requests. The script utilized concurrency to swiftly clone repositories, guaranteeing that all project files, especially Kubernetes configuration files in YAML, were accessible for later scanning. This local copy furnished a reproducible and controlled dataset for extracting Kubernetes-related signals.

4.3.4 Compute Matrices and Scan Repositories

Upon cloning the 1,429 qualifying repositories, we designed and executed a custom-built scanner capable of activating multiple categories of Kubernetes signals. These categories span a broad spectrum of Kubernetes usage: core objects such as Deployment and Service; platform components like Operators and Controllers; Infrastructure-as-Code (IaC) tools including Helm and Terraform; observability stacks such as Prometheus and Grafana; typical workloads like web applications or data pipelines; and pod scheduling features such as node affinity and anti-affinity rules.

The scanner systematically traverses every .yaml file within the cloned repositories, parsing them to identify whether these features were present. Beyond simple detection, the scanner was also configured to count the frequency of each element within a repository. For instance, if multiple Deployment objects or Service definitions appeared across different manifests, the scanner recorded not just their presence but also the number of occurrences. This was achieved by incrementing counters for each matched pattern during parsing, thereby producing both binary (present/absent) and quantitative (frequency) evidence of Kubernetes practices.

The outcome of this process was a consolidated dataset named `yaml_matrix.csv`, which records the detection results in tabular form, mapping each repository against both the presence and occurrence counts of Kubernetes signals.

While this scanner was purpose-built for the present study, its design was inspired by earlier efforts in the research community, particularly the parsing and scanning

logic demonstrated in the KubeSec project's open-source artifacts (scanner.py and parser.py) [49][50]. By extending and tailoring these ideas, the scanner developed in this work is capable of parsing at scale across thousands of repositories, generating a dataset that enables both quantitative frequency analysis and triangulation with interviews and surveys.

4.3.5 Repository Level Insights

The `yaml_matrix.csv` file provides a detailed overview of Kubernetes adoption at the repository level. Each row represents a unique Kubernetes element (objects, components, IaC tools, workloads, logging/monitoring tools, scheduling features), whereas each column denotes one of the 1,429 repositories. The matrix encodes two complementary forms of evidence:

1. A binary signal where 1 indicates the element's presence in the repository and 0 signifies its absence.

2. A frequency count representing how many times the element was detected across the .yaml files of that repository.

For example, a repository might contain a single Deployment (binary = 1, frequency = 1), while another repository might define ten different Deployments (binary = 1, frequency = 10). This dual encoding ensures that the matrix not only captures whether Kubernetes constructs are used but also how extensively they are employed.

This enriched representation enables deeper examination of adoption trends, relative emphasis across repositories, and frequency-driven comparisons. As such, the `yaml_matrix.csv` functions as a conduit between unprocessed repository scans and higher-level insights into Kubernetes engineering methodologies, providing both qualitative traceability and quantitative rigor.

In addition to the matrix, a set of aggregated count files, such as `object_count.csv`, `component_count.csv`, `iac_count.csv`, `workloads_count.csv`, `logging_monitoring_count.csv`, `scheduling_features_count.csv`, were generated. These files summarize the binary and frequency signals at the category level, providing a repository-by-repository breakdown of how many distinct objects, components, or tools were present. Together, the matrix and these summary counts form the foundation for comparative analysis, enabling both fine-grained inspection of individual repositories and broad pattern recognition across the entire dataset.

4.4 Triangular Integration of Methods

This study employed a triangular, mixed-method design that combined qualitative, quantitative, and computational approaches to capture a comprehensive picture of Kubernetes adoption. Integration was planned from the outset and systematically applied in both design and analysis.

The process began with semi-structured interviews, which revealed practitioners' motivations for adoption, common workloads, operational pain points, and tooling practices. These insights directly informed the survey instrument, with themes such as PersistentVolumeClaim (PVC) management, Helm chart vulnerabilities, and multi-cluster governance transformed into structured question.

In parallel, a GitHub repository mining strategy was developed to extend practitioner perspectives to the broader open-source ecosystem. This strand produced large-scale evidence of ecosystem activity, including CustomResourceDefinition(CRD) diversity, commit frequencies, issue resolution, and release patterns.

During analysis, the three strands were integrated through triangulation. Interviews provided depth, surveys added breadth, and repositories offered scale. We examined both convergence (e.g., Deployments and Services consistently dominant across all sources) and divergence (e.g., industry emphasis on StatefulSets or RBAC contrasted with their rarity in open-source projects).

For instance, when interviewees emphasized multi-cluster management challenges, survey responses confirmed prevalence, while repository evidence showed whether related tools were gaining or losing momentum. Similarly, declining commit activity in certain repositories was compared with reports of tool substitution in practice.

Taken together, this triangular integration balanced narrative depth, quantitative generalization, and empirical traceability, enabling corroboration where sources aligned and exposing gaps where they diverged. This produced a more nuanced understanding of Kubernetes adoption and ecosystem dynamics than any single method could provide.

4.5 Ethical Considerations

This study was executed in accordance with stringent ethical guidelines to protect participants and guarantee responsible data management throughout all stages. Participants were thoroughly informed about the study's objectives and their rights, with informed consent secured before recording the interviews. All transcripts were anonymised to eliminate personal or organizational identities, and both recordings and transcripts were securely maintained in access-controlled systems. The poll adhered to the same principles: participation was voluntary, and only non-identifiable information, including role, organization size, and region, was gathered. Responses were securely saved and processed in aggregate to maintain anonymity.

During the GitHub mining phase, solely publicly accessible metadata (such as stars, forks, issues, commits, and repository properties) was gathered. No private data, contributor identification, or sensitive information were accessed, and all data collecting adhered completely to GitHub's Terms of Service and API usage standards. Collectively, these methods guaranteed the preservation of privacy, security, and compliance, while ensuring the transparency and reproducibility of the study's results.

Chapter 5

Results and Findings

This section describes the outcomes of our study, organized through three distinct methodologies: semi-structured interviews with 22 practitioners, survey responses from 31 participants, and the examination of 1,429 public Kubernetes repositories. Each method offers a unique perspective: interviews provide practitioner experiences and viewpoints, surveys emphasize wider usage trends and practices, and repository mining gathers tangible technical proof of Kubernetes adoption in open-source projects.

In the last segment of this section, we synthesize various strands of evidence about our five research questions (RQ1–RQ5). This method allows us to transcend individual stories by amalgamating practitioner narratives, self-reported behaviors, and observable repository indicators, thus providing a thorough and empirically substantiated comprehension of Kubernetes uptake, utilization, and maintenance in practice.

5.1 Interview Findings

We conducted a total of twenty-two (22) semi-structured interviews with practitioners offering varied opinions on Kubernetes. The participants encompassed a variety of organizations, including software service providers, fintech startups, large technology corporations, AI-centric firms, digital platforms, and retail groupings around Asia. Their professional positions encompassed solution architects, DevOps and cloud engineers, site reliability engineers, and backend developers, many of whom possessed extensive hands-on expertise with Kubernetes. Collectively, these discussions offer a pragmatic perspective on the adoption, operation, and practical experience of Kubernetes, highlighting the reasons that propel companies towards it and the daily obstacles they face.

5.1.1 Workloads

Across organizations, web and microservice workloads remain the primary drivers of Kubernetes adoption, with respondents consistently emphasizing scalability, elasticity, and high availability. A senior backend developer working across large consumer platforms described the scale imperative succinctly: *"When we deal with such a*

large-scale company we must need Kubernetes... The main need... is for places where we need to deal with microservices; startups do not need Kubernetes... a ride-hailing super-app needs to deal with 70 million requests per day." (I17) Similarly, engineers in software services environments stressed autoscaling benefits: *"One of the reasons we use Kubernetes is that we can scale very quickly... if my resources are being used more than 50%, Kubernetes can automatically create a new pod... I can scale up to 1000 pods without facing issues... scale according to the demand."* (I3) The reliability rationale was especially salient in regulated sectors such as fintech: *"This is a fintech company, so even a millisecond of downtime matters... 100 or 200 transactions could fail... This is why we have adopted the Kubernetes approach. If the main service goes down, Kubernetes will automatically start the replica on another server, ensuring zero downtime."* (I5)

Databases, by contrast, were treated with caution in production contexts, creating a pragmatic boundary around stateful workloads. Participants frequently recommended external managed services for relational databases, citing operational risks and data-loss concerns when hosting stateful data in-cluster: *"Kubernetes is bad for stateful sets—databases... there is a chance of data loss if kept in Kubernetes... we recommend not to host databases in Kubernetes."* (I17) This norm was echoed by software services engineers: *"Mainly the web applications we have are deployed on Kubernetes, but persistent applications, such as databases, we try to avoid using Kubernetes for those."* (I2) At the same time, some organizations deliberately consolidated everything including databases inside Kubernetes, often with operators, reflecting a different risk posture: *"In our organization, everything starting from database, service, application all is managed by the kubernetes cluster... If we need a database we can manage it using a PGO operator from the same cluster."* (I7)

Workloads such as batch jobs and data pipelines appeared more variably across contexts, but in AI-focused companies, pipelines and GPU-intensive tasks were central. A cloud architect described a tight coupling between GPU enablement and higher-level orchestration: *"Because we are an AI-based company most of our services require GPUs, so we make heavy use of NVIDIA's GPU operators to dynamically provision and register new GPUs from different nodes on the clusters. The operator ensures GPUs are consistently available, and the higher-level jobs or pipelines then orchestrate workloads on top."* (I14)

5.1.2 Object Usage

Interviewees consistently treated Deployments and Services as foundational building blocks for delivering and scaling applications. Practitioners listed these among their most common objects alongside ingress and persistence primitives: *"The components we work with the most are deployments, replica sets, ... ingress controllers, services, and persistent volumes."* (I2) ConfigMaps and Secrets were pervasive for configuration and sensitive values, and sometimes explicitly tied to database credentials and environment-specific routing: *"Every service has config env, secrets, etc. These are by default present."* (I3) and *"DB passwords in Secrets... server specific URLs are in the ConfigMap."* (I17; also I16). Namespaces and Ingress were widely used for multi-tenant hygiene and traffic management, though ingress routing still created operational friction: *"Namespace wise separation implementation is possible. Centrally we can manage all... From one single place monitoring and bug*

fixing is also possible." (I7) and *"I faced some technical challenge in ingress... For path specification I could not find the ingress reachability."* (I7) CustomResourceDefinitions (CRDs) surfaced prominently where teams embraced operators, including advanced placement and GPU-related constraints: *"We utilize CRDs... for cases where one pod has two containers but each container needs to be deployed on a separate node to use two different GPUs... that's where CRDs come in."* (I14) Storage primitives (PV/PVC) remained a frequent pain point, particularly around lifecycle and recovery after failures: *"Managing these persistent volumes in the Kubernetes world is still challenging... for applications where data is crucial, like databases, we cannot afford to lose data."* (I13) and *"New pod won't attach if a PVC already has data; we need an operator/controller to find and clean stale PVCs... we extended Prometheus agents and built a custom dashboard to prune safely."* (I17) Autoscaling and disruption controls were acknowledged but not uniformly tuned or emphasized. Some respondents referenced autoscalers and pod-scaler components in passing rather than as systematically governed SLO instruments: *"We use the kubectl system, kube-proxy, controllers, pod scaler, auto scaler."* (I19)

5.1.3 Components Usage

Organizations diverged in their depth of reliance on operators and custom controllers. Several teams minimized operator usage, leaning on Helm-based automation: *"We don't currently use any operators... if we need automation, we mainly use Helm for that."* (I2) In more mature settings, however, controllers were ubiquitous and bespoke: *"There are almost 100–200 controllers... we write controllers to sync ConfigMaps, do A/B routing, improve stateful recovery, handle backups, and namespaced access control (merge-request-gated RBAC)."* (I17) Platform components such as kube-proxy, controller-manager, and CoreDNS were mentioned primarily in operational contexts—things to be configured, observed, or debugged rather than objects of design choice: *"We use the kubectl system, kube-proxy, controllers, pod scaler, auto scaler... we updated the default CNI plugin..."* (I19) and *"CoreDNS or k8s metrics add-ons were used."* (I6)

5.1.4 Infrastructure as Code and Delivery

The delivery toolchain coalesced around Helm for packaging, Git-based workflows for desired state, and Terraform for infrastructure provisioning. Multiple teams emphasized ubiquitous Helm usage: *"All our services implement Helm charts, and these charts are used in combination with Ansible."* (I1) and *"Helm is a package manager... rather than building everything from scratch, we use Helm to manage these packages."* (I3) Some organizations even standardized reusable Helm templates across environments: *"Yes, we use a customized Helm package manager... basically done here using templates... the base environment file, we will use this at different stages, in different contexts, differently."* (I6) Whereas the industry generally celebrates GitOps tools, not all teams adopted them, preferring custom pipelines around Helm and Ansible: *"The industry tends to prefer GitOps tools like Argo and Flux... but neither does it 100% well. We didn't choose them even though 90% of the industry uses them. Instead, we built custom tools with Ansible and Helm."* (I1) That said, many respondents described conventional CI/CD with GitHub Actions

and ArgoCD: *"We handle the CI part with GitHub Actions, and the CD part is done with Argo CD... when a developer pushes to the main branch, it automatically triggers the build... and it's automatically deployed to Kubernetes."* (I3)
On the provisioning side, Terraform was widely cited for cluster and cloud resources, sometimes with policy encapsulated in modules: *"We manage everything through Terraform... developers can create clusters as needed, while maintaining the policies enforced in the hidden Terraform modules." (I1)* and *"For deploying Kubernetes clusters we use Terraform... all the provisioning regarding the cluster and cloud we do through Terraform."* (I17) Several participants also articulated a pragmatic path from local Docker/Compose to production Kubernetes underscoring a pipeline mindset that emphasizes progressive hardening: *"Local: build a Dockerfile, run with docker-compose; for production we deploy into Kubernetes... a very standard way to bring something from local into production."* (I14)

5.1.5 Monitoring and Observability

Prometheus and Grafana functioned as the de facto observability stack across many settings, complemented by Datadog in larger or more instrumented environments. Respondents described standard patterns around metrics collection and customizable dashboards: *"For monitoring, we use Prometheus and Grafana... Prometheus collects all resource allocation data... Grafana presents this data in graphs, charts, bars... We can customize the dashboard."* (I5) and *"We actually use Prometheus and Grafana for all sorts of monitoring... we also started using Datadog; alerts hit Slack."* (I17) Other tooling such as Vector, OpenSearch, and Lens featured in targeted roles: *"For cluster logging and metrics, we use Datadog's cluster agent... for logs, we deploy Vector to collect pod logs and dump them in S3 or GCS buckets before parsing them into OpenSearch."* (I1) and *"We use Lens to monitor the Kubernetes cluster in development."* (I3) The operational reality of alerts, incident triage, and recovery—rather than purely architectural design—was a recurrent theme in these accounts: *"We always have alerts in Datadog... issues will happen, like pods in CrashLoopBackOff or unresponsive nodes, but the key for us is recovering quickly so customers like large global clients aren't impacted."* (I1)

