International Journal of Sustainability and Innovation in Engineering (IJSIE)
https://www.doi.org/10.56830/IJSIE2024

Vol.2 2024

Helm Charts for Kubernetes Deployments: Simplifying Application Deployment and Management

Naga Murali Krishna Koneru
Hexaware Technologies Inc, USA
Author Email: nagamuralikoneru@gmail.com

Received: 4 June 2024. Accepted: 8 August 2024. Published: 10 October 2024

Abstract

Helm charts function as the principal subject of examination in this study regarding their role in simplifying Kubernetes application deployment and management procedures. Developers find it difficult to deploy Kubernetes applications through container orchestration even though Kubernetes offers scalable platforms and fault tolerance due to its complex nature for those without configuration experience. Through its package manager role, Helm provides reusable charts known as "charts," which solve deployment issues by streamlining the installation process. Developers can automate Kubernetes deployments through Helm charts because these tools create standardized packages that organize application configurations together with their dependencies. An automated Helm package system removes difficult-to-handle YAML configuration files, establishing uniform deployments and minimizing mistakes from human operators. Helm charts give developers three main advantages: user-friendly deployment through charts alongside version management features and rollback protocols and automatic environment deployment configuration capabilities. The paper analyzes Helm's position in Kubernetes management through discussions about its CI/CD pipeline integration and GitOps workflow adoption. The paper examines future developments in Kubernetes and Helm mainly through the lens of enhanced multi-cluster management and serverless architecture growth. The paper supports Helm charts because they enable organizations to achieve faster deployment times, enhanced practice management, and decreased Kubernetes environment complexity. Helm charts within Kubernetes deployment methods produce simplified processes and grant application managers complete control to operate across different infrastructure types.

Keywords: *Helm charts, Kubernetes, Application deployment, Container orchestration, CI/CD pipelines, Version control, GitOps, Dependency management*

1. Introduction

Kubernetes functions as an efficient open-source tool that automates the deployment scaling and management of containerized applications. The container orchestration solution Kubernetes gained prominence after Google created it because it provides exceptional scalability and flexible and robust features. Through Kubernetes, developers can handle complex multi-container setups by letting it run automated operations for service discovery, scheduling, and load-balancing tasks. Organizations choose this solution to construct distributed applications that operate in cloud environments because of its essential features. Kubernetes achieves high availability and fault tolerance through its efficient process of managing failover mechanisms and resource distribution. The multiple advantages of Kubernetes application deployment remain, matching the complexity of operating and managing these deployments. The complex nature of Kubernetes makes it difficult for users because it uses multiple advanced concepts such as pod services deployments and namespaces. The Kubernetes abstractions present challenges for developers who lack experience working with them because they spend excessive time and make mistakes in the Kubernetes ecosystem. Application management efficiency is negatively impacted because deployment requires various configuration files alongside uniform environment needs. The complexity of large-scale deployments increases because of rising microservice numbers and advanced dependencies. Manual configuration attempts by humans without proper tools lead the system toward unmanageability since this method produces deployment delays and human errors.

The Helm package manager functioned as a Kubernetes deployment resolution tool. The Helm design delivers improved Kubernetes deployments by offering essential management tools for Kubernetes applications. Developers can create standardized Kubernetes resources through charts functionality, enabling them to produce ready-to-use deployments and ingress controllers. The application deployment processes delivered by Helm enable YAML configuration file management suppression alongside clean application work while bypassing complicated Kubernetes requirements. The basic components of Helm charts make it possible for users to simplify Kubernetes management. Through Helm charts, you can establish all definition requirements for Kubernetes applications and service deployments. The Helm charts improve deployment efficiency through their bundled configuration elements, which provide necessary template files and required dependencies for effectively executing Kubernetes deployments. Users modify Helm charts by adding values files, which allow them to customize the charts for their environment, development cycle, and production needs. The design modifications in Helm charts lead to decreased operational requirements for humans while maintaining uniform deployment standards across multiple operational settings. Big systems obtain substantial value through Helm charts because these charts deliver exceptional deployment management capabilities to improve efficiency above manual deployment methods. Organizations presently deploy all controlled applications

with low-maintenance demands via Helm charts. The adoption of Kubernetes environments has risen dramatically because Helm lets its users manage versions and execute rollbacks through an update process that provides this convenience. The Kubernetes application becomes simpler through the Helm interface so teams can perform reliable application deployment tasks efficiently.

2. Understanding Kubernetes Deployment

As an open-source technology, Kubernetes serves as an orchestration platform that simplifies containerized applications' launch, adjustment, and control. The system provides an extensive framework for operating distributed systems inside a cloud-native environment. A significant part of Kubernetes deployment functionality consists of automated application deployment processes, which make it essential for building contemporary cloud-native software applications (Konneru, 2021). The process requires defining cluster-based applications and their container deployment, followed by application availability maintenance and scalability adjustments according to demand patterns.

2.1 What is Kubernetes Deployment?

Kubernetes deployment functions as the process defining the application state that operators must maintain consistently. A Kubernetes deployment operates as an advanced abstraction that handles groups of application-similar pods despite pods being the smallest Kubernetes compute components. Kubernetes pods possess either a single container or multiple connected containers that function within combined network domains and mutual storage environments. The main objective of deployments is sustaining running pods at the declared quantity and providing automatic update and rollback mechanisms (Giraldo Moreno, 2020). The implementation of Kubernetes deployments proceeds through multiple sequential operations. During the first stage, users create a deployment configuration through YAML files, which contain specifications of replica count and container images, resource limitations, and environmental components. Kubernetes cluster deployment first requires a YAML file application through the kubectl command-line tool that creates a replica set to run the desired number of pods. Continuous monitoring of the replica set ensures that the application keeps functioning according to the desired configuration. When a pod crashes or gets terminated, the replica set rapidly creates a new pod to protect application availability.

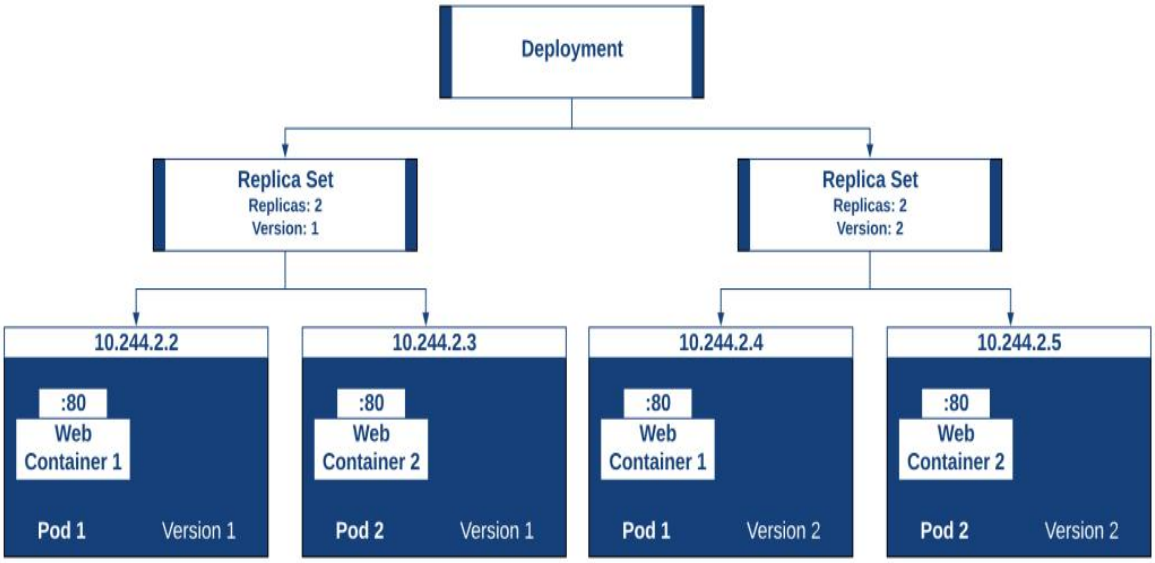

Figure 1: Kubernetes - Deployments

2.2 Components of Kubernetes Deployments

The deployment of Kubernetes requires various essential elements. These include pods, deployments, and replica sets. Kubernetes object pods represent the most fundamental deployment unit since they demand the minimum resources for execution. The pod serves as a single instance of cluster execution where one or more containers run together. Containers hosted in pods share their network space and storage domains, simplifying their management and communication functions. Kubernetes depends on pods as its essential abstraction because they combine container processes with their essential environment elements while guaranteeing uniformity between different systems. A deployment serves as a high-level entry that regulates the lifespan of pods. The management system provides sustained operation of the defined pod replica count. The deployment feature handles rolling updates and rollbacks, allowing developers to update applications without disrupting system availability. The deployment controller conducts continuous state monitoring of pods while making automatic adjustments when the actual state differs from the specified one. Replica Sets function as the mechanisms that keep the predefined pod replication count. The deployment configuration determines the desired state number of pods, which the replica set keeps running at all times. A replica set safeguards high availability by creating a new pod when an existing one gets deleted or crashes (Bekas, 2017).

2.3 Challenges with Traditional Deployment Methods

The deployment capabilities of Kubernetes provide effective processes, yet common deployment techniques create specific obstacles to handling large-scale applications. The main difficulty has emerged from the complex nature of configurations and YAML files. Configuring Kubernetes through detailed YAML files becomes complex and error-prone

because application complexity increases. The configuration files establish different elements to specify pod structures, networking controls, storage definitions, and environment variables. Evolutionary application growth leads to excessive difficulty handling different YAML files used for development, staging, and production environment deployment. Traditional deployment methods encounter two frequent issues involving version control and dependency administration. A containerized application requires a version match between its software application and its container, together with dependent services. Successful operation depends on perfect version matches between application components since version disparities cause system failures and performance drops. The task of version management for image dependencies and configuration files across multiple Kubernetes services becomes complex since different components interconnect with diverse patterns.

Traditional deployment frameworks do not offer strong capabilities to expand services or control configurations throughout various deployment environments (Estrem, 2003). Users need to carry out manual handling tasks to manage application scaling, system configuration, and resource allocation. Human operators who execute these tasks face errors and operational inefficiencies, which create scalability problems. Kubernetes automates several operational duties because its declarative approach allows single-parity scalability and configuration management. The automation capabilities permit rapid propagation of configuration errors whenever such mistakes remain unmanaged. The Kubernetes deployment system offers tools and abstractions to help organizations handle complex, large-scale applications through simplified management procedures. Using these tools becomes challenging for those lacking experience because organizations usually need time to adapt their deployment practices to Kubernetes standards. The Helm software development initiative helps users work with Kubernetes through automation tools that simplify configuration definition and management tasks.

3. Helm: The Package Manager for Kubernetes

3.1 Introduction to Helm

With Helm, as its name suggests, developers acquire a powerful open-source package management solution to handle applications and services deployed in Kubernetes clusters (Block et al, 2022). The Helm Community developed Helm as a solution in 2015, and the Cloud Native Computing Foundation (CNCF) maintains its development today. Helm's fundamental role is to build a system for automatic application deployment and improvement and maintenance tasks for complex systems on Kubernetes clusters. The deployment process becomes smoother through "charts, " Kubernetes resource templates Helm provides. Simplifying configuration and management functions has turned Helm into a vital tool for the Kubernetes ecosystem.

International Journal of Sustainability and Innovation in Engineering
(IJSIE)
https://www.doi.org/10.56830/IJSIE2024
Vol.2 2024

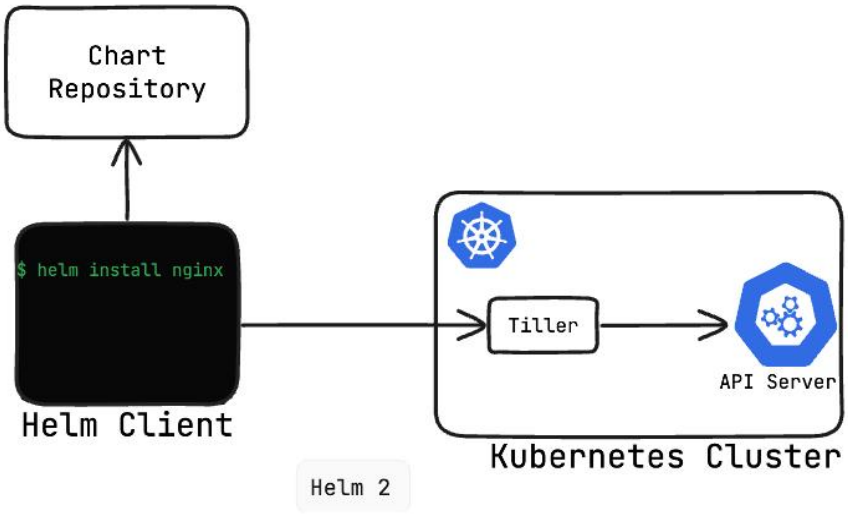

Figure 2: Introduction to Helm

History and Development of Helm

The Helm project originated as a solution for expanding complexity in Kubernetes application deployment. The initial release of Kubernetes operated with basic resource management tools until the rising popularity of Kubernetes demanded improved methods for application deployment. The Helm project emerged to provide a solution for developers needing a standardized method of packaging Kubernetes applications. Helm underwent multiple version updates throughout its development history, adding better usability, functional capability, and security measures. Helm achieved a major advance in 2018 with the release of Helm 3 because it eliminated Tiller as a mandatory component and simplified operations while providing an enhanced user experience.