5.1.6 Security and Access Control

Security practices spanned RBAC, namespace isolation, least privilege, image scanning, and policy guardrails, but consistency varied with team maturity and scale. Some organizations implemented tight identity-to-permission mappings: *"We've implemented RBAC mapped 1-to-1 with AWS IAM roles so developers have different levels of access, from viewing logs to executing database migrations."* (I1) and *"We do have RBAC in place... we implement least privilege... some are only allowed to view nodes/pods; others, like myself, have full control."* (I14) Others acknowledged that team size or skill depth limited immediate adoption: *"Not at this point since the team is very small now... there are some restriction options in AWS. We use that."* (I16) Security-by-configuration, while recognized as essential, was also fragile, reinforcing the need for strong defaults and continuous validation: *"In monolithic platform security was highly ensured but in kubernetes security is still not up to the mark [here]... If one service is exposed then the full cluster can be hacked.*

Here... correct security policies must be implemented... namespace slicing should be known." (I7) At scale, teams adopted Pod Security profiles and network-level controls to constrain blast radius: *"Pod Security: privileged / baseline / restricted profiles... in EKS we use security groups so pods can't autoscale/spawn or change cluster roles."* (I17)

5.1.7 Scheduling Features

Placement controls were applied to balance performance, reliability, and heterogeneous node capacities, but they introduced operational overhead. Fintech teams operationalized taints, tolerations, and affinity to right-size workloads: *"We use Taints and Tolerations, and Affinity... to ensure the service runs on the desired node."* (I5) High-availability requirements motivated anti-affinity to improve resilience against single-node failures: *"We use pod anti-affinity... e.g., Elasticsearch across nodes so a single node failure doesn't take it down."* (I17) Even so, respondents acknowledged that per-service placement rules can become burdensome as fleets grow, pointing to a gap between conceptual best practice and day-to-day maintainability: *"When a service needs to run on one node instead of another... as the number of services increases, it becomes overwhelming... set these rules for each... a bit cumbersome."* (I5)

5.1.8 Cross-cutting Adoption Rationale

Finally, the interviews converge on a pragmatic adoption logic: Kubernetes is most compelling when systems reach a threshold of complexity where scaling, orchestration, and availability requirements dominate. Engineers recounted the organizational progression from monoliths to VMs to container orchestration under operational pressure: *"Before I joined, we were using VM-based deployments, but there were issues with VM deployments, such as the inability to scale up or scale out at a granular level. That was one of the primary reasons for migrating to Kubernetes... We need to orchestrate thousands of containers — with 60+ microservices, each having around 10 pods, plus management pods. Managing them manually on VMs became unmanageable, which is another reason for choosing Kubernetes."* (I9) Yet, participants also warned against indiscriminate adoption, advocating for timing and fit: *"Don't jump on the Kubernetes bandwagon just because everyone else is doing it... when I was part of a startup, we ran five services on VMs and never felt the need for Kubernetes."* (I1) and *"If you are dealing with microservices only then I suggest Kubernetes, otherwise it's a waste of resources."* (I19)

5.2 Survey Findings

In addition with the interviews, the survey responses provide insight into practitioners' actual experiences with Kubernetes in their daily operations. The subsequent figures illustrate not just the most frequently utilized tools and items but also the obstacles teams encounter, the challenges they face, and the strategies they employ to manage Kubernetes effectively.

5.2.1 Workloads and Clusters

The overwhelming majority of respondents (93.8%) reported running web applications on Kubernetes, underscoring its dominant role in serving production-facing services.Data pipelines (40.6%) and databases using StatefulSets (34.4%) were also notable, showing that teams are extending Kubernetes beyond stateless workloads. Batch jobs (31.3%) were less common but still represent an important use case for resource scheduling and automation.Together, these results highlight Kubernetes' versatility, with strong adoption in traditional web services while gradually expanding into more stateful and data-intensive domains.

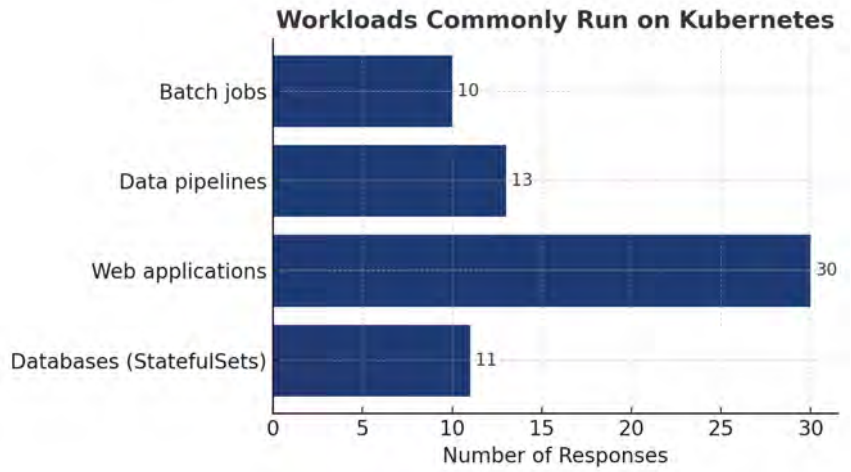


Figure 5.1: Workloads Commonly Run on Kubernetes

Cloud-managed clusters such as EKS, GKE, and AKS were the most common choice, used by 67.7% of respondents, highlighting the appeal of managed services for ease of operation. Self-managed, on-premises clusters were also significant, adopted by 61.3%, indicating that many organizations still prefer direct control over their environments. Hybrid setups, combining both models, were less frequent (16.1%), but they point toward a growing trend of balancing flexibility and governance. Overall, the findings suggest that while cloud-managed clusters dominate, a notable share of teams still invest in on-prem solutions, reflecting diverse operational needs.

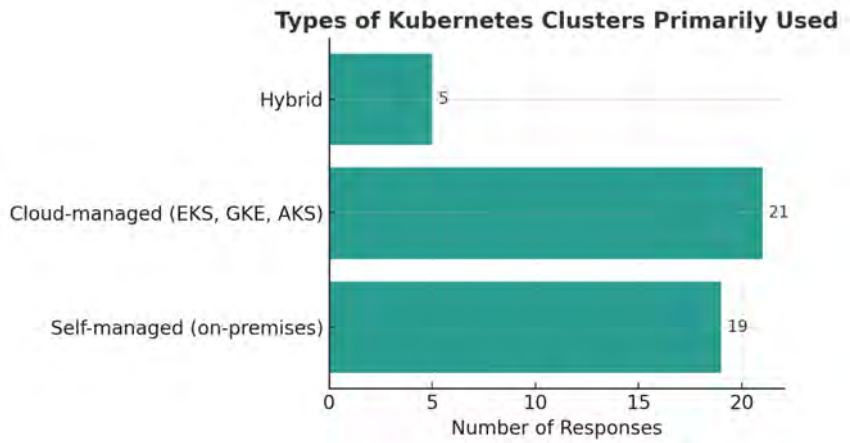


Figure 5.2: Types of Kubernetes Clusters Primarily Used

Table 5.1: Workloads managed with corresponding percentages and proportions.

Element	Percentage
Databases(StatefulSets)	34.4
Web Applications	93.8
Data Pipelines	40.6
Batch Jobs	31.3

5.2.2 Infrastructure as Code

The survey responses show Helm leads adoption at 65.6%, followed by Terraform at 62.5%. GitOps tools—Argo CD (31.25%) and Flux (31.25%) along with Ansible (31.3%) show moderate (one-third) uptake, typically complementing Helm/Terraform workflows.

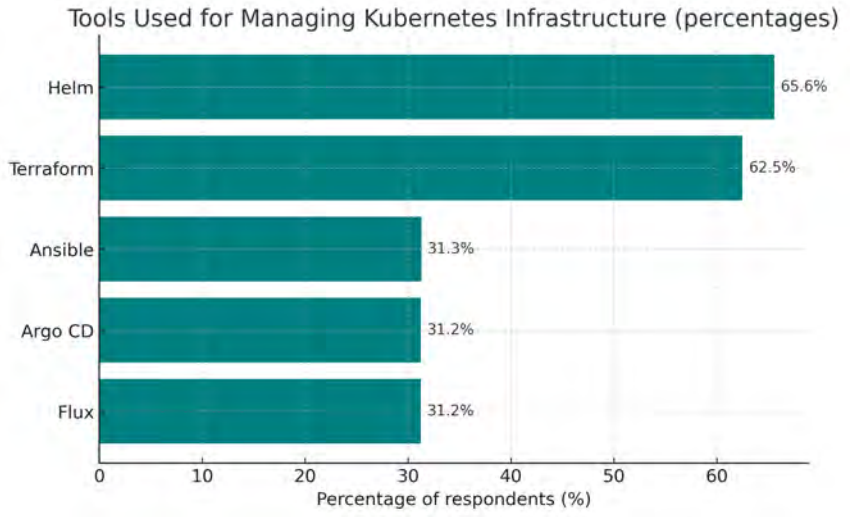


Figure 5.3: Tools Used for Managing Kubernetes Infrastructure

Table 5.2: Infrastructure-as-Code (IaC) elements with percentages and proportions.

Element	Usage Percentage
Terraform	62.5
Helm	65.60
Ansible	31.3
Argo CD	31.25
Flux	31.25

5.2.3 Objects

In workload management, the results underscore the dominance of core scheduling and deployment objects. **Pods** and **Deployments** are the most consistently reported, with over 93% of respondents frequently using them. These objects form the backbone of Kubernetes application management and are indispensable in day-to-day practice. **ReplicaSets** also show high adoption, with nearly 87% of respondents using them frequently, though their role is often implicit as they are created automatically through Deployments.
Less frequent, but still significant, are **DaemonSets** and **StatefulSets**, which cater to specialized use cases. DaemonSets are employed by about three-quarters of respondents, often for background services such as monitoring or logging agents. StatefulSets, while less dominant, are still adopted by 61% of respondents, reflecting their importance in managing stateful workloads like databases. Their relatively lower adoption compared to Deployments and Pods suggests that not all organizations run workloads that demand persistent state.

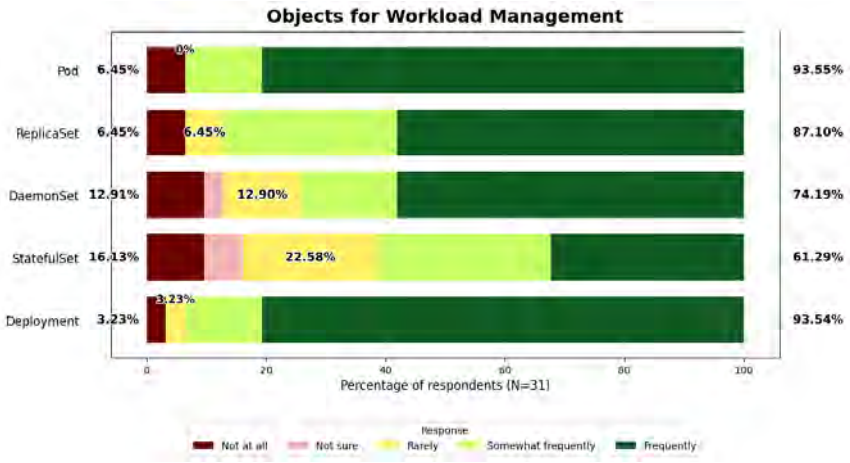


Figure 5.4: Usage of Kubernetes Objects for Workload Management

The survey responses show distinct variation in how practitioners rely on Kubernetes objects for managing network-related configurations. **Services** stand out as the most widely used object, with more than 90% of respondents reporting frequent use. This reinforces the role of Services as a foundational building block in enabling connectivity between workloads. **Ingress** follows closely, with 87% of respondents indicating frequent adoption, reflecting its central role in routing external traffic.

By contrast, **NetworkPolicy** reveals a more uneven picture. While 58% of respondents report using it frequently, nearly 29% indicate they do not use it at all, highlighting variability in how security and isolation policies are enforced across organizations. **EndpointSlice** appears less common, with more than half of respondents (55%) reporting they do not use it, and only about one-quarter adopting it with any regularity. This suggests that while newer networking constructs exist, they are not yet mainstream among practitioners.

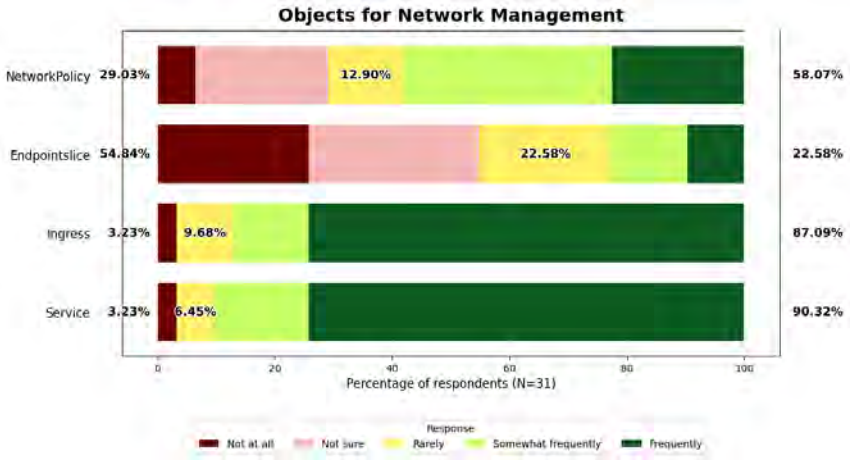


Figure 5.5: Usage of Kubernetes Objects for Network Management

In storage and configuration management, the survey responses highlight the widespread use of **Secrets** and **ConfigMaps**, with more than 90% of respondents indicating frequent adoption of both. These objects are fundamental for injecting configuration data and managing sensitive information in Kubernetes applications. **Persistent Volumes (PV)** and **Persistent Volume Claims (PVC)** also show considerable adoption, with 77% and 74% of respondents respectively reporting frequent use. Their slightly lower usage compared to Secrets and ConfigMaps suggests that while persistent storage is important, not all workloads require it. Overall, the results indicate that configuration objects are nearly universal, while storage-related constructs are adopted more selectively depending on workload demands.

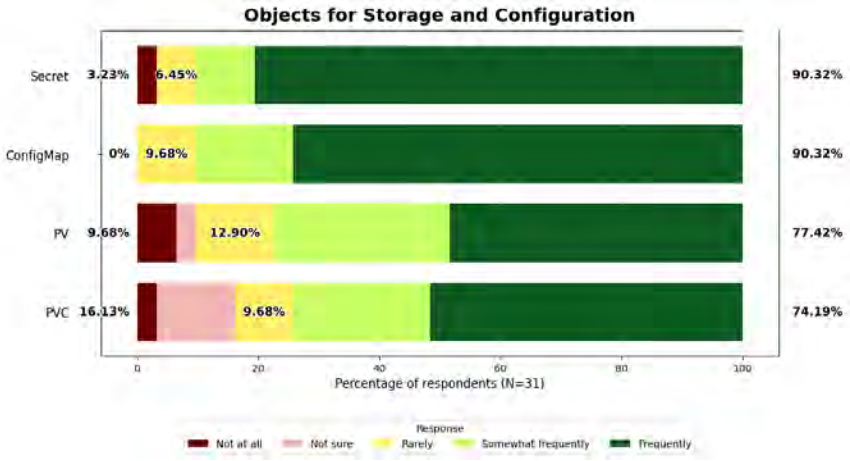


Figure 5.6: Usage of Kubernetes Objects for Storage and Configuration

For scaling and policy-related objects, the survey highlights clear differences in adoption. **Namespaces** emerge as the most widely used, with over 90% of respondents indicating frequent usage, confirming their central role in organizing and isolating Kubernetes resources. **Autoscalers** (HPA/VPA) also show strong adoption, with 84% of respondents reporting frequent use, underlining the importance of automated scaling for resource efficiency. In contrast, policy-oriented constructs reveal more uneven adoption. **PodSecurityPolicy** is used frequently by 61% of respondents but is avoided or unused by more than one-third, reflecting the complexity and deprecation of this feature. Similarly, **CustomResourceDefinitions (CRDs)** are adopted by 45% of respondents, indicating their growing but still specialized role in extending Kubernetes functionality. Meanwhile, **LimitRange** remains the least adopted, with less than 40% using it frequently and a large proportion (45%) not using it at all, suggesting that resource constraint policies are not consistently enforced across organizations.