Helm's Role as a Package Manager for Kubernetes

Helm operates as a Kubernetes package manager that functions like the package managers in other ecosystems, including apt for Debian, yum for CentOS, and npm for Node.js. Through charts, Helm enables developers to manage Kubernetes applications by defining these artifacts, which contain pre-built Kubernetes resources representing applications and their required dependencies. With standard deployment processes, Helm charts simplify the deployment procedure alongside application management for complex applications running within Kubernetes infrastructure. Helm's package manager mechanism enables developers to cut into the manual management of Kubernetes objects while enhancing their ability to deploy and administer applications.

3.2 Key Features of Helm

Helm Charts as Pre-Configured Kubernetes Resources

One of Helm charts is the leading functionality that enhances the application experience. The bundled templates in Helm serve as Kubernetes resource packages, which contain deployment configurations alongside dependency settings for Kubernetes application

deployment. A standard Helm chart contains one or more YAML documents representing Kubernetes resources, including Deployments, Services ConfigMaps and Secrets, and Ingress resources. Users can execute single commands that deploy complex applications through templates while minimizing manual configuration risks.

Templates and Value Files to Manage Configurations Dynamically

Dynamic configuration management operates through Helm charts via templates and value files. The templates within a Chart function as Kubernetes manifest files that incorporate Go templating elements for inserting configuration data dynamically (Bogdan, 2023). Users define most chart values inside separate YAML files called values files. User-defined value files enable Helm to let operators customize application configurations independently from the original chart definition. Different environments can deploy the same Helm chart through separate values files for configuration using a single chart template. Teams using this feature maintain distinct boundaries between code and configuration, just like in modern DevOps approaches.

Versioning and Upgrading Applications Seamlessly

The versioning system in Helm charts enables users to monitor application modifications together with the capability to return to earlier versions of their deployment. The versioning system is vital for organizations that update their applications routinely but need reliable deployment capabilities. Rolling upgrades within Helm provide direct assistance for handling application version updates. Deploying new chart versions through Helm leads to automatic updates of associated Kubernetes resources and low downtime. Helm's tracking function enables users to execute rollbacks to prior application versions when needed. Deployment failures can be efficiently managed through this feature, which enables users to restore past configurations that have proven reliable and stable (Oppenheimer et al, 2003)

Helm Repositories and Sharing Charts within Teams

The core of Helm system functionality rests on Helm repositories that enable users to store and distribute charts among each other. A Helm repository is a platform for keeping packaged charts that users may access through public or private systems. Helm repositories help teams distribute charts among internal members or the wider community to promote coordinated work and standardized Kubernetes deployment methods. Organizations should operate private Helm repositories to retain exclusive charts that target their applications and infrastructure requirements. Helm repositories are an effective platform for chart management to guarantee consistent configurations across environments, as all team members use identical setups.

3.3 How Helm Enhances Kubernetes Deployments

Simplification of Deployment Processes

Helm delivers an easy approach for developers to run their applications within Kubernetes clusters. Before its existence, Helm allowed developers to replace manual

YAML configuration drafting for Kubernetes resources. Helm charts enable automatic deployment through Kubernetes resource templates, reducing user implementation needs. A single Helm install command allows users to deploy application stacks with needed resources, configurations, and dependencies to Kubernetes clusters (Gokhale et al, 2021).

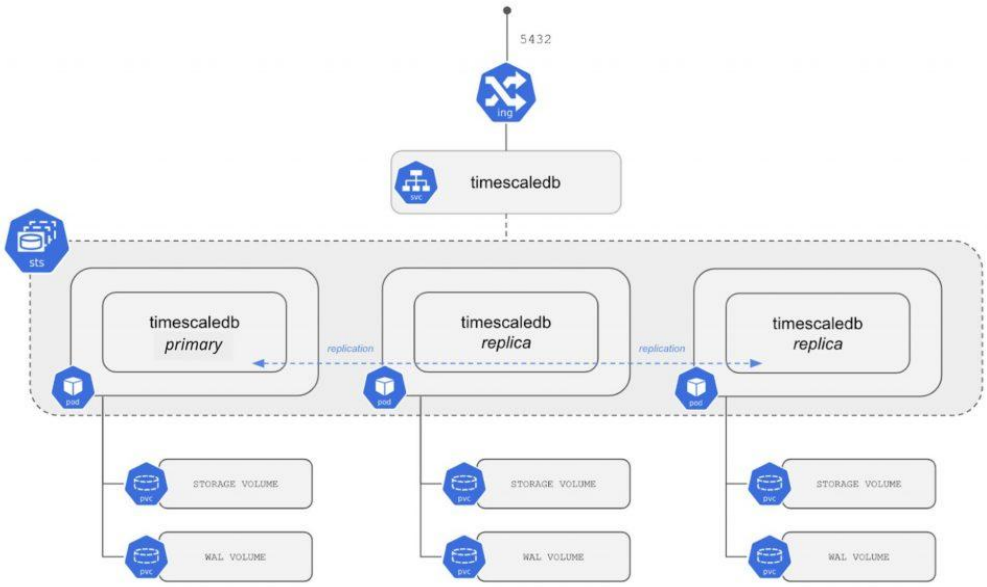

Figure 3: How to Use Helm for Frontend Kubernetes Deployments

Reusability and Maintainability of Helm Charts

The main benefit of Helm charts lies in their capability to facilitate repeated use across multiple projects and environments. Through its modularity and consistency pillars, Helm allows developers to make charts that other teams or clusters can reuse easily. Helm charts enable repeated deployment applications through reusability, which performs two beneficial tasks: time savings and uniform deployment practices. Helm charts receive version control repository management, enabling smooth tracking of configuration file modifications throughout time. Maintainability is essential in quickening development cycles since it lets users quickly change or roll backroll back their configurations to achieve success.

Managing Dependencies with Helm

Via Helm charts, application developers can control dependency relationships because complex applications require several services and components. The Helm dependency management process allows main charts to link with additional charts, so Helm automatically installs necessary dependencies when deploying main charts. Complex multi-tier applications benefit from Helm because it undertakes the responsibility to sequence and configure all needed components for successful deployment. The dependency management features of Helm help developers skip individual component

handling, thus creating more dependable system deployments with fewer errors (Matteson, 2010).

Rollback and Upgrades in Helm Charts

Helm's capability to enable application rolling back and upgrades constitutes an essential feature that boosts Kubernetes deployment effectiveness. Helm enables application update deployment through its system, which tracks changes and distributes them throughout the cluster for consistent application. Helm offers users a simple method to return their applications to prior versions, thus avoiding operational interruptions and downtime after update problems emerge. The upgrade process managed by Helm adopts a method to safely maintain application updates while avoiding both system glitches and configuration issues.