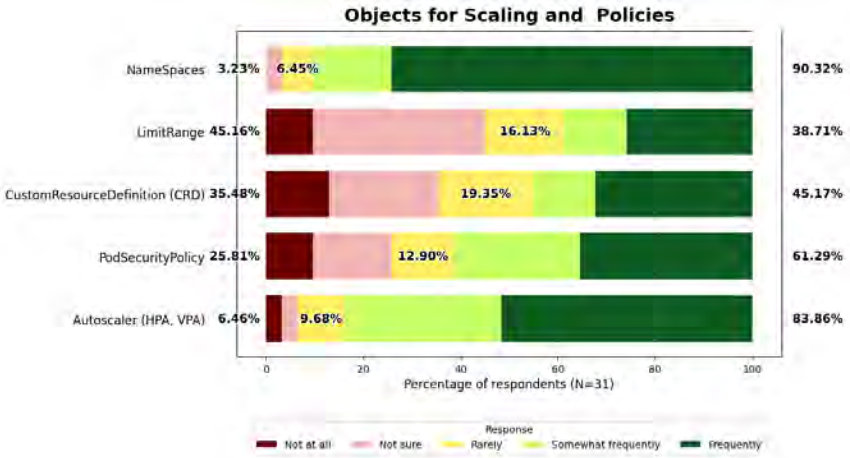


Figure 5.7: Usage of Kubernetes Objects for Scaling and Policies

Table 5.3: Kubernetes object usage with corresponding percentages (frequently used).

Element	Usage Percentage (Freq. used)
Deployment	93.54
Pod	93.55
ReplicaSet	87.10
DaemonSet	74.19
StatefulSet	61.29
Service	90.32
Ingress	87.09
NetworkPolicy	58.07
EndpointSlice	22.58
ConfigMap	90.32
Secret	90.32
PersistentVolume (PV)	77.42
PersistentVolumeClaim (PVC)	74.19
Namespace	90.32
LimitRange	38.71
CustomResourceDefinition (CRD)	45.17
PodSecurityPolicy	61.29
Autoscaler (HPA/VPA)	83.87

5.2.4 Kubernetes Core Components and System Components

The survey results on Kubernetes system components reveal that **kubelet**, **kube-scheduler**, and **etcd** are the most consistently used, with around 71% of respondents indicating frequent adoption. These components are essential for the operation of Kubernetes clusters, reflecting their universal role in workload execution, scheduling, and maintaining cluster state. The **kube-controller-manager** is also widely adopted (65%), while the **cloud-controller-manager** shows a more mixed pattern, with only 52% of respondents using it frequently and a significant share not adopting it at all. This variation suggests that while core system services are indispensable, cloud-specific extensions are used more selectively, depending on organizational environments.

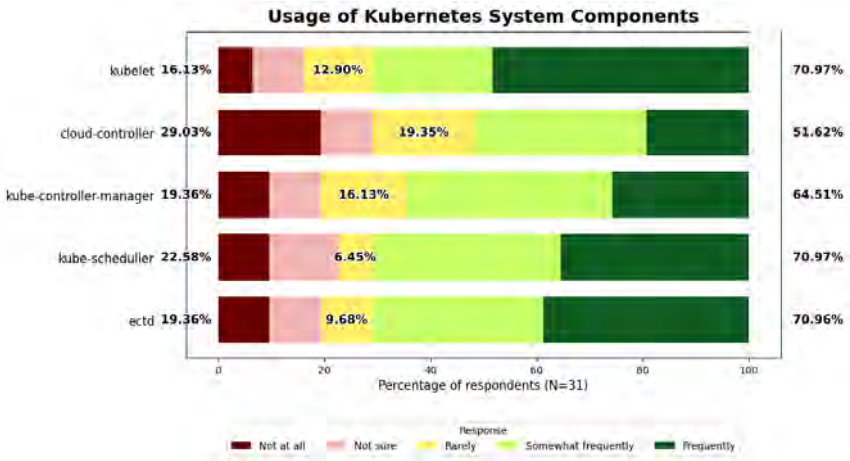


Figure 5.8: Usage of Kubernetes System Components

For core components, the survey highlights the strong adoption of **Controllers** (77%) and the **Scheduler** (68%), underscoring their central role in orchestrating workloads. **Operators** also show considerable adoption (58%), reflecting their growing role in managing complex applications. By contrast, **Kube-DNS** (55%) and especially the **Node Planner** (48%) are less frequently used, indicating that DNS services and node-level planning are not universally prioritized across organizations. Overall, the results suggest that while fundamental controllers and schedulers are widely integrated, specialized components such as node planning or DNS services exhibit more uneven uptake.

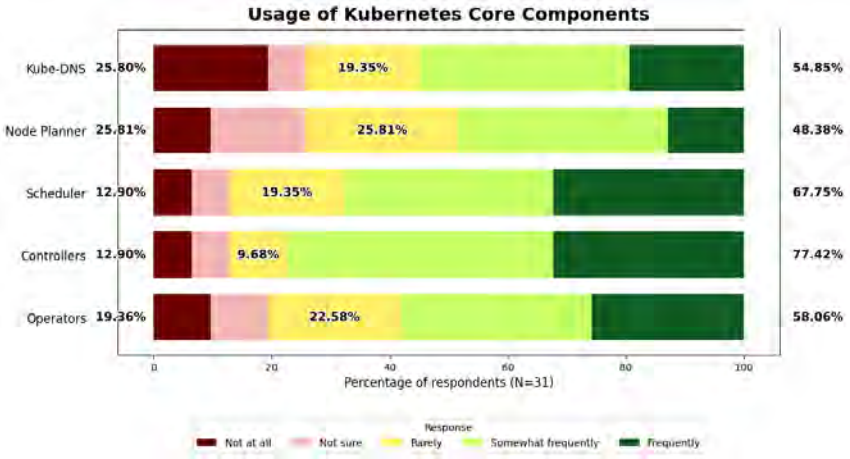


Figure 5.9: Usage of Kubernetes Core Components

Table 5.4: Kubernetes component usage with corresponding percentages (frequently used).

Element	Usage Percentage (Freq. used)
kubelet	70.97
etcd	70.96
kube-scheduler	70.97
kube-controller-manager	64.51
cloud-controller-manager	51.62
Controllers	77.42
Scheduler	67.75
Operators	58.06
Kube-DNS	54.85
Node Planner	48.38

5.2.5 Usage of Monitoring and Logging Tools

Among the logging and monitoring tools, Prometheus is the clear front-runner, reported by 74.2% of respondents. Datadog follows at 28.1%. Among the remaining tools, KubeDB (15.6%) and OpenSearch (12.5%) show moderate uptake, while Lens (6.3%) and Grafana (3.1%) appear only occasionally. Overall, the results indicate a strong tilt toward the open-source Prometheus ecosystem, with other options playing more targeted or niche roles.

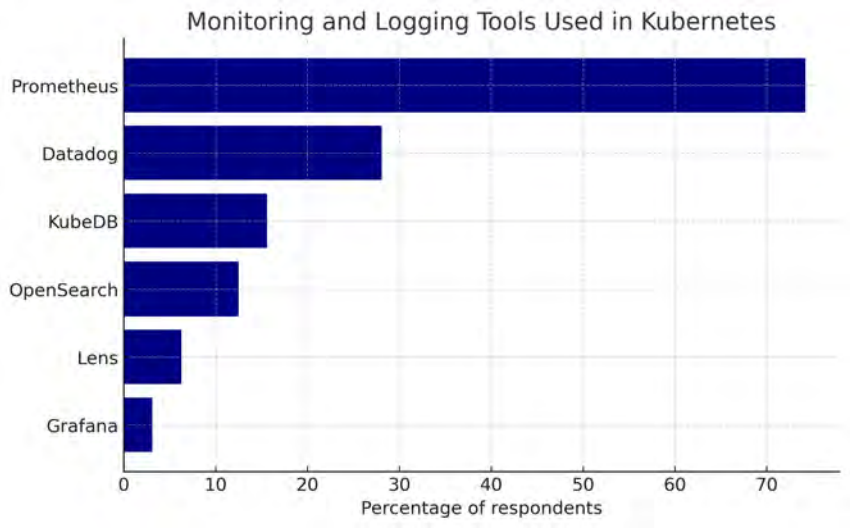


Figure 5.10: Monitoring and Logging Tools Used in Kubernetes

Table 5.5: Logging and monitoring tool usage with corresponding percentages and proportions.

Element	Percentage
Prometheus	74.2
Datadog	28.1
OpenSearch	12.5
KubeDB	15.6
Lens	6.3
Graphana	3.1

5.2.6 Usage of Pod Scheduling Features

The survey responses on scheduling features reveal a selective pattern of adoption. **Node Affinity** emerges as the most widely used, with 61% of respondents reporting frequent usage, reflecting its importance in ensuring workloads are placed on appropriate nodes. Both **Pod Anti-Affinity** and **Taints and Tolerations** are used by nearly half of the respondents (48%), indicating that while these features are valuable for workload isolation and placement control, they are not universally applied across organizations. Overall, the findings suggest that scheduling policies are employed strategically, with Node Affinity being the most common, while Anti-Affinity rules and Taints and Tolerations see more situational use.

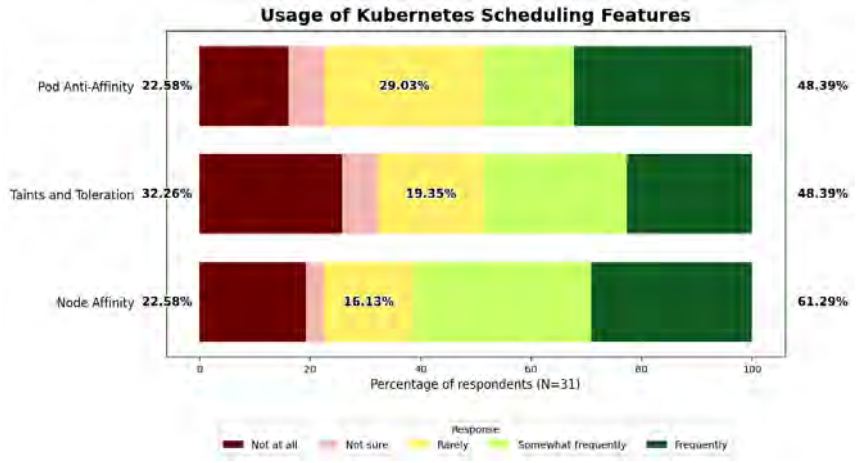


Figure 5.11: Usage of Kubernetes Pod Scheduling Features

Table 5.6: Scheduling feature usage with corresponding percentages (frequently used).

Element	Usage Percentage (Freq. used)
Pod Anti-Affinity	48.39
Taints and Tolerations	48.39
Node Affinity	61.29

5.2.7 Manual Effort and Challenges in Kubernetes

The survey responses indicate that Kubernetes upgrades are widely perceived as a challenging task. Nearly half of the participants (48%) described upgrades as *challenging*, while an additional 29% considered them *somewhat challenging* or *very challenging*. Only about 23% reported that upgrades were not a challenge at all. These findings suggest that upgrading Kubernetes clusters often requires significant manual effort and careful coordination, making it one of the more demanding aspects of cluster maintenance.

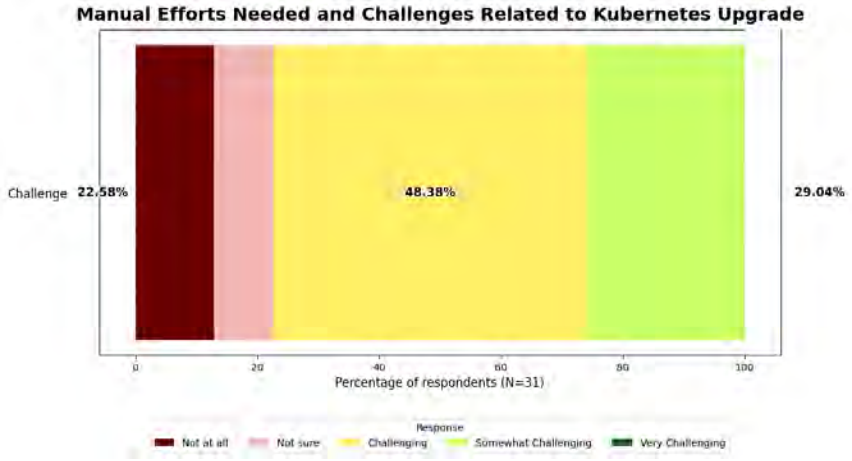


Figure 5.12: Perceived Manual Effort and Challenges in Kubernetes Upgrades

The survey responses reveal that network and security tasks remain a consistent source of difficulty for Kubernetes practitioners. The reported percentages here reflect the combined share of respondents who rated each task as *challenging*, *somewhat challenging*, or *very challenging*. **Secrets management** stood out as a major issue, with 42% of respondents experiencing difficulties, reflecting the sensitivity and complexity of securely handling credentials. Similarly, **API exposure risks** were cited by 49% of respondents as a challenge, highlighting concerns about protecting external interfaces. **RBAC integration** (52%) and **IMA integration** (58%) also emerged as commonly reported difficulties, underscoring the configuration overhead required for fine-grained access control and system integrity. Notably, **network security** was identified as a challenge by 45% of respondents, making it one of the most prominent security-related concerns. Taken together, these findings indicate that while Kubernetes offers rich security features, their practical implementation demands significant effort and introduces notable challenges in real-world environments.

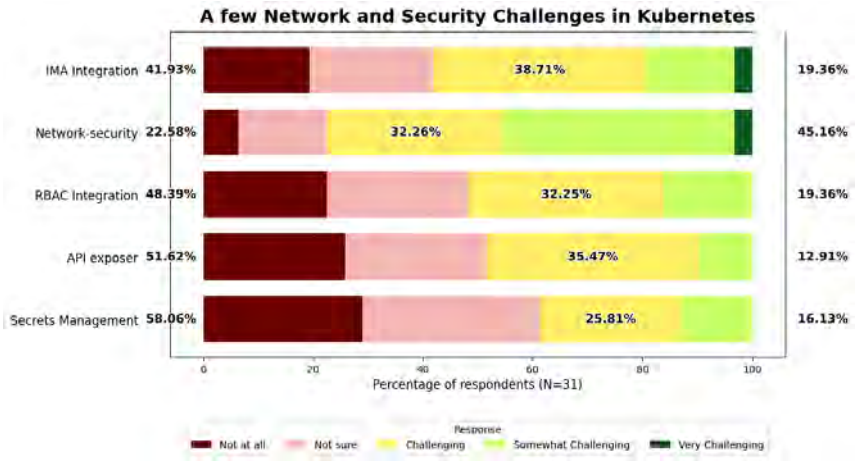


Figure 5.13: Challenges in Kubernetes Security Tasks

The survey results highlight that practitioners frequently encounter operational difficulties in Kubernetes. The reported percentages reflect the combined share of respondents who rated each issue as *challenging*, *somewhat challenging*, or *very challenging*. Among the most common problems, **port forwarding failures** stood out, with 84% of respondents identifying them as a challenge, making it the most widely reported operational difficulty. **DNS issues** (71%) and **API rate limits** (68%) were also highly cited, indicating recurring reliability bottlenecks in cluster operations. Additionally, over half of the respondents experienced challenges with **external IP issues** (58%), **resource waste due to poor scheduling** (52%), and **network ingress setup** (52%), underscoring the complexity of managing connectivity and efficient resource usage. **CrashLoopBackOff errors** were reported as a challenge by 42% of respondents, pointing to ongoing instability in workload execution. Overall, these results emphasize that even routine operational tasks in Kubernetes often demand manual oversight and remain a consistent source of frustration for practitioners.

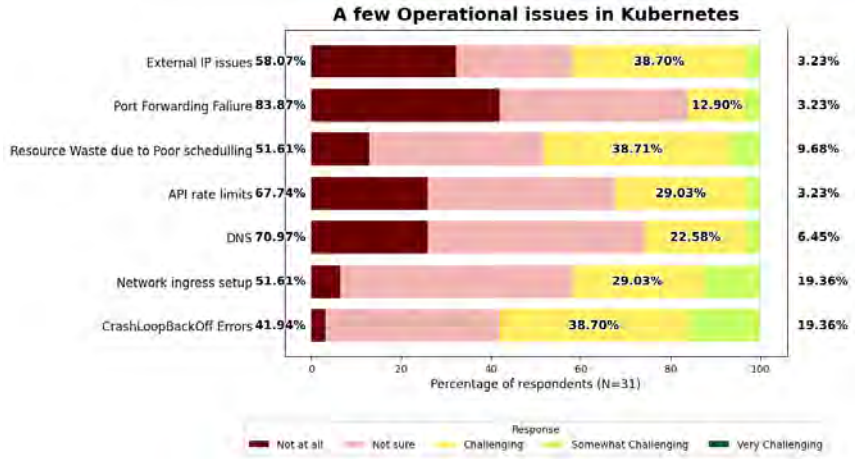


Figure 5.14: Frequency of Operational Issues in Kubernetes

Overall, these findings suggest that while Kubernetes provides powerful abstractions for managing workloads at scale, practitioners continue to encounter a wide range of manual and technical hurdles. Upgrades and scaling require careful planning,

security integration often introduces added complexity, and day-to-day operational issues remain a persistent source of overhead.

5.3 Repository Findings

In addition to the insights from interviews and surveys, we examined an extensive collection of public Kubernetes-related repositories (n = 1,429). This analysis offers an outsider perspective on adoption, practices, and ecosystem indicators as they appear in open-source codebases. By examining the presence of Kubernetes objects, components, infrastructure-as-code frameworks, workload patterns, and operational features, we aimed to identify how practitioners' choices are reflected in real repositories. These findings not only anchor our study in empirical evidence but also establish a foundation for triangulating with the qualitative and survey data in the next sections.

5.3.1 Workloads

Kubernetes repositories reveal not just the use of core objects but also the types of workloads developers prioritize. By mapping workload-related signals across 1,429 repositories, we can see which application types are most commonly deployed and how their prevalence differs.

Figure 5.15: Repository Signals of Workloads Across 1,429 Repositories (Bar chart showing the number of repositories that contain different workload types)

Table 5.7: Distribution of Workloads Across Repositories

(Repo counts and percentages for each workload type)

Element	Repo_Count	Percentage
Web applications	810	56.68
Databases (StatefulSets)	459	32.12
Batch jobs	195	13.65
Data pipelines	4	0.28

The results (Figure 5.15, Figure 5.16, Table 5.7) highlight a clear dominance of web applications, present in over half of the repositories. Databases, often managed

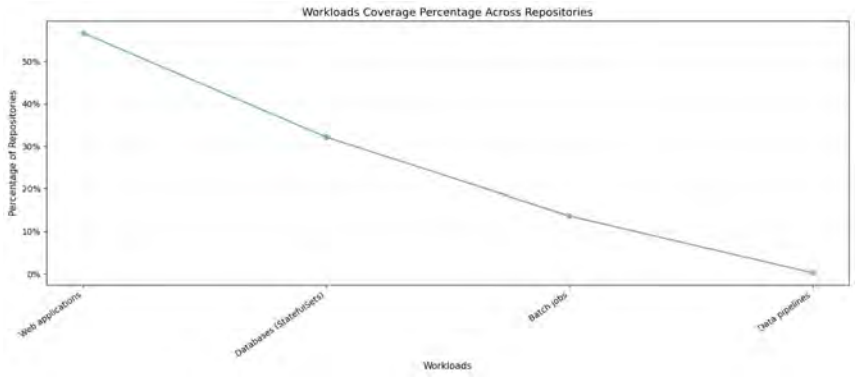


Figure 5.16: Workloads Coverage Percentage Across Repositories (Line chart showing workload usage as a percentage of repositories)

through StatefulSets, form the next most common category, followed by batch jobs. Data pipelines, by contrast, appear rarely. Together, these patterns suggest that while Kubernetes is widely adopted for running user-facing and data-serving applications, its use for more specialized workloads like pipelines remains limited.

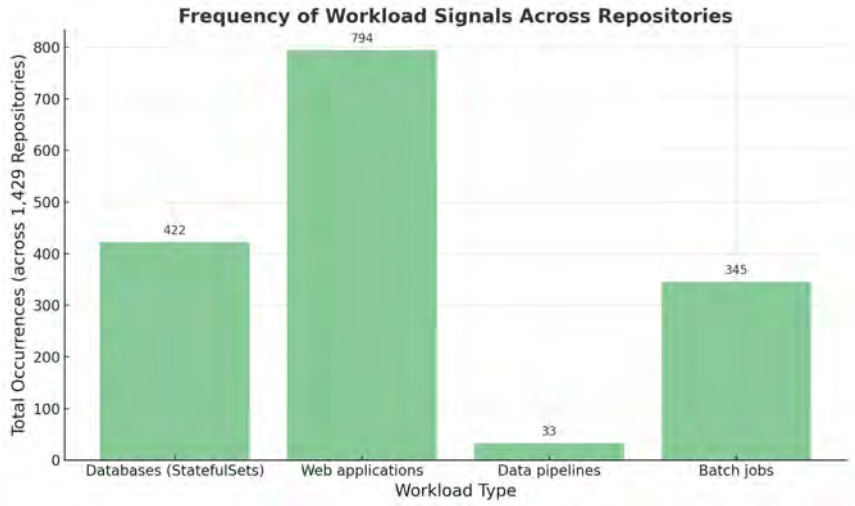


Figure 5.17: Frequency of workload Signals Across Repositories (Bar chart showing the number times different workloads are dealt across 1429 repositories)

The chart (figure 5.17) shows that web applications are the dominant workload in public repositories, with nearly twice as many occurrences as any other category. Databases (via StatefulSets) also appear prominently, despite being often considered challenging to run on Kubernetes. Batch jobs are present in a moderate number of cases, highlighting their role in scheduled or background processing. By contrast, data pipelines are rarely observed, suggesting they are still a niche use case in open-source projects.

5.3.2 Infrastructure as Code

Infrastructure as Code tools provide the scaffolding that makes Kubernetes deployments reproducible and maintainable. By analyzing repository signals, we can observe which IaC frameworks are most frequently adopted and how their usage spreads across the ecosystem.

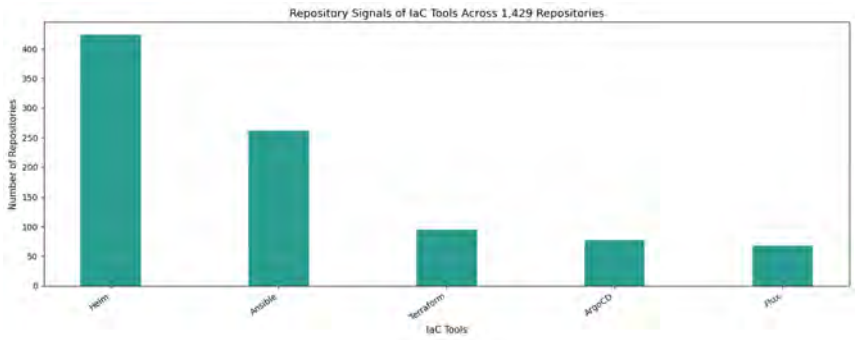


Figure 5.18: Repository Signals of IaC Tools Across 1,429 Repositories (Bar chart showing the number of repositories that include different IaC tools)

Table 5.8: Distribution of IaC Tools Across Repositories

(Repo counts and percentages for each IaC tool)

Element	Repo_Count	Percentage
Helm	424	29.67
Ansible	261	18.26
Terraform	95	6.65
ArgoCD	77	5.39
Flux	68	4.76

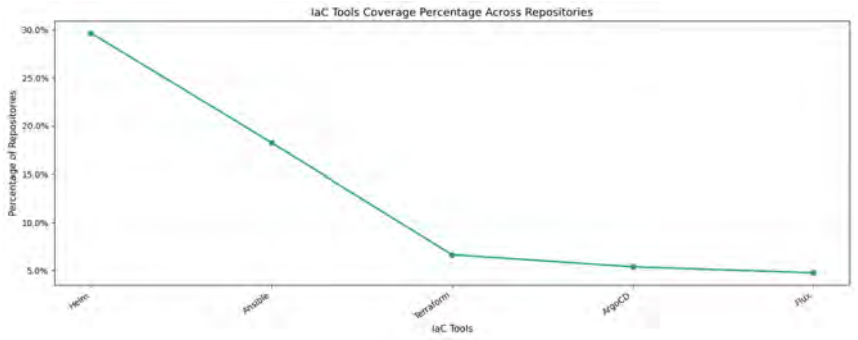


Figure 5.19: IaC Tools Coverage Percentage Across Repositories (Line chart showing IaC tool usage as a percentage of repositories)

The analysis (Figure 5.18, Figure 5.19, Table 5.8) shows Helm leading by a wide margin, appearing in nearly 30% of repositories. Ansible follows as the second most common tool, while Terraform appears less frequently. GitOps-oriented tools such as ArgoCD and Flux are present but still represent a smaller fraction of repositories.

Overall, this indicates that while Helm has become the de facto standard for managing Kubernetes manifests, newer GitOps tools are still in the process of gaining traction in the open-source community.

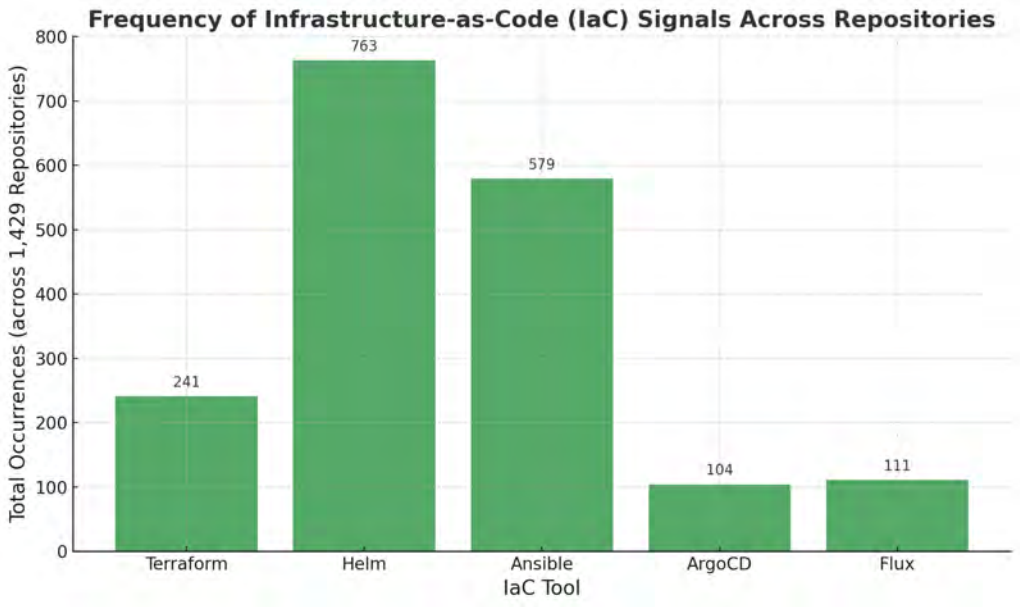


Figure 5.20: Frequency of Infrastructure-as-Code (IaC) Signals Across Repositories (Bar chart showing the number times different IaC tools are used across 1429 repositories)

The chart (Figure 5.20) shows that Helm is by far the most common IaC signal in public repositories, reflecting its popularity for packaging and managing Kubernetes applications. Ansible and Terraform also appear frequently, highlighting their role in provisioning and configuration management alongside Kubernetes. In contrast, GitOps-oriented tools such as ArgoCD and Flux are present but in much smaller numbers, suggesting that while these approaches are gaining traction, they remain less widely adopted in the open-source landscape.

5.3.3 Object Usage

Kubernetes objects form the backbone of any cluster, defining workloads, configurations, and access policies. By examining their presence across repositories, we can identify which objects dominate real-world usage and which remain more niche.