4. What Are Helm Charts?

Helm charts represent an essential framework allowing users to streamline the deployment and manage Kubernetes applications (Spillner, 2019). Helm functions as a Kubernetes-specific open-source package manager whose built-in automation enables the smooth operation of Kubernetes application definition and deployment management. Helm charts provide Kubernetes applications with packaging solutions by packaging deployment configurations alongside templates needed to deploy sophisticated systems into Kubernetes infrastructure. Kubernetes managers and developers need full knowledge of Helm charts to achieve efficient and reusable application management patterns with scalable deployment systems.

4.1 Definition and Structure of Helm Charts

A Helm chart comprises a defined directory structure containing multiple files that develop Kubernetes resources as related components. Developers and administrators use this tool to create deployable application packages containing every dependency in one self-contained unit. The main goal of Helm charts is to aid Kubernetes deployment management through their ability to include resource definitions and configurations together. A Helm chart contains several essential parts organized into specific directories that include: This file presents the chart metadata containing vital information, including name and version, along with text description and dependency specifications. As the Chart's fundamental element, it furnishes Helm with essential details for installation and management tasks (Chavan, 2023). The default configuration values for the Chart exist in the values—yaml file. Users can change the default configuration values in a chart when they install it for individual deployment customization. Depending on the deployment's needs and values, the yaml file enables users to configure deployment parameters through its simple configuration process.

The templates/ directory contains the definitions for Kubernetes resources that form the platform's base. The Go templating language in templates uses values.yaml data points to dynamically create Kubernetes manifests, including Pods, Services, Deployments, and

other resources. Helm charts' strength exists in these templates because they enable developers to generate customizable Kubernetes resources. This directory contains sub-charts on which the current Chart depends. The sub-charts system allows developers to organize their applications through chart reuse capabilities, making managing large systems easier. This directory enables the storage of Custom Resource Definitions (CRDs) for the Kubernetes API extension through new resource definition capabilities. CRDs play an important role when users must utilize custom resources during application deployment.

4.2 Creating Helm Charts

The helm design process demands Kubernetes resource definition in template forms alongside parameter specification. Developing a basic Helm chart starts by initializing a new chart through the Helm create command. The command produces the standard Helm directory configuration and creates Chart.yaml and values.yaml files and insert template files, too. The first step in building a Chart in Yaml involves adding fundamental metadata. This file must contain a unique name, versioning, and purpose description. During this step, the yaml file must be modified to introduce default configuration elements for the Chart. Users who install the Chart can replace these default values. When installing the Chart, users can adjust container image versions, resource request definitions, and replicate numbers according to their operational settings. The third necessary action involves designing Kubernetes resource templates for the Chart during deployment. The Kubernetes resource deployment includes Deployments, while Service deployment serves as the second key resource alongside ConfigMaps and Secrets (Burns & Tracey, 2018). The Go templating syntax enables writing these template files with values derived from the values.yaml file becomes accessible.

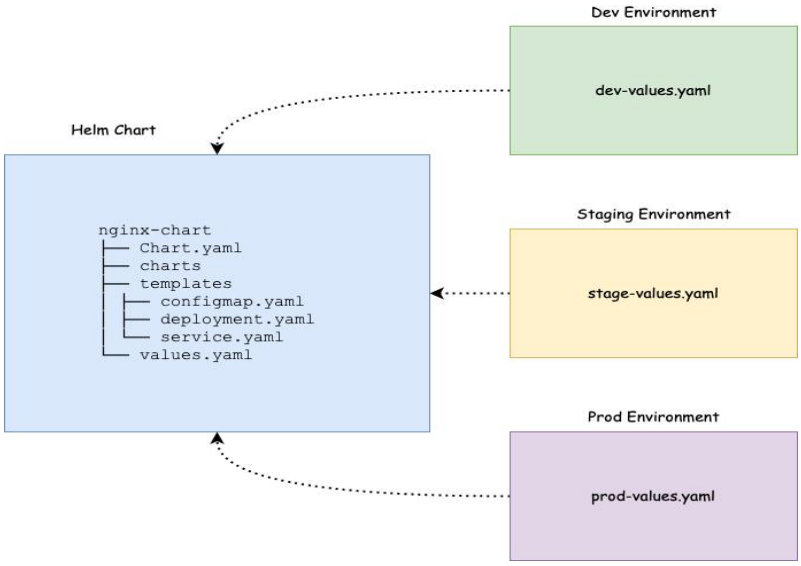

Figure 4: How to Create Helm Chart

Following the application development, the testing phase begins with a helmet installation to verify that the helmet chart operates accurately. The next phase involves putting the Chart into Kubernetes to observe the accurate generation of resources and verify the proper configuration application. Step 5 includes version control procedures, which store Helm charts within repositories to track versions and support teamwork. All charts must be designed for flexibility because users should be able to normalize or develop them when essential conditions transform. Helm chart creators should follow Helms community standards while naming everything descriptively and separating their components to enable repeat value use across various teams. Charts' active version control enables both the suitability of updates and the maintenance of multiple active versions.

4.3 Installing and Using Helm Charts

After performing Helm installation, installing Helm charts begins with the cluster deployment of an already created Kubernetes chart. A Kubernetes cluster needs several preparatory actions before enabling the installation of Helm. The local machine needs the Helm client as its initial installation requirement. The installation of Helm provides users access to add Helm repositories to store pre-built charts (McBride & Reynolds, 2020). Helm repositories function as collective data centers that store Helm charts while they support either public domains like Helm Hub or operate in private configurations. The first step in Helm implementation is installing it on a local machine and then initializing it within the Kubernetes cluster through Helm init. Installing this connection enables Helm to interact with the Kubernetes API server. Helm charts regularly reside in remote repositories, which users must add in the second stage. Users implement Helm repositories through their command line interface using the Helm repo-add command. The installation process for charts starts with the command helm install and continues with Chart downloading through this same command. The category adds Helm charts to Kubernetes clusters using the Helm install command. During installation, users can control default values from values yaml using custom configuration files or individual command-line value specifications. Helm enables users to change default values during installation using the values modification feature. The command allows users to specify custom individual values through the—-set flag and custom values files. A chart's characteristics can be modified to match the requirements of the deployed environment.

5. Benefits of Using Helm Charts for Kubernetes Deployments

Kubernetes has become the dominant platform for managing applications through containers in the cloud-native realm. Its features provide robust scalability mechanisms, yet maintaining the platform becomes complex for administrators. The challenge of Kubernetes deployment complexity meets its solution through Helm charts, which create an advanced abstraction layer that enhances its simplicity, management capabilities, and features. The following part examines Helm charts for Kubernetes deployments while showing their benefits, which reduce management complexity, improve version control

and configuration simplification, and enable scalability and flexible deployment.