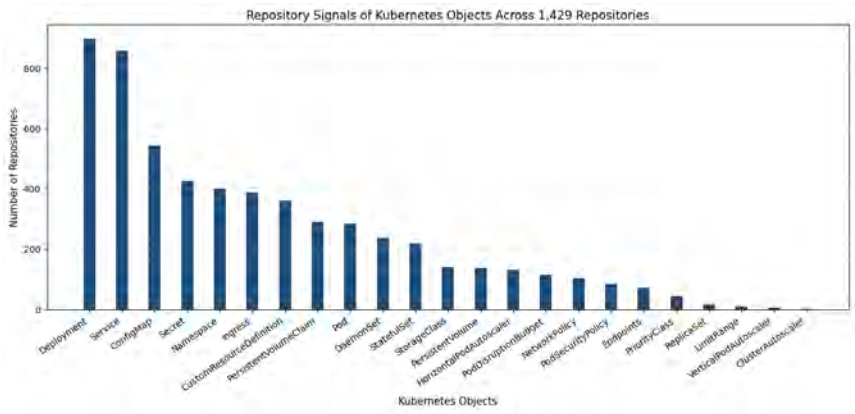


Figure 5.21: Repository Signals of Kubernetes Objects Across 1,429 Repositories (Bar chart showing the number of repositories that use different Kubernetes objects)

Table 5.9: Distribution of Kubernetes Objects Across Repositories (Repo counts and percentages for each object type)

Objects	**Repo_Count**	**Percentage (%)**
Deployment	900	62.98
Service	858	60.04
ConfigMap	543	38.00
Secret	427	29.88
Namespace	401	28.06
Ingress	387	27.08
CustomResourceDefinition	362	25.33
PersistentVolumeClaim	292	20.43
Pod	285	19.94
StatefulSet	219	15.33
StorageClass	140	9.80
PersistentVolume	138	9.66
HorizontalPodAutoscaler	132	9.24
PodDisruptionBudget	113	7.91
NetworkPolicy	103	7.21
PodSecurityPolicy	85	5.95
Endpoints	71	4.97
ReplicaSet	18	1.26
LimitRange	10	0.70
VerticalPodAutoscaler	5	0.35

The findings (Figure 5.21, Figure 5.22, Table 5.9) clearly show that Deployment and Service are the most prevalent, appearing in over 60% of repositories, underscoring their central role in Kubernetes application management. ConfigMaps and Secrets also appear widely, highlighting the importance of configuration and credential management in real-world projects. Namespace and Ingress usage further illustrates how teams structure clusters and manage external access. Beyond these, more advanced or specialized objects such as HorizontalPodAutoscaler, NetworkPolicy, and PodSecurityPolicy show moderate adoption, while features like VerticalPodAutoscaler or ClusterAutoscaler appear only rarely. Taken together, this suggests a pattern where

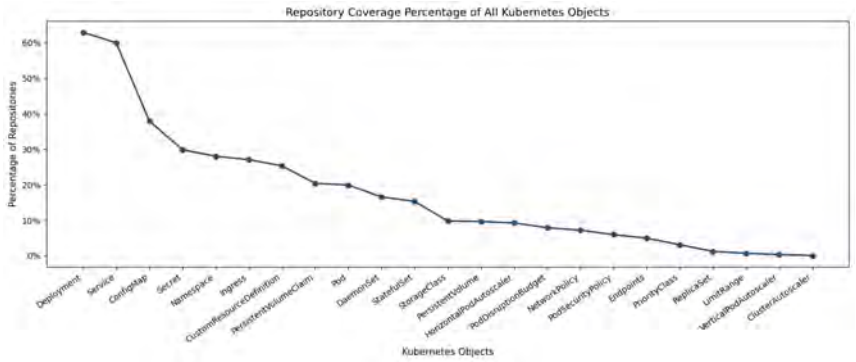


Figure 5.22: Repository Coverage Percentage of Kubernetes Objects (Line chart showing object usage as a percentage of repositories)

foundational objects dominate practice, while more advanced constructs remain selectively adopted.

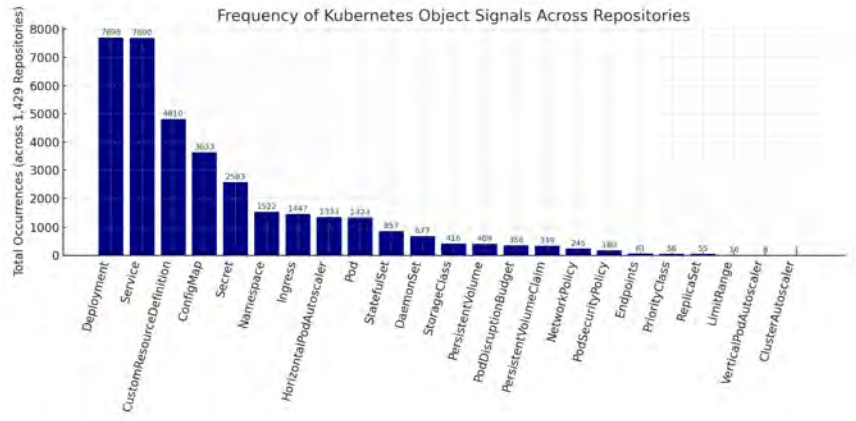


Figure 5.23: Frequency of Objects Signals Across Repositories (Bar chart showing the number times different objects are used across 1429 repositories)

The bar chart (Figure 5.23) visually reinforces the repository evidence: Deployment is by far the most frequently encountered object, with Service also appearing heavily across projects. ConfigMap and Secret are widely used, reflecting common needs for configuration and credential handling. Meanwhile, objects tied to scaling or policy enforcement appear only sporadically, suggesting that while Kubernetes provides many advanced features, most public repositories rely primarily on its core objects for everyday workloads.

5.3.4 Component Usage

Beyond workloads and objects, repositories also signal which core Kubernetes components developers directly interact with or configure. These components reflect how practitioners extend, customize, and operate their clusters in practice.

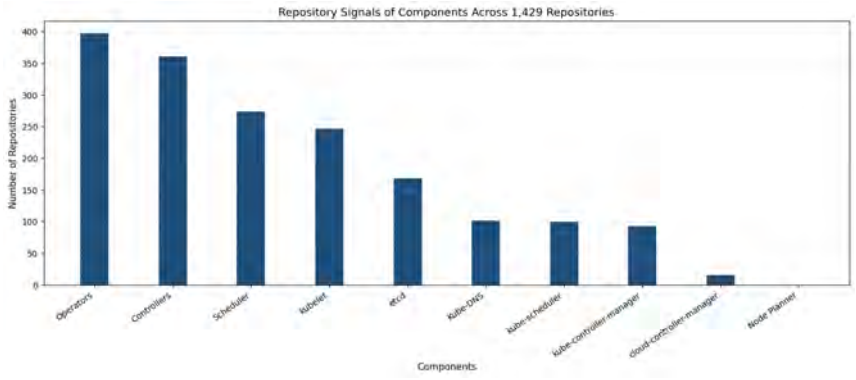


Figure 5.24: Repository Signals of Components Across 1,429 Repositories (Bar chart showing the number of repositories referencing different Kubernetes components)

Table 5.10: Distribution of Kubernetes Components Across Repositories

(Repo counts and percentages for each component)

Element	**Repo_Count**	**Percentage (%)**
Operators	397	27.78
Controllers	361	25.26
Scheduler	274	19.17
kubelet	246	17.21
etcd	168	11.76
Kube-DNS	101	7.07
kube-scheduler	99	6.93
kube-controller-manager	93	6.51
cloud-controller-manager	15	1.05
Node Planner	0	0.00

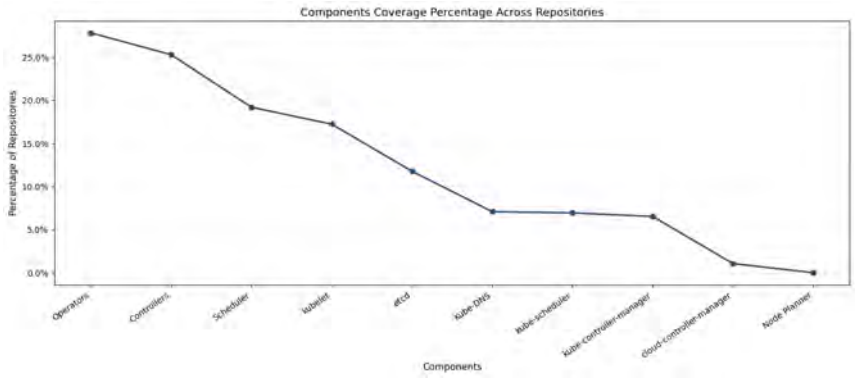


Figure 5.25: Components Coverage Percentage Across Repositories (Line chart showing component usage as a percentage of repositories)

The most prominent signals come from Operators and Controllers, appearing in more than a quarter of repositories, reflecting their central role in automation and extending Kubernetes functionality. Scheduler and kubelet references also feature strongly, indicating attention to core workload management and node-level execution (Figure 5.24, Figure 5.25, Table 5.10). By contrast, lower-level components like etcd, Kube-DNS, and controller-managers are present but less common, suggesting that

while critical to Kubernetes operation, they are less frequently exposed in repository-level configuration. Interestingly, some components like Node Planner appear absent altogether, showing a divide between what practitioners configure directly and what remains largely under the hood.

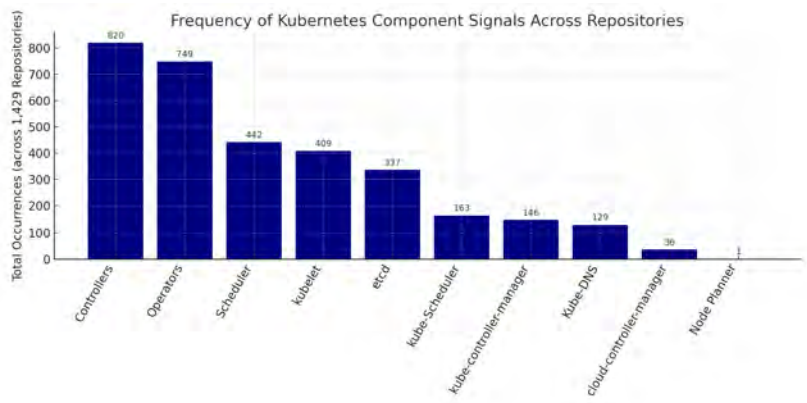


Figure 5.26: Frequency of Kubernetes Component Signals Across Repositories (Bar chart showing the number times different components are used across 1429 repositories)

The chart (Figure 5.26) shows that controllers and operators are the most widely observed components across public repositories, underscoring their central role in managing Kubernetes workloads and extending cluster functionality. Core infrastructure services like the scheduler, kubelet, and etcd also appear frequently, reflecting their importance in cluster orchestration and node-level operations. By contrast, specialized components such as cloud-controller-manager or node planner show minimal adoption, suggesting that while fundamental components are consistently present, advanced or environment-specific components remain far less common in open-source repositories.

5.3.5 Logging and Monitoring Tools

Observability is essential for operating Kubernetes at scale, and repositories often reveal which logging and monitoring tools developers rely on most. These signals highlight both the maturity of monitoring practices and the degree to which open-source communities standardize on certain tools.

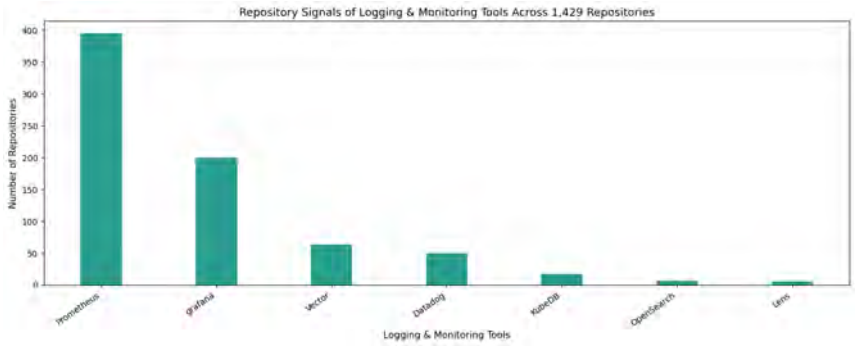


Figure 5.27: Logging and Monitoring Tools Coverage Percentage Across Repositories (Line chart showing logging and monitoring tools usage as a percentage of repositories)

Table 5.11: Distribution of Logging and Monitoring Tools Across Repositories

(Repo counts and percentages for each tool)

Element	Repo_Count	Percentage (%)
Prometheus	395	27.64
grafana	200	14.00
Vector	63	4.41
Datadog	50	3.50
KubeDB	17	1.19
OpenSearch	6	0.42
Lens	5	0.35

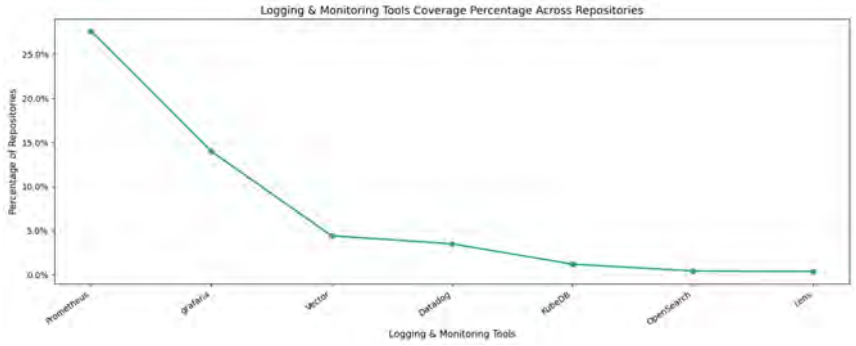


Figure 5.28: Repository Signals of Logging and Monitoring Tools Across 1,429 Repositories (Bar chart showing the number of repositories referencing different logging and monitoring tools)

The findings (Figure 5.27, Figure 5.28, Table 6.11) clearly show that observability within Kubernetes projects is strongly anchored around the Prometheus–Grafana stack. Prometheus emerges as the most dominant tool, appearing in more than a quarter of repositories, while Grafana is the second most common, reflecting its role as the go-to visualization layer. Other tools such as Vector, Datadog, and Lens appear, but at much lower levels, suggesting they are used in specific contexts rather than as defaults. Meanwhile, KubeDB and OpenSearch are rarely observed, pointing to their more specialized adoption. Taken together, the results illustrate

that while a variety of logging and monitoring options exist, the open-source ecosystem overwhelmingly gravitates toward the Prometheus–Grafana combination as the standard observability stack.

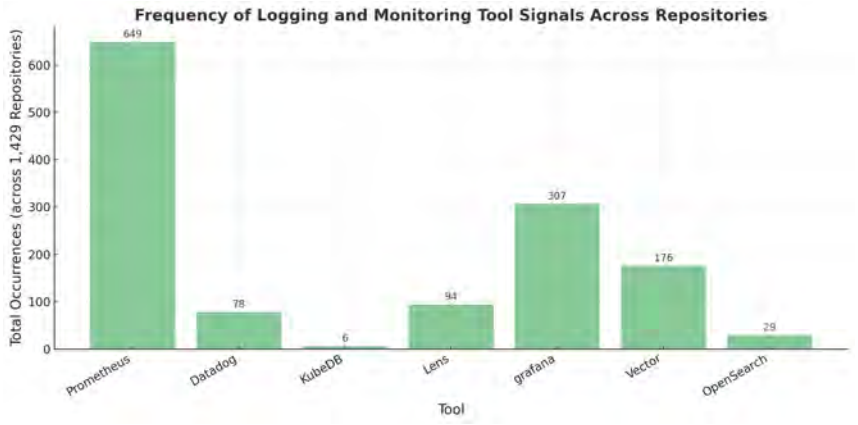


Figure 5.29: Frequency of Kubernetes Logging and Monitoring Tools Signals Across Repositories (Bar chart showing the number times different logging and monitoring tools are used across 1429 repositories)

Unsurprisingly, Prometheus stands out as the most widely adopted tool, present in more than a quarter of repositories, with Grafana following as the primary visualization layer (Figure 5.29). Other tools like Vector and Datadog appear but are far less common, while newer or more specialized options such as KubeDB, OpenSearch, and Lens show only marginal adoption. This distribution suggests that open-source communities have largely consolidated around the Prometheus–Grafana stack, while alternative or commercial solutions remain secondary choices.

5.3.6 Scheduling Features

Scheduling policies determine how workloads are distributed across nodes in a cluster. Repository signals around these features provide a window into how often practitioners explicitly manage workload placement and resource isolation.