5.1 Simplification of Kubernetes Management

Helm charts' primary functionality is to enhance the management process of Kubernetes systems. Kubernetes setup becomes difficult because users need a strong understanding of multiple applications and their configurations. Helm charts also achieve higher efficiency in the deployment process because users can deploy operations with reusable templates for automation. Helm eliminates manual Kubernetes manifest creation by allowing developers to produce predefined templates. The charts in these templates load every application deployment requirement, accelerating the process and minimizing human mistakes. Helm charts make managing the complexity of Kubernetes deployments easy because developers can focus on application state definition rather than low-level implementation settings. Helm charts help decrease operational costs and labor needs, streamlining the management of applications that run as microservices. Developers gain accelerated application development cycles alongside higher team productivity due to the time conservations achieved through Helm automation of Kubernetes deployments (Chavan & Romanov, 2023).

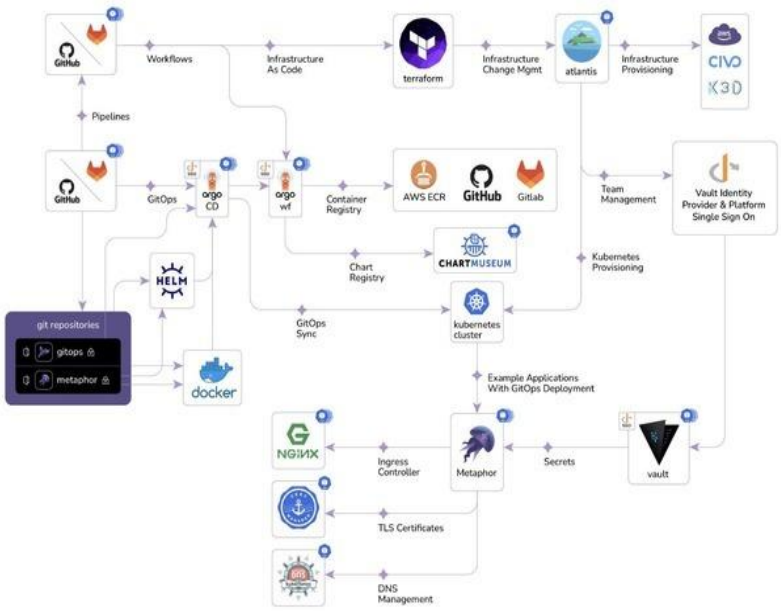

Figure 5: Simplifying Kubernetes Management with Kubefirst

5.2 Version Control and Rollbacks

Helm charts provide users with an efficient system to track and revert versions through their management framework. Applications operating in Kubernetes platforms need regular updates that trigger configuration alterations and scaling assignments. The Helm system maintains control of different Helm chart versions through its tracking feature.

The deployment system in Helm operates through version control, enabling teams to browse their previous chart versions. The version control system tracks deployment modifications so auditing procedures and troubleshooting processes can be successfully conducted (Loeliger & McCullough, 2012). Through Helm, users receive an effortless method for performing deployment rollbacks until failures are resolved. The rollback command allows developers to restore a stable version of their deployment when new deployments produce unidentified problems or mistakes. The rollback function from Helm protects applications from downtime and preserves their always-accessible state during deployment breakdowns. Helm charts provide quick rollback functionality that enhances Kubernetes deployment reliability and stability through their safety mechanisms for developers and application managers.

5.3 Easier Configuration Management

An effective configuration management system extends because of Helm charts and their useful attributes. Due to increasing service numbers, Kubernetes makes managing environment-specific configurations (development, testing, and production) difficult. Seeder Find () Binds to Error Prone Complex Configuration Files Need Different Settings Secrets and Resource Allocations for Each Environment. Users gain improved service configuration easement through Helm charts since these tools supply structured systems for configuration management. Users can set different configuration definitions through the Helm chart values files to adapt to multiple deployment settings. Users find it easy to change environmental variables and settings in values files that do not impact the core Helm chart. Second-stage SDLC managers can enhance deployment management through values files independently of the root chart configuration. Helm enables managers to develop unified configuration control systems that maintain operations stability between multiple deployment environments. The centralized management system decreases configuration drift between environments, thus reducing operational failures and deploying errors. Managers can streamline configuration management and deployment operations through Helm charts while also enhancing system maintainability, according to Imadali and Bousselmi (2018). A reusable Chart is a single maintenance tool that enables teams to redefine their configurations for particular needs by reproducing environments and uniform deployment.

5.4 Scalability and Flexibility

Helm charts deliver notable advantages regarding scalability and flexibility, among other benefits. Kubernetes' automated scaling of applications depends heavily on manual adjustments to manage resources when the application load demands transformation. Application scaling becomes more efficient through Helm charts because these charts operate flawlessly within Kubernetes' built-in auto-scaling functions. Users can define the standardized scaling parameters, including several replica resource limits and autoscaling rules, through Helm charts templates. The standardized approach helps both teams and the application respond better to workload changes. The functionality of Helm charts

allows users to make extensive modifications. Developers can change preexisting charts to fulfill their application needs by including new services alongside changes in resource settings and implementation of third-party tools. Such a flexible tool as Helm charts enables Kubernetes deployments to adapt and accommodate diverse requirements of any application regardless of size, complexity, or scope. Any application size requires deployment tools from Helm to match its operational requirements during the deployment process. Scalability must be balanced against cost-effective considerations in microservices systems (Kommera, 2013). Helm charts achieve resource optimization by allowing efficient scaling operations that fit resource availability alongside business requirements. Through Helm charts, developers maintain the ability to optimize scaling parameters to stop their applications from scaling expensively beyond necessary infrastructure requirements.

6. Advanced Helm Features

Helm is a widely used Kubernetes application management tool. Its "charts" deliver packaged Kubernetes resources to simplify complex application deployment and management. The DevOps environment relies on Helm because this tool optimizes deployment workflows, improving operational efficiency. Helm Chart dependencies, secrets management, CI/CD pipeline integration, and Helm testing are major features discussed in this section.

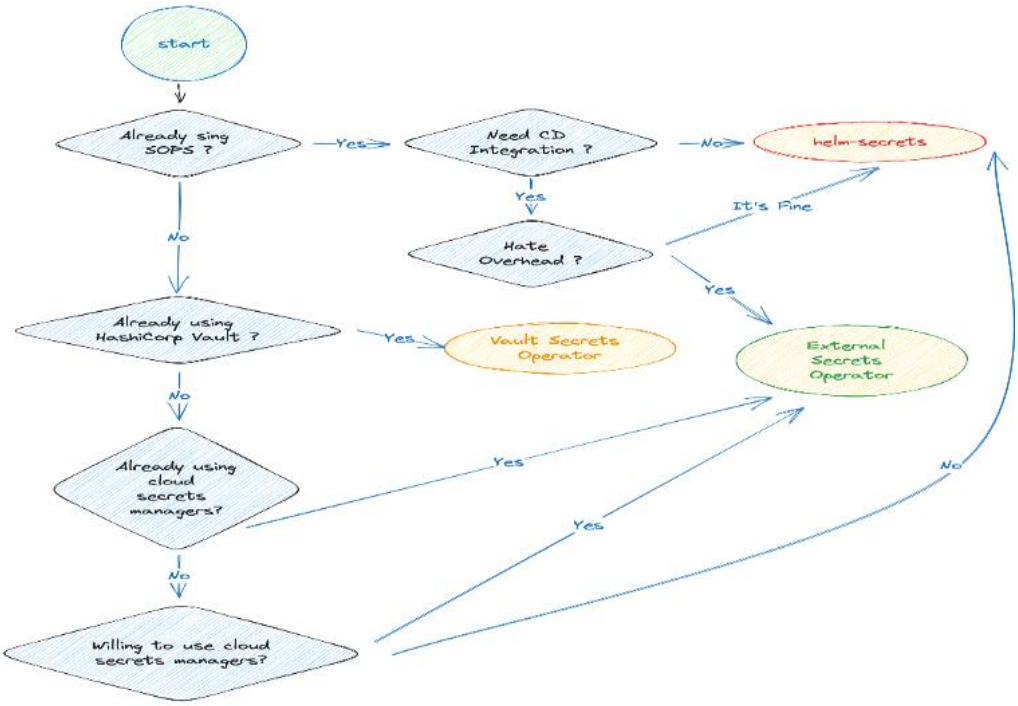

Figure 6: How to Handle Secrets in Helm

6.1 Helm Chart Dependencies

The current application development model uses multiple microservices, which ought to

work together for effective operation. However, managing dependent applications becomes difficult because manual administration may introduce human errors. Helm charts enable developers to establish efficient relationships between operating applications and services. The dependency functionality of Helm charts permits users to add other charts into their configurations as needed components. A single Helm chart can control both a web application and its required database service grouped. The list of dependencies with version requirements exists in the Chart—yaml file from the parent chart (Kumar, 2019). Applications requiring various dependencies can easily be deployed through Helm because it arranges service deployment while properly configuring dependent services and resources. The dependency feature in Helm enables two configuration options: direct dependency mode, where users govern packaged charts, and external ones through sub-chart mode. Helm deploys dependencies automatically so users achieve streamlined deployment of integrated systems, producing efficient results with minimal errors. Through Helm's dependency management system, companies achieve independent service control, which results in adaptable, simpler-to-manage application structures.

6.2 Helm Secrets Management

Web application management depends vitally on the effective handling of passwords, API keys, and configuration secrets, which are classified as sensitive information. Organizations that maintain their Kubernetes sensitive data in plaintext format create security vulnerabilities that expose their systems to attack. The Helm Secrets plugin enables organizations to protect their secrets through encryption before they are deployed into their systems. The Helm Secrets plugin maintains compatibility with Helm workloads to handle sensitive values encrypted and securely stored across AWS KMS or HashiCorp Vault platforms. Helm achieves secure data protection through encryption, preventing secret information from appearing as plain text in charts or version control repositories. For example, developers who need to configure database credentials in Helm charts should use Helm Secrets to encrypt those credentials in file form. The file allows secure, centralized management of sensitive information while offering version control and other application configurations through auditable practices. Organizations benefit from Helm Secrets deployment-managing functionality since it helps them adopt best security practices while retaining high operational efficiency (Filer, 2009). Organizations benefit from Helm Secrets because it streamlines Kubernetes Secret management yet still functions along with Kubernetes' built-in Secret resources for sensitive data protection.

6.3 Helm and Continuous Integration/Continuous Deployment (CI/CD)

Operating Helm within CI/CD pipelines is necessary for fully automated Kubernetes deployment infrastructure. The CI/CD tools Jenkins, GitLabb CI, and CircleCI bring features that automatically build, test and deploy modified software code. Helm is an automation tool that enables organizations to deploy Kubernetes applications through

their current pipelines. A user employing Jenkins can program automated pipeline sequences that invoke Helm deployment of new application updates upon repository change events. Helm chart validation and testing occur throughout the pipeline due to its integrated deployment automation. CI/CD pipelines that employ Helm allow development teams to speed up application deployments and deploy applications with consistent reliability throughout all environments. Users can implement Helm operations in GitLab CI, though gitlab-ci.yml configuration files contain predefined Helm commands. During the pipeline execution, Helm charts auto-starts for Kubernetes clusters are deployed either after build results or when merging feature-branch code. Automation makes the release process faster while human errors decrease because new features with bug fixes and improvements move quickly toward production (Dustin et al, 1999). The integration between Helm and CI/CD tools creates opportunities for teams to achieve all the advantages of DevOps by delivering continuously and boosting their operational speed.

6.4 Helm Testing

In application deployment, the evaluations must guarantee that all Helm chart resources and configurations correspond to their intended functionality for correct execution. Helm enables developers, through its testing features, to validate chart deployment readiness while ensuring proper configuration settings have been achieved. A test section dedicated to Helm must be added to the templates/ directory of the Chart. The testing section contains Kubernetes jobs and pods that evaluate the Chart's deployment performance. The application can test database connectivity while another test checks for proper environment variables configuration during pod deployment. The early stages of deployment reveal operational problems because teams conduct pre-production testing. Operators perform Testing through the helm test command to validate the operational integrity of installed charts across development staging and testing deployments. The functionality helps teams find hardware defects and configuration issues to address them prior to starting production activities. According to Sellami et al. (2020), Helm includes pre-packaged testing automation that supports developers to validate charts using continuous integration and delivery mechanisms. Testing Helm charts proves crucial for complex application environments but adds especially high value to major application deployment frameworks because of their increased error potential. Helm testing in deployment pipelines enables organizations to release dependable software versions while lowering the number of faulty configuration deployments.

International Journal of Sustainability and Innovation in Engineering
(IJSIE)
https://www.doi.org/10.56830/IJSIE2024
Vol.2 2024

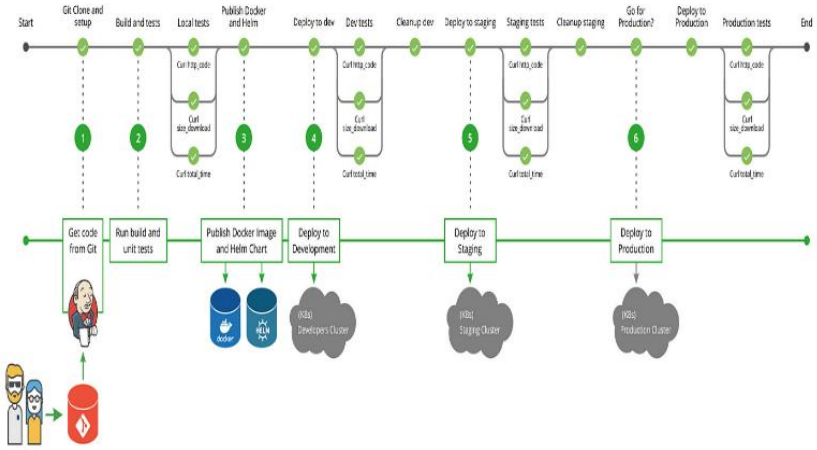

Figure 7: Easily Automate Your CI/CD Pipeline with Jenkins, Helm, and Kubernetes

7. Best Practices for Managing Helm Charts

The Helm chart system functions as the base for Kubernetes deployments because it simplifies the control of complicated application deployment processes. Through its Helm platform, users can develop complex Kubernetes systems, including definitions for infrastructure charts and installation and update automation. High success rates when using Helm features depend on user compliance with approved chart architectures, security, and performance best practices. The following guidelines help users increase Helm chart management efficiency in an important section.