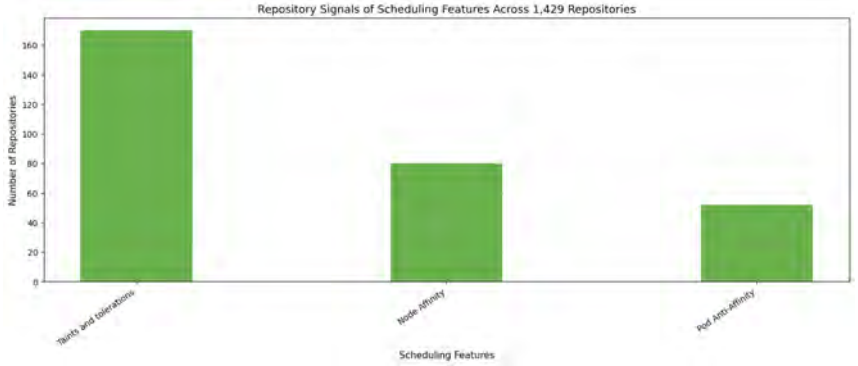


Figure 5.30: Repository Signals of Scheduling Features Across 1,429 Repositories (Bar chart showing the number of repositories using different scheduling features)

Table 5.12: Distribution of Scheduling Features Across Repositories

(Repo counts and percentages for each scheduling feature)

Element	Repo_Count	Percentage (%)
Taints and tolerations	170	11.90
Node Affinity	80	5.60
Pod Anti-Affinity	52	3.64

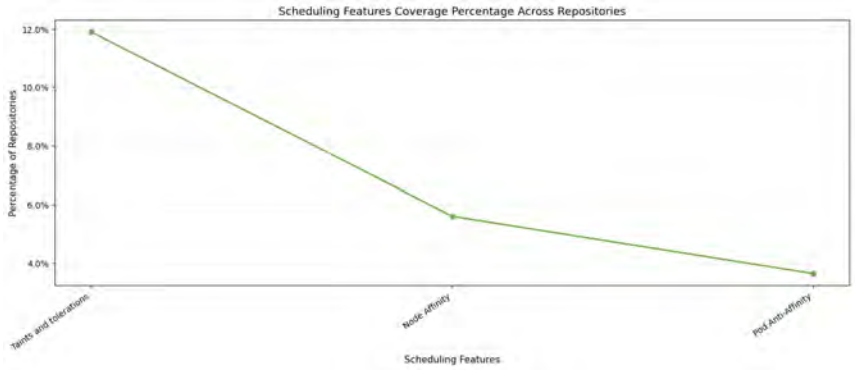


Figure 5.31: Scheduling Features Coverage Percentage Across Repositories (Line chart showing scheduling feature usage as a percentage of repositories)

The results (Figure 5.30, Figure 5.31, Table 5.12) show that taints and tolerations are the most common scheduling feature, found in nearly 12% of repositories. Node affinity appears less frequently, while pod anti-affinity is the least used among the three. This pattern suggests that although scheduling controls are available, most repositories do not heavily rely on them, with only a minority adopting these features to fine-tune workload placement.

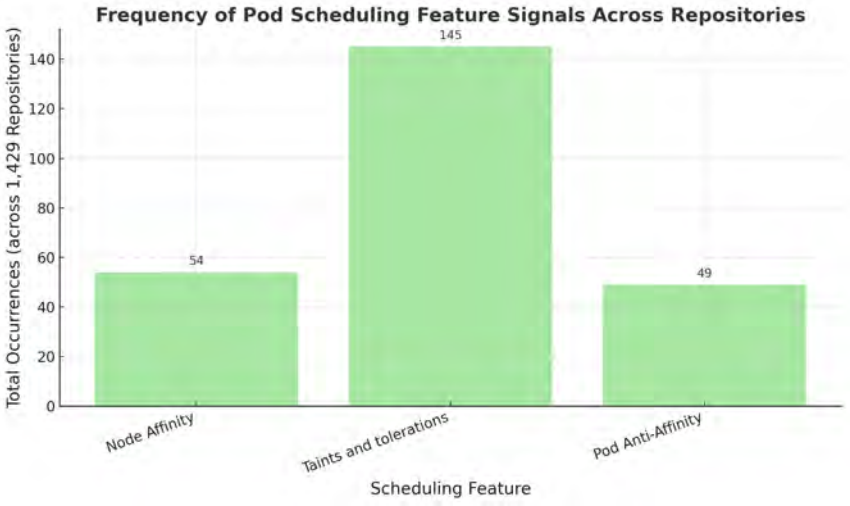


Figure 5.32: Frequency of Kubernetes Scheduling Features Signals Across Repositories (Bar chart showing the number times different Scheduling Features are used across 1429 repositories)

The chart (Figure 5.32) shows that scheduling-related features are relatively uncommon in public repositories. Taints and tolerations appear most frequently, indicating

their role in workload isolation and control over node assignments. In contrast, node affinity and pod anti-affinity are used less often, suggesting that while advanced scheduling strategies are available in Kubernetes, many projects still rely on simpler default scheduling mechanisms.

5.3.7 Validation and Evaluation

Validation: To validate the stability of my scanner, I randomly selected the first 50 repositories out of the 1,429 from the `yaml_matrix.csv`file (this file records whether a Kubernetes signal such as any object, any component, any IaC, etc., is present in a repo as 1 or absent as 0). I cloned these 50 repositories again and regenerated the `yaml_matrix.csv`. The results were identical to the earlier scan, which confirmed that the scanner consistently produces stable outputs and can be considered valid.

Evaluation: For evaluation, I again used the first 50 repositories. This time, I created a "goldFile" by manually checking each repository. Whenever my scanner predicted a signal as absent (0) but I found it present, I corrected it to 1 (this is a false negative → FN). Likewise, if the scanner marked a signal as present (1) but it was not in the repo, I corrected it to 0 (this is a false positive → FP). Correct predictions were true positives (TP) and true negatives (TN). Using these counts, I calculated precision, recall, F1-score, and accuracy for each group of signals.

Groupwise Metrics:

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP+FP} \\ \text{Recall} &= \frac{TP}{TP+FN} \\ \text{F1} &= \frac{2 \times TP \times Recall}{Precision + Recall} \\ \text{Accuracy} &= \frac{TP+TN}{TP+FP+FN+TN} \end{aligned}$$

Table 5.13: Group-wise Evaluation Matrix for 50 Repositories

Category	Precision	Recall	F1	Accuracy
Objects	0.85	0.83	0.84	0.96
Components	0.95	0.95	0.95	0.99
IaC	0.95	0.84	0.89	0.98
Scheduling Features	0.60	1.00	0.75	0.99
Logging and monitoring	0.94	0.94	0.94	0.99
Workloads	1.00	0.91	0.95	0.98

Overall Metrics:

$$
\begin{aligned}
\text{Precision}_{\text{overall}} &= \frac{\sum TP}{\sum (TP + FP)} \\
\text{Recall}_{\text{overall}} &= \frac{\sum TP}{\sum (TP + FN)} \\
\text{F1}_{\text{overall}} &= \frac{2 \times TP \times Recall}{Precision + Recall} \\
\text{Accuracy}_{\text{overall}} &= \frac{\sum (TP + TN)}{\sum (TP + FP + FN + TN)}
\end{aligned}
$$

Table 5.14: Overall Evaluation Metrics for 50 Repositories

Metric	Score
Precision	0.89
Recall	0.87
F1	0.95
Accuracy	0.98

5.4 Triangulation of the Findings

To synthesize insights across interviews, surveys, and repository analysis, we employed a triangulation framework grounded in statistical effect-size comparisons. This approach allowed us to identify where practitioner perceptions (RQ1), self-reported practices (RQ2), and observable repository signals (RQ3) converged to reinforce common patterns and where they diverged to reveal gaps between reported and observable practice.

This categorization represents the current practitioner ecosystem. The interviews.Provide the initial perspective, revealing which objects and tools are most significant in practice. These influenced our survey questionnaire, which recorded usage frequency across a wider sample. Finally, through repository mining, we examined how these constructs manifest in open-source projects, for example by scanning YAML files. This triangulation guarantees that the results we present are based on both practitioner views and observable repository indicators. The subsequent Results section presents pertinent statistics derived from these Kubernetes objects and tools, illustrating their practical application. These structures are integral to the Kubernetes ecosystem and form the basis for addressing our five research questions (RQ1–RQ4).

Chapter 6

Discussion

This study set out to understand how Kubernetes is adopted and practiced in the industry, and in doing so, it makes several contributions that go beyond prior work. By combining interviews, surveys, and repository analysis, we not only confirmed certain established patterns but also surfaced areas where perception and practice diverge, where community discourse does not reflect reality, and where important gaps remain in research and tooling. Below, we highlight the most significant novelties and show how they directly answer our five research questions (RQ1–RQ5).

6.1 Reading the Signals: How Frequency Counts Explain Ecosystem Practices

The frequency with which particular Kubernetes elements appear in repositories offers a powerful signal of ecosystem practices. Rather than being just numerical tallies, these counts capture how central certain constructs have become in real-world usage. High-frequency elements such as Deployments, Services, and ConfigMaps emerge as foundational building blocks, reflecting their maturity and indispensability in production environments. By contrast, low-frequency elements, including advanced scheduler rules or the Vertical Pod Autoscaler, point to features that remain niche, experimental, or aspirational.

Importantly, these distributions not only reveal adoption patterns but also provide a way to compare perception with practice. When practitioners frequently highlight an element in interviews or surveys but repositories show limited signals, the gap signals that adoption is uneven or context-dependent. Conversely, strong presence in both discourse and repositories underscores alignment and the consolidation of best practices. In this way, frequency counts function as more than descriptive statistics: they are indicators of ecosystem maturity, stability, and direction. Established elements with widespread presence reflect mainstream practices, while low-frequency yet high-discussion elements highlight areas where practitioner intent outpaces implementation.

Taken together, these signals ground our understanding of Kubernetes in evidence from large-scale repositories, enabling us to move beyond anecdotal practitioner

narratives. They allow us to situate adoption within a continuum — from widely entrenched practices to emerging, less adopted features — and to interpret how the ecosystem is evolving in practice.

6.2 Reading Across Sources: Where Kubernetes Practices Align and Diverge

To better situate our findings, we compared Kubernetes practices across three vantage points: practitioner interviews, survey responses, and evidence from 1,429 repositories. This triangulation helps to highlight not only what practitioners perceive as central, but also what open-source traces show in actual use. In doing so, we identify areas of strong agreement, areas where adoption is consistently low, and cases where perceptions and repository signals diverge.

(a) Practices frequent in both. Across all sources, the foundational constructs of Kubernetes stood out as highly consistent. Interview participants often described *Deployments, Pods, and Services* as the everyday building blocks of Kubernetes. The survey reinforced this view, showing very high adoption rates—over 90% for Pods, Deployments, and Services. Repository data provided further confirmation: *Deployments (62.9%)* and *Services (60.0%)* appeared in the majority of projects. Configuration and security primitives followed a similar pattern: both *ConfigMaps* and *Secrets* were frequently cited in interviews and survey responses, and their repository presence (38.0% and 29.9%, respectively) indicates they are embedded in routine practice.

Workloads showed a similar alignment. Running *web applications* on Kubernetes was nearly universal, appearing in 93.8% of survey responses and over half (56.7%) of repositories. On the tooling side, *Helm* and *Terraform* emerged as standard practices across both industry narratives and repositories. Helm was used by 65.6% of survey respondents and appeared in 29.7% of repositories, while Terraform showed 62.5% in the survey and 6.7% in repositories. Together, these results reflect a clear backbone of Kubernetes usage - core objects, web workloads, and mainstream IaC tools - that are deeply embedded across contexts.

(b) Practices less frequent in both. Some features remained consistently peripheral across data sources. *Batch jobs and data pipelines* were only occasionally mentioned in interviews, reported by less than one-third of survey respondents (31.3% and 40.6%), and were rarely present in repositories (13.6% and 0.3%). Likewise, advanced scheduling features such as
pod affinity/anti-affinity and *taints/tolerations* attracted little attention in interviews, had modest survey adoption (22–48%), and appeared only weakly in repositories (below 12%).

A similar story emerges with GitOps and IaC tools beyond Helm and Terraform. While *ArgoCD* and *Flux* were acknowledged in interviews and adopted by about 31% of survey participants, their repository presence remained very small (5% or less).

These converging signals suggest that such practices are still either situational—useful in specific domains like HPC or data engineering or are in earlier stages of adoption.

(c) Practices frequent in one but less frequent in the other. The most interesting patterns arise in cases of divergence. *Databases (StatefulSets)* illustrate this clearly. Interviewees and survey respondents (34.4%) often discussed StatefulSets as a necessary construct for running persistent workloads such as databases. Yet in repositories, StatefulSets appeared in only 15.3% of projects. This gap suggests that while databases are central in enterprise discourse, open-source projects may avoid stateful complexity or rely on managed database services instead.

Another divergence comes from *Operators and Controllers*. These elements appeared strongly in repository evidence (Operators 27.8%, Controllers 25.3%) but were less visible in interview or survey data. Their presence in repositories highlights their growing role in automation and lifecycle management, even if they are not always top of mind for practitioners describing their daily workflows.

Observability tools reveal a further gap. In the survey, practitioners highlighted both commercial and open-source tools, mentioning *Datadog* alongside *Prometheus*. Repository evidence, however, painted a different picture: *Prometheus (27.6%)* and *Grafana (14%)* dominated, while Datadog appeared in only 3.5% of projects. This points to a contrast between enterprise environments, where paid tools are common, and the open-source ecosystem, which favors community-supported solutions.

Overall taken together, the triangulation shows a layered picture. At the center, core constructs (Deployments, Pods, Services, ConfigMaps, web applications, Helm) shows strong alignment across all sources, reflecting their foundational role. At the periphery, features like batch jobs, pipelines, scheduling rules, and GitOps tools remain consistently less frequent, suggesting niche or emerging use. In between, divergences emerge in areas such as StatefulSets, Operators, and observability tools, where enterprise realities differ from open-source signals. These gaps may reflect cost considerations, organizational maturity, or the natural distinction between production-grade systems and open-source experimentation.

6.3 The Gaps Between Perceptions and Practice

Another novelty of this study lies in uncovering the gap between what practitioners say and what they actually do. This contradiction became visible only when we triangulated interviews, surveys, and repository evidence. A few significant gaps are listed below:

The Persistence Paradox: Databases on Kubernetes

Several interviewees stressed that "databases don't belong on Kubernetes," echoing conventional wisdom and earlier research that positioned Kubernetes as most suitable for stateless services [72], [73], [1]. This narrative has been repeated in best-practice guides and community forums. However, our survey results showed that

a notable proportion of respondents reported running stateful workloads. Taken together, the three data sources suggest that while engineers recognize the risks of persistence in Kubernetes, they nonetheless adopt it when operational demands outweigh prescriptive guidance.

The Scheduling Gap: Best Practices vs. Everyday Shortcuts

Practitioners frequently described scheduling features such as Pod Anti-Affinity and Node Affinity as critical for resilience, ensuring that replicas are distributed across nodes and workloads land on appropriate hardware. Some even called these rules "basic reliability hygiene." Yet, some admitted they seldom apply them in real projects. As one engineer reflected: "We know anti-affinity avoids single points of failure, but most of the time we just let the default scheduler do its job" [I6].This candid acknowledgment reveals a tension between knowing the right thing to do and taking the fastest path forward, often due to time pressure or configuration complexity. Repository evidence supports this gap. Pod Anti-Affinity appeared in only 3.6% of projects and Node Affinity in 5.6%, underscoring that while intelligent scheduling is celebrated in principle, it remains uncommon in practice.