7.1 Organizing Helm Chart for Easy Maintenance

The article underlines the significance of Helm chart organization since it leads to enhanced maintenance potential alongside increased reusability, according to Singh (2022). The key reason for Helm chart organization involves constructing systems enabling easy maintenance and management of charts in the future. Helm chart repositories function as necessary distribution systems that let users handle multiple charts. Users utilize these repositories as their data storage platform to manage their charts and obtain remote access and teamwork capabilities for sharing their charts. An excellent repository organization enables teams to maintain recent versions of charts, which support exact application matching for productive deployments. Artifact Hub and ChartMuseum are prominent chart repositories that unify team and organizational chart management. The need for Helm chart version control provides users with a way to track application development and chart modifications throughout time. Helm repositories act as platforms that unify Git-based platforms such as GitHub and GitLab to perform automatic deployment management through change monitoring features. The version control system helps users identify chart versions that function properly with specific Kubernetes versions and their dependency requirements. Helm charts require complete documentation to run them at optimal performance levels. The documentation section of each chart must provide complete deployment needs, which should be paired with

dependency specifications and configuration points to ensure successful deployment. The stability of charts and documentation requires version control implementation in a proper documentation system.

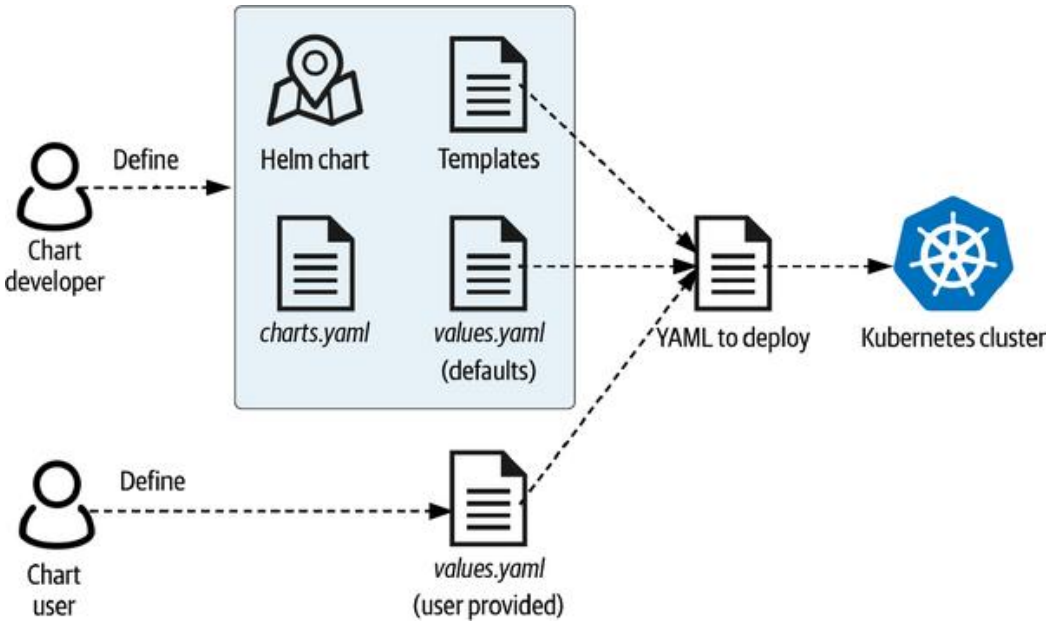

Figure 8: Simplify Application Deployment and Management with Helm Charts

7.2 Security Best Practices

Security needs special focus during Helm chart management because production environments handle essential details with these charts. The fundamental security aspect is safeguarding the procedures involved in Helm chart deployment. Proof of two security measures must exist within these procedures: data encryption for repositories and user access control through authentication and authorization systems. The system requires an adequate access control solution for repository management, enabling authorized users to execute update and download operations. Organizations running public Helm chart repositories must establish security protocols with vulnerability detection and access record creation to protect their resources from known threats. The secure management of sensitive data is crucial when working with Helm. Storage of API keys, passwords, and access tokens must be avoided within Helm charts since these represent sensitive information. The application of Kubernetes Secrets provides the secure storage of secrets for Helm to utilize when installing charts. Storage of critical secrets must be executed securely as Kubernetes supports built-in encryption features. The best practice dictates that sensitive data remains encrypted during rest and transit storage. Helm Secrets and Sealed Secrets are extra security measures that automate the encryption and decryption process of deployment-specific data through Helm charts. Secured deployment of Helm charts becomes possible through these practices, decreasing accidental exposure risks of sensitive information (Garfinkel, 2005).

7.3 Optimizing Helm Chart Performance

Hermetic resource management of Helm charts remains essential to achieve satisfactory deployment outcomes across production environments. Inadequate chart writing will cause application resources to perform poorly, resulting in diminished performance. Top-level optimization of Helm charts depends on how well resources are managed for maximum efficiency. The Helm chart performance depends on setting container resource requests alongside limits within the Helm chart configuration to guarantee that each container has ample resources but does not exceed system capacity. The defined resource configurations enable Kubernetes to function as an efficient workload manager by avoiding the starvation of resources with other running applications. Performance optimization requires administrators to modify Helm charts specifically for production deployment needs. The system requires configuration options, including cloud provider details and hardware setup specifications. The Helm chart needs supplemental features and automatic scaling capabilities to operate properly at high availability levels in permanent production environments. By modifying its Helm chart fork, the application requires the implementation of infrastructure standards, which consist of volume management features using a stateful database system and enhanced health check capabilities. Helm charts deliver important performance advantages to applications because the optimization leads to optimized operations within performance-maximized environments (Spillner, 2019).

8. Case Studies: Real-World Applications of Helm Charts

Organizations use Helm charts to enable simple, standard application deployment across multiple operation environments. The section provides evidence that Helm charts prove they are essential Kubernetes deployment configuration management tools and security improvement enablers when used in actual implementations.