The Networking Shortcut: Ingress in Principle, LoadBalancers in Practice

Interviewees frequently highlighted that networking is one of the most difficult parts of Kubernetes. Many pointed to Ingress as the right way to handle secure and scalable service exposure, often framing them as a standard best practice. Yet, in practice, they admitted to falling back on simpler options. Several confessed to using LoadBalancer Services for applications knowing it drives up costs but appreciating the speed and ease. One interviewee's said, "Spinning up a LoadBalancer is quick" [I13], this reveals a gap. While Ingress is promoted as the ideal, everyday work often relies on quick fixes. Repository data reflects this tension as well, with only 27% of projects containing Ingress manifests, and limited evidence of advanced networking patterns.

The Security Confidence Gap: Trusted in Principle, Fragile in Practice

Security was described by practitioners as both a strength and a vulnerability of Kubernetes. One interviewee confidently remarked that "Kubernetes is pretty secure unless someone inside the organization wants to cause harm," reflecting a belief that the platform is inherently robust. In contrast, another participant, a PhD candidate, who had previous organizational working experience, conducted a few tests for his project on Kubernetes security which pointed out that "there are many cases where the security fails," often due to gaps in enforcement.

They emphasized that network policies are not consistently applied, and PodSecurityStandards are not always configured end-to-end, leaving room for misconfigurations. This creates a tension between the perception of Kubernetes as secure-by-default and the operational reality of fragile setups. Repository evidence reinforces this point. Traces of PodSecurityPolicies and newer security controls appeared only

sporadically, suggesting that while security is praised in discourse, it remains unevenly implemented in practice.

6.4 Challenges in Practice

Our findings reveal five key challenges that practitioners repeatedly face when working with Kubernetes:

1. Upgrades and operational overhead: Upgrades emerged as one of the most disruptive aspects of Kubernetes. Engineers managing multiple clusters described upgrades as "a major pain point," particularly when zero downtime was required. Survey data supports this view, with most respondents rating upgrades as moderately to highly challenging. Beyond versioning, day-to-day operations frequently involve resolving CrashLoopBackOff errors, DNS issues, or external IP failures, underscoring the hands-on burden of maintaining clusters.

2. Security and compliance gaps: Security was identified as a recurring pain, especially around RBAC enforcement, namespace isolation, and ingress configuration. One fintech practitioner cautioned that "if one service is exposed then the full cluster can be hacked." Survey responses align with this concern: network security and API exposure were rated as the most challenging areas, followed by RBAC integration. Compliance requirements in regulated industries further compound these pressures, with financial and healthcare teams stressing strict adherence to local standards.

3. Storage and persistence issues: Persistent volumes and stateful workloads continue to pose difficulties. Several engineers emphasized that "storage is the 1 pain," citing remounting failures, durability risks after outages, and limited tooling for troubleshooting. Even with solutions like Longhorn, problems persisted. Especially when operating under compliance-driven constraints for regulated clients.

4. Steep learning curve and skills gap: Across interviews, practitioners emphasized Kubernetes' steep learning curve. Understanding configuration details, scheduling features, and debugging practices was described as "overwhelming." One participant observed that "only a few people can work with Kubernetes well," reflecting a shortage of professionals who bridge both DevOps and software engineering. Survey data also identified skills and knowledge gaps as a common adoption barrier, especially for smaller teams.

5. Configuration complexity and desired improvements: Beyond core functionality, practitioners expressed a need for simpler and more automated processes. Requests included easier installation, clearer default settings, and more built-in support for networking and storage.

6.5 Limitations of the Study

This study examined Kubernetes adoption and practices through a mixed-methods approach, combining 22 semi-structured interviews, 31 survey responses, and an analysis of 1,429 public repositories. By triangulating practitioner narratives, self-reported practices, and repository-level evidence, we developed a holistic view of Kubernetes usage across contexts.At the same time, interviews and surveys come with natural constraints. While interviews enabled us to capture rich, first-hand perspectives from practitioners, the sample size (22 participants) limits broad generalization across the diverse Kubernetes ecosystem. Similarly, the survey responses (31 in total) offered valuable insights but may reflect some degree of self-selection bias, as participants who are more engaged with Kubernetes were more likely to respond. These methods capture perceptions and experiences, but they cannot on their own fully represent the breadth of practices across all industries and geographies.

It is important to acknowledge that public repositories do not always mirror production-grade systems. Many reflect prototypes, small-scale projects, or experimental efforts where advanced features such as scheduling, CI/CD pipelines, or strict security enforcement are not a priority. Yet, these repositories remain valuable because they capture the practices developers deliberately make visible, share, and reuse signals that shape the Kubernetes ecosystem in meaningful ways. By triangulating repository evidence with practitioner interviews and survey responses, we mitigate the limitations of any single source. In fact, the mismatches themselves become insightful, they reveal the difference between practices that are widely promoted in production discourse and those that are in open-source projects, highlighting areas where adoption is still emerging or uneven.

Chapter 7

Conclusion and Future Work

This study examined Kubernetes adoption and practices in the industry using a mixed-methods approach, incorporating 22 semi-structured interviews, 31 survey responses, and the investigation of 1,429 public repositories. Through the triangulation of practitioner narratives, self-reported practices, and open-source indicators, we constructed a thorough picture of the motivations behind businesses' adoption of Kubernetes, the methods of its integration into workflows, and the problems encountered in its maintenance. The study methodically examined five research issues, encompassing adoption factors, engineering methods, maintenance indicators, alignment between perceptions and repository data, and the operational challenges associated with Kubernetes.

Our findings underscore multiple reoccurring patterns. Kubernetes is regarded as the prevailing standard for the deployment and management of services, with Deployments and Services constituting the foundation of most systems. Infrastructure-as-Code, specifically Helm and Terraform, together with observability tools such as Prometheus, Grafana, and Datadog, are essential to practice, although their application differs among teams and stages of maturity. Repositories validate these patterns, indicating robust signs of Infrastructure as Code (IaC), monitoring, and object utilization. Simultaneously, practitioners exhibited nuanced perceptions: although many advocated specific approaches as best practices, others saw that these practices were challenging to maintain or not consistently implemented within their organizational contexts.

The analysis affirms Kubernetes' significance in contemporary cloud-native engineering while highlighting areas of alignment and slight discrepancies between views, documented practices, and open-source data.

Future research should extend repository mining beyond the curated GitHub dataset to include other ecosystems such as GitLab, Bitbucket, and private enterprise repositories, thereby improving the generalizability of observed trends. Moreover, incorporating Generative AI (GAI) and Large Language Models (LLMs) into repository analysis offers promising opportunities. LLMs could be leveraged to automatically classify Kubernetes configurations, detect emergent usage patterns, and even infer organizational maturity levels from commit histories and configuration practices. Finally, longitudinal studies that combine AI-assisted mining with field observations

could shed light on how organizations evolve in handling upgrades, multi-cluster deployments, and resource optimization.

Reference

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016. DOI: 10.1145/2890784.

[2] Cloud Native Computing Foundation, *Cncf annual survey 2022*, https://www.cncf.io/reports/cncf-annual-survey-2022/, Accessed: 2025-09-11, 2022.

[3] M. Villamizar, O. García, H. Castro, *et al.*, "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," in *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, 2015, pp. 584–591. DOI: 10.1109/CLOUD.2015.84.

[4] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *Journal of Systems and Software*, vol. 163, p. 110 542, 2020. DOI: 10.1016/j.jss.2020.110542.

[5] N. Kratzke, "A brief history of cloud application architectures: From monoliths via n-tier to microservices and their impact on cloud-native platforms like kubernetes," *Journal of Cloud Computing*, vol. 7, no. 1, pp. 1–17, 2018. DOI: 10.1186/s13677-018-0112-6.

[6] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017. DOI: 10.1109/ACCESS.2017.2685629.

[7] R. Peinl, F. Holzschuher, and B. Pfitzinger, "Why do enterprises struggle with kubernetes? an empirical study," in *IEEE International Conference on Cloud Engineering (IC2E)*, 2019, pp. 192–198. DOI: 10.1109/IC2E.2019.00035.

[8] S. I. Shamim, J. A. Gibson, P. Morrison, and A. Rahman, "A multi-vocal literature review of kubernetes," *Empirical Software Engineering*, 2022, arXiv preprint arXiv:2211.07032.

[9] G. Turina *et al.*, "Predicting resource consumption of kubernetes via formal modeling," *Journal of Systems and Software*, 2023, PDF available; contains background on Kubernetes architecture.

[10] M. Imran, V. Kuznetsov, L. Marcella, K. M. Dziedziniewicz-Wojcik, A. Pfeiffer, and P. Paparrigopoulos, "Migration of cmsweb cluster at cern to kubernetes," *arXiv preprint arXiv:2102.12751*, 2021.

[11] Kubernetes Authors, *Kubernetes documentation: Pods*, https://kubernetes.io/docs/concepts/workloads/pods/, Accessed: 2025-09-11, 2025.

[12] Kubernetes Authors, *Kubernetes documentation: Deployments*, https://kubernetes.io/docs/concepts/workloads/controllers/deployment/, Accessed: 2025-09-11, 2025.

[13] Kubernetes Authors, *Kubernetes documentation: Replicaset*, https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/, Accessed: 2025-09-11, 2025.

[14] Kubernetes Authors, *Kubernetes documentation: Statefulset*, https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/, Accessed: 2025-09-11, 2025.

[15] Kubernetes Authors, *Kubernetes documentation: Daemonset*, https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/, Accessed: 2025-09-11, 2025.

[16] Kubernetes Authors, *Kubernetes documentation: Cronjob*, https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/, Accessed: 2025-09-11, 2025.

[17] Kubernetes Authors, *Kubernetes documentation: Services*, https://kubernetes.io/docs/concepts/services-networking/service/, Accessed: 2025-09-11, 2025.

[18] Kubernetes Authors, *Kubernetes documentation: Ingress*, https://kubernetes.io/docs/concepts/services-networking/ingress/, Accessed: 2025-09-11, 2025.

[19] Kubernetes Authors, *Kubernetes documentation: Namespaces*, https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/, Accessed: 2025-09-11, 2025.

[20] Kubernetes Authors, *Kubernetes documentation: Horizontal pod autoscaler*, https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, Accessed: 2025-09-11, 2025.

[21] Kubernetes Authors, *Kubernetes documentation: Persistent volumes*, https://kubernetes.io/docs/concepts/storage/persistent-volumes/, Accessed: 2025-09-11, 2025.

[22] Kubernetes Authors, *Kubernetes documentation: Persistent volume claims*, https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistent\volumeclaims, Accessed: 2025-09-11, 2025.

[23] Kubernetes Authors, *Kubernetes documentation: Configmaps*, https://kubernetes.io/docs/concepts/configuration/configmap/, Accessed: 2025-09-11, 2025.

[24] Kubernetes Authors, *Kubernetes documentation: Secrets*, https://kubernetes.io/docs/concepts/configuration/secret/, Accessed: 2025-09-11, 2025.

[25] Helm Authors, *Helm documentation*, https://helm.sh/docs/, Accessed: 2025-09-11, 2025.

[26] Kubernetes Authors, *Kubernetes documentation: Operators*, https://kubernetes.io/docs/concepts/extend-kubernetes/operator/, Accessed: 2025-09-11, 2025.

[27] HashiCorp, *Terraform documentation*, https://developer.hashicorp.com/terraform/docs, Accessed: 2025-09-11, 2025.

[28] Argo Project, *Argo cd documentation*, https://argo-cd.readthedocs.io/, Accessed: 2025-09-11, 2025.

[29] Prometheus Authors, *Prometheus documentation*, https://prometheus.io/docs/, Accessed: 2025-09-11, 2025.

[30] Grafana Labs, *Grafana documentation*, https://grafana.com/docs/, Accessed: 2025-09-11, 2025.

[31] Google Cloud, *Google kubernetes engine documentation*, https://cloud.google.com/kubernetes-engine/docs, Accessed: 2025-09-11, 2025.

[32] Amazon Web Services, *Amazon elastic kubernetes service documentation*, https://docs.aws.amazon.com/eks/, Accessed: 2025-09-11, 2025.

[33] Microsoft Azure, *Azure kubernetes service documentation*, https://learn.microsoft.com/en-us/azure/aks/, Accessed: 2025-09-11, 2025.

[34] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, "Challenges of adopting continuous integration in the industry: A survey study," *Information and Software Technology*, vol. 123, p. 106 294, 2020. DOI: 10.1016/j.infsof.2020.106294.

[35] A. Rahman, R. Mahdavi-hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Information and Software Technology*, vol. 108, Dec. 2018. DOI: 10.1016/j.infsof.2018.12.004.

[36] A. Wiedemann, N. Forsgren, M. Wiesche, H. Gewald, and H. Krcmar, "Research for practice: The devops phenomenon," *Commun. ACM*, vol. 62, no. 8, pp. 44–49, Jul. 2019, ISSN: 0001-0782. DOI: 10.1145/3331138. [Online]. Available: https://doi.org/10.1145/3331138.

[37] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Assuring the evolvability of microservices: Insights into industry practices and challenges," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Sep. 2019, pp. 546–556. DOI: 10.1109/icsme.2019.00089. [Online]. Available: http://dx.doi.org/10.1109/ICSME.2019.00089.

[38] Y. Jiang and B. Adams, "Co-evolution of infrastructure and source code: An empirical study," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15, Florence, Italy: IEEE Press, 2015, pp. 45–55, ISBN: 9780769555942.

[39] A. Zerouali, R. Opdebeeck, and C. De Roover, "Helm charts for kubernetes applications: Evolution, outdatedness and security risks," in *Proceedings of the 20th IEEE/ACM International Conference on Mining Software Repositories (MSR 2023)*, IEEE, 2023, pp. 523–533. DOI: 10.1109/MSR59073.2023.00078.

[40] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, *Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices*, 2020. arXiv: 2006.15275 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2006.15275.

[41] E. K. Adejumo and B. Johnson, *An empirical validation of open source repository stability metrics*, 2025. arXiv: 2508.01358 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2508.01358.

[42] M. Koskinen, T. Mikkonen, A. Knutas, and C. E. Cuesta, "Containers in software development: A systematic mapping study," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 17, e5668, 2019. DOI: 10.1002/cpe.5668.

[43] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 17, e5668, 2020. DOI: 10.1002/cpe.5668.

[44] D. Tyresson, "Security implications for docker container environments," M.S. thesis, University of Skövde, Sweden, 2020. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1463573/FULLTEXT01.pdf.

[45] X. Li *et al.*, "A systematic literature review on microservices: Quality attributes and architectures," *Journal of Systems and Software*, vol. 170, p. 110 798, 2020. DOI: 10.1016/j.jss.2020.110798.

[46] F. Erich, C. Amrit, and M. Daneva, "A qualitative study of devops usage in practice," *Journal of Software: Evolution and Process*, vol. 29, no. 6, e1885, 2017. DOI: 10.1002/smr.1885.

[47] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 33, no. 3, pp. 24–35, 2016. DOI: 10.1109/MS.2016.62.

[48] I. Kumara, M. Garriga, A. U. Romeu, *et al.*, "The do's and don'ts of infrastructure code: A systematic gray literature review," *Information and Software Technology*, vol. 137, p. 106 593, 2021. DOI: 10.1016/j.infsof.2021.106593.

[49] J. Soldani,
bibinitperiod Tamburri, and W.-J. {van den Heuvel}, "The pains and gains of microservices: A systematic grey literature review," English, *Journal of Systems and Software*, vol. 146, pp. 215–232, Dec. 2018, ISSN: 0164-1212. DOI: 10.1016/j.jss.2018.09.082.

[50] X. Zhou, S. Li, L. Cao, *et al.*, "Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry," *J. Syst. Softw.*, vol. 195, no. C, Jan. 2023, ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111521. [Online]. Available: https://doi.org/10.1016/j.jss.2022.111521.

[51] N. Lazarev, S. Xiang, N. Adit, Z. Zhang, and C. Delimitrou, "Dagger: Efficient and fast rpcs in cloud microservices with near-memory reconfigurable nics," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21, Virtual, USA: Association for Computing Machinery, 2021, pp. 36–51, ISBN: 9781450383172. DOI: 10.1145/3445814.3446696. [Online]. Available: https://doi.org/10.1145/3445814.3446696.

[52] A. Rahman, E. Farhana, and L. Williams, "The 'as code' activities: Development anti-patterns for infrastructure as code," *Empirical Softw. Engg.*, vol. 25, no. 5, pp. 3430–3467, Sep. 2020, ISSN: 1382-3256. DOI: 10.1007/s10664-020-09841-8. [Online]. Available: https://doi.org/10.1007/s10664-020-09841-8.

[53] A. Rahman, S. I. Shamim, R. Ali, and L. Williams, "Security misconfigurations in kubernetes manifests: An empirical study," *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 4, pp. 1–28, 2020. DOI: 10.1145/3387905.

[54] A. Zerouali, R. Opdebeeck, and C. D. Roover, "Helm charts for kubernetes applications: Evolution, outdatedness and security risks," in *Proceedings of the IEEE/ACM International Conference on Mining Software Repositories (MSR)*, 2023, pp. 523–533. DOI: 10.1109/MSR59073.2023.00078.

[55] F. Minna, F. Massacci, and K. Tuma, "Analyzing and mitigating the security misconfigurations of helm charts with large language models," in *Proceedings of the IEEE/ACM International Conference on Mining Software Repositories (MSR)*, 2024, pp. 401–412. DOI: 10.1145/3643991.3645112.

[56] H. Zhang, A. Zerouali, and T. Mens, "On the evolution and maintenance of helm charts for kubernetes," in *Proceedings of the 18th IEEE/ACM International Conference on Mining Software Repositories (MSR)*, 2021, pp. 425–436. DOI: 10.1109/MSR52588.2021.00057.

[57] V. Nguyen, K.-D. Lange, and J. Brustoloni, "Benchmarking kubernetes, docker swarm and apache mesos as container orchestrators," in *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2019, pp. 251–260. DOI: 10.1109/CloudCom.2019.00045.

[58] J. Kim, S. Seo, and B. Tak, "Performance analysis of kubernetes auto-scaling for cloud-native applications," *Future Generation Computer Systems*, vol. 109, pp. 24–34, 2020. DOI: 10.1016/j.future.2020.02.018.

[59] L. Rossi, A. Bartolini, and L. Benini, "Simulation-based performance evaluation of large-scale kubernetes clusters," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 198–207. DOI: 10.1109/ISPASS48437.2020.00029.

[60] A. Bauer, D. Scheinert, and H. Karl, "Custom scheduling for kubernetes: Opportunities and trade-offs," in *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2019, pp. 78–90. DOI: 10.1145/3357223.3362714.

[61] S. Bhat, J. Rilling, and H. Jamjoom, "Fair scheduling in kubernetes multitenant clusters," in *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*, 2021, pp. 12–21. DOI: 10.1109/IC2E52221.2021.00009.

[62] D. Gruner, S. Hack, and C. Lengauer, "Numa-aware scheduling in kubernetes: A practical evaluation," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 42–55, 2020. DOI: 10.1016/j.jpdc.2020.06.007.

[63] M. Al-Doghman, A. Ibrahim, and S. Sakr, "Anomaly detection in kubernetes clusters using prometheus metrics," in *Proceedings of the IEEE International Conference on Big Data (BigData)*, 2021, pp. 2345–2354. DOI: 10.1109/BigData52589.2021.9671809.

[64] M. Boukhechem, S. Hammoudi, and N. Sahli, "Log aggregation and observability challenges in kubernetes," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1–19, 2022. DOI: 10.1186/s13677-022-00318-5.

[65] G. Casale, M. Ciavotta, and E. Ozturk, "Chaos engineering in kubernetes clusters: Fault injection and resilience analysis," in *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, 2021, pp. 100–109. DOI: 10.1109/CLOUD53861.2021.00022.

[66] R. Peinl, F. Holzschuher, and B. Pfitzinger, "Why do enterprises struggle with kubernetes? an empirical study," in *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*, 2019, pp. 192–198. DOI: 10.1109/IC2E.2019.00035.

[67] C. Lyu, Z. M. Jiang, and A. E. Hassan, "On the evolution and maintenance of kubernetes operators," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2022, pp. 412–423. DOI: 10.1109/ICSME55016.2022.00052.

[68] D. Almeida, A. Hora, and D. A. da Costa, "An empirical study on kubernetes configuration smells in open-source systems," in *Proceedings of the IEEE/ACM International Conference on Mining Software Repositories (MSR)*, 2021, pp. 480–491. DOI: 10.1109/MSR52588.2021.00063.

[69] J. Cito, P. Leitner, and H. Gall, "Empirical insights into kubernetes adoption and cloud-native application development," in *Proceedings of the International Conference on Cloud Engineering (IC2E)*, 2020, pp. 192–203. DOI: 10.1109/IC2E48712.2020.00025.

[70] N. Kratzke, "A brief history of cloud application architectures: From monoliths via n-tier to microservices and their impact on cloud-native platforms like kubernetes," *Journal of Cloud Computing*, vol. 7, no. 1, pp. 1–17, 2018. DOI: 10.1186/s13677-018-0112-6.

[71] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006. DOI: 10.1191/1478088706qp063oa.

[72] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014, ISSN: 1075-3583.

[73] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. May 2015, ISBN: 0134049845.

[74] A. Rahman, *Kubesec: Security analysis of kubernetes artifacts (scanner.py)*, https://github.com/paser-group/KubeSec/blob/master/scanner.py, [Online; accessed: 01-Nov-2021], 2020.

[75] A. Rahman, *Kubesec: Parser script for security scanning (parser.py)*, https://github.com/paser-group/KubeSec/blob/master/parser.py, [Online; accessed: 01-Nov-2021], 2020.

[76] J. Chen, W. Shang, and A. E. Hassan, "A survey on AIOps for software systems," *ACM Computing Surveys*, vol. 54, no. 10, pp. 1–36, 2022. DOI: 10.1145/3464420.

Appendices

Appendix A

Interview Questionnaire

1. What is your position in your company?

2. Have you used Kubernetes in your professional work? Which components of Kubernetes, e.g., operators and add-ons, do you work with? What are the use cases of Kubernetes in your organization?

3. How long have you been using Kubernetes?

4. What specific Kubernetes platform, e.g., Amazon EKS, Azure AKE, and GKE does your organization use?

5. How has Kubernetes been implemented in the organizations you have worked for? Can you elaborate?

6. What types of applications do you typically deploy using Kubernetes?

7. Are there any specific tools or extensions you use with Kubernetes?

8. Do you also work with other container orchestration tools or technologies?

9. Do you use any AI tools to enhance your Kubernetes management or operations?

10. What do you like and dislike about Kubernetes? Can you please elaborate?

11. What challenges have you faced while using Kubernetes, and how do you solve these challenges? Are these challenges unique to the use cases for your organization?

12. Do you deal with compliance or regulations when managing Kubernetes clusters?

13. Have you experienced any incidents in your organization related to Kubernetes? Can you please elaborate?

14. How do you manage configurations for your Kubernetes deployments?

15. How hard or easy is it to find onboard professionals with decent Kubernetes

skills?

16. What are your perspectives on the environmental impacts of Kubernetes-based infrastructure provisioning?

17. What features would you like to see improved in future Kubernetes releases?

18. Is there anything else you would like to add regarding Kubernetes practices in the context of Bangladesh and other developing countries?

Appendix B

Interviewee Codes

For confidentiality, all interview participants have been anonymized. They are denoted as I1, I2, ..., I22 throughout the thesis. The table below provides the assigned codes along with a brief description of each participant's role and organization type. Company names are omitted to preserve anonymity.

Table B.1: Interviewee Codes and Anonymized Roles

Code	Role / Organization (anonymized)
I1	Senior DevOps Engineer, SaaS Company
I2	Software Architect, IT Services Firm
I3	Software Engineer, E-commerce Company
I4	Site Reliability Engineer, Fintech
I5	Cloud Infrastructure Lead, Fintech Company
I6	Software Engineer, Global Tech Enterprise
I7	DevOps Specialist, IT
I8	Solutions Architect, Cloud Provider
I9	Senior Engineer, AI/ML Startup
I10	Systems Engineer, Government Agency
I11	Lead DevOps Engineer, IT
I12	Infrastructure Engineer, Large Bank
I13	Kubernetes Administrator, Retail Sector
I14	Platform Reliability Engineer, SaaS Provider
I15	DevOps Lead, Software Firm
I16	Senior Software Engineer, Tech Startup
I17	Back-end Developer A Ride-Hailing Company
I18	Systems Engineer, Software Firm
I19	Solution Architect, Transportation Tech Firm
I20	Senior DevOps Engineer, Cloud-Native Vendor
I21	Platform Engineer, Manufacturing Sector
I22	Senior Engineer, Research Organization

Appendix C

Survey Questionnaire

Instructions. Please answer all required questions. Unless otherwise noted, select one option per row.
Scales used:

- Challenge scale: (a) Not Challenging (b) Slightly Challenging (c) Neutral (d) Challenging (e) Very Challenging
- Frequency scale: (a) Never (b) Rarely (c) Sometimes (d) Often (e) Always
- Usage scale: (a) Not at all (b) Not sure (c) Rarely (d) Somewhat frequently (e) Frequently

Q1. Which country do you work in? (Required)

Q2. How many years of experience do you have working with Kubernetes? (Required)
(a) Less than 1 year
(b) 1–2 years
(c) 3–4 years
(d) 5–7 years
(e) 8+ years

Q3. What is your current role? (Required)
(a) Developer
(b) DevOps Engineer
(c) Site Reliability Engineer (SRE)
(d) System Administrator
(e) Manager / Tech Lead
(f) Architect
(g) Other: _____

Q4. To what extent do Kubernetes upgrades require manual effort and cause challenges in your team? (Required)
(a) Not at all (1)
(b) 2

(c) 3
(d) 4
(e) Extremely challenging (5)

Q5. Which of these tools do you use for managing Kubernetes infrastructure? (Select all that apply) (Required)
(a) Terraform
(b) Helm
(c) Ansible
(d) GitOps (e.g., Argo CD, Flux)
(e) Other: _____

Q6. What kind of clusters do you primarily use? (Required)
(a) Self-managed (on-premises)
(b) Cloud-managed (e.g., EKS, GKE, AKS)
(c) Hybrid
(d) Other: _____

Q7. Which workloads do you run on Kubernetes? (Select all that apply) (Required)
(a) Databases / Stateful services (StatefulSets)
(b) Web applications / APIs
(c) Data pipelines / Streaming
(d) Batch jobs / CronJobs
(e) Other: _____

Q8. Which monitoring and logging tools do you use? (Select all that apply) (Required)
(a) Prometheus
(b) Datadog
(c) Vector
(d) OpenSearch
(e) KubeDB
(f) Lens
(g) Other: _____

Q9. How challenging are the following security tasks in your Kubernetes environment? (Required; select one per row)
Scale: (a) Not Challenging (b) Slightly Challenging (c) Neutral (d) Challenging (e) Very Challenging

	(a)	(b)	(c)	(d)	(e)
Secrets management	☐	☐	☐	☐	☐
API exposure	☐	☐	☐	☐	☐
RBAC integration	☐	☐	☐	☐	☐
Network security	☐	☐	☐	☐	☐
IAM integration	☐	☐	☐	☐	☐

Q10. How often do you face these operational issues? (Required; select one per row)
Scale: (a) Never (b) Rarely (c) Sometimes (d) Often (e) Always

	(a)	(b)	(c)	(d)	(e)
CrashLoopBackOff errors	☐	☐	☐	☐	☐
Network ingress setup	☐	☐	☐	☐	☐
DNS issues	☐	☐	☐	☐	☐
API rate limits	☐	☐	☐	☐	☐
Resource waste (scheduling)	☐	☐	☐	☐	☐
Port-forwarding failure	☐	☐	☐	☐	☐
External IP issues	☐	☐	☐	☐	☐

Q11. To what extent do you use the following Kubernetes pod scheduling features? (select one per row)

Scale: (a) Not at all (b) Not sure (c) Rarely (d) Somewhat frequently (e) Frequently

	(a)	(b)	(c)	(d)	(e)
Node Affinity	☐	☐	☐	☐	☐
Taints and tolerations	☐	☐	☐	☐	☐
Pod Anti-Affinity	☐	☐	☐	☐	☐

Q12. To what extent do you use the following Kubernetes Objects? (Required; select one per row)

Scale: (a) Not at all (b) Not sure (c) Rarely (d) Somewhat frequently (e) Frequently

	(a)	(b)	(c)	(d)	(e)
StatefulSet	☐	☐	☐	☐	☐
Pod	☐	☐	☐	☐	☐
Deployment	☐	☐	☐	☐	☐
ReplicaSet	☐	☐	☐	☐	☐
Persistent Volume (PV)	☐	☐	☐	☐	☐
DaemonSet	☐	☐	☐	☐	☐
Namespace	☐	☐	☐	☐	☐
HorizontalPodAutoscaler	☐	☐	☐	☐	☐
PersistentVolumeClaim (PVC)	☐	☐	☐	☐	☐
ConfigMap	☐	☐	☐	☐	☐

	(a)	(b)	(c)	(d)	(e)
Secret	☐	☐	☐	☐	☐
Service	☐	☐	☐	☐	☐
Ingress	☐	☐	☐	☐	☐
EndpointSlice	☐	☐	☐	☐	☐
CustomResourceDefinition	☐	☐	☐	☐	☐
LimitRange	☐	☐	☐	☐	☐
NetworkPolicy	☐	☐	☐	☐	☐
Autoscaler (HPA/VPA/CA)	☐	☐	☐	☐	☐
PodSecurityPolicy	☐	☐	☐	☐	☐

Q13. To what extent do you use the following Kubernetes Components? (Required; select one per row)

Scale: (a) Not at all (b) Not sure (c) Rarely (d) Somewhat frequently (e) Frequently

	(a)	(b)	(c)	(d)	(e)
Operators	☐	☐	☐	☐	☐
Controllers	☐	☐	☐	☐	☐
Scheduler	☐	☐	☐	☐	☐
Node Planner	☐	☐	☐	☐	☐
Kube-DNS	☐	☐	☐	☐	☐
etcd	☐	☐	☐	☐	☐
kube-scheduler	☐	☐	☐	☐	☐
kube-controller-manager	☐	☐	☐	☐	☐
cloud-controller-manager	☐	☐	☐	☐	☐
kubelet	☐	☐	☐	☐	☐

Q14. What lightweight Kubernetes variants have you used? (Select all that apply) (Required)

(a) K3s
(b) K3d
(c) Minikube
(d) MicroK8s
(e) None

Q15. Any other comments or tips for practitioners? (Required)