8.1 Case Study 1: Simplifying CI/CD with Helm Charts in a DevOps Pipeline

DevOps operations apply continuous integration and continuous deployment (CI/CD) protocols to accelerate the deployment of functional products into their current software deployment framework. According to Singh et al. (2019), the deployment process of Kubernetes cluster applications uses Helm charts to simplify CI/CD operations. Successful code modifications necessitate testing the production environment transfer method during the CI/CD system deployment phases. Human operators perform extensive configuration procedures when executing the deployment commands for different system types. Helm charts serve as recombined deployment management utilities for handling configurations and dependency packages alongside templates. Modern e-commerce organizations use Kubernetes to manage their infrastructure, while Helm charts operate inside the CI/CD system to automate application version deployment. The deployment process needed manual YAML file configuration for every environment before Helm was implemented because this led to higher error probabilities. Helm charts

automated deployment management by tracking version control while managing deployments and configuration needs and enabling the recovery of previous versions. The new method decreased human mistakes while reducing feature release time and enabling better enhancements to the software. Helm charts enabled DevOps to create automated configuration alterations through templating capabilities that detected environmental contexts (development, staging, and production). The organization achieved improved stability and performance through Helm charts, ensuring environmental uniformity. Automated CI/CD pipeline deployment streamlined productivity and deployment outcomes by reducing errors.

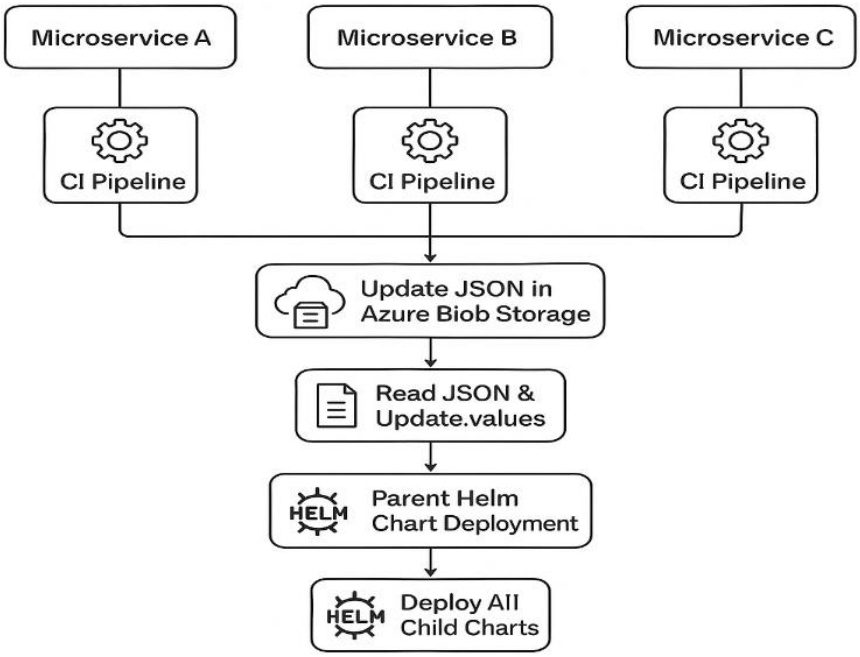

Figure 9: Simplifying CI/CD for Microservices Using Parent-Child Helm Charts

8.2 Case Study 2: Managing Multi-Environment Deployments with Helm

Helm charts deliver the advantage of handling application deployments in diverse deployment environments. Software development encounters consistent behavior challenges throughout development staging and production environments. Multi-environment deployment management through Helm charts is possible because developers can parameterize config values. Programmers enable different environment settings through their code base, while developers make separate alterations to those settings through configuration values that do not affect application code. A worldwide financial services organization adapted Helm charts to control their Kubernetes deployments, which operated across multiple environments throughout their system (Sharma & Singh, 2019). The company employed Kubernetes to host applications that required different environments with specific infrastructure specifications. Through Helm

charts, the team eliminated duplicates by controlling different environment settings while avoiding manual configuration adjustments. The secure storage of sensitive information occurred throughout deployment when developers used Helm's values—yaml file injection method. The method enabled organizations to handle varying environments collectively, and developers enhanced their operational connections with staff members. The deployment configurations inside Helm charts served as a single control point to remove errors and create uniform application deployment methods globally. The Helm charts functioned as developer tools that implemented automatic configuration updates between different operation systems.

8.3 Case Study 3: Enhancing Security in Kubernetes Deployments with Helm Charts

Organizations worldwide protect their Kubernetes deployment through applications because Torkura et al. (2018) demonstrate how it meets security needs to defend sensitive information and vital assets. Users benefit from the Helm chart framework to manage Kubernetes securely because it offers three features: security parameters, storing confidential information, and updating distribution management. The organization installed Helm charts to develop critical security-based patient data processing applications in their healthcare research programs. Security measures needed to be extensively integrated into the Kubernetes infrastructure for compliance with HIPAA and other regulatory standards. Helm charts helped the organization organize a structure that enabled their security template configuration tracking. The encryption system in Kubernetes Secrets maintained the security of every secret detailed in charts, safeguarding key data and database login details. The Helm charts employed an automatic application patching system through which security vulnerabilities could trigger updates. A quick deployment of security updates occurred through this method to decrease the chance of exploitation. Helm charts provide security management through their capability to force uniform security standards in all deployments. The healthcare organization should include security standards like network policies, role-based access control (RBAC), and pod security policies in their Helm charts. Organizational application of their security policies to every Kubernetes cluster resulted in standardized best practices throughout their deployments, thus enhancing resistance to security attacks. The Helm charts helped the organization achieve better access control management by letting them restrict what authorized users could see or change within their resources. RBAC enforcement within Helm chart templates enabled the organization to achieve maximal security because authorized personnel remained the only group with access to sensitive data (Fischer-Hellmann, 2012). The combination of Helm charts enabled the healthcare organization to fulfill security compliance standards and improve protection for their Kubernetes deployment framework.

9. Common Issues and Troubleshooting with Helm Charts

Helm charts transform Kubernetes deployment into a simplified operation, enabling

developers and system administrators to create, deploy, and configure applications. Helm charts enable users to manage applications through a simplified approach, but certain deployment obstacles frequently occur. This section covers the deployment problems users typically face alongside Helm chart error detection and provides methods to resolve failed Helm release errors when deploying Kubernetes systems.

9.1 Common Deployment Issues

Configuration mismatches are one of the main problems when deploying Helm charts. The application configuration needs of deployment do not match the values specified in the Helm chart installation. The deployment of Helm charts depends on values yaml file that sets all configuration parameters for the target application. The application receives information from deployment that differs from its intended requirements when values are set improperly. Inaccurate values set for environment variables and resource limits or application-specific settings cause problems in the deployed application to behave unexpectedly. To prevent such problems, proper validation must occur between the deployment requirements and values to verify their correspondence. One important challenge for Helm chart deployment stems from the different versions of the Helm chart and the Kubernetes cluster. The intended Kubernetes versions served by Helm charts determine their compatibility because deploying charts designed for different Kubernetes versions can produce matching issues (Krochmalski, 2017). This issue causes deployment failures, inability to access resources, and Helm command errors. The resolution requires checking Helm chart compatibility with the Kubernetes version target or updating the Helm chart and Kubernetes environment.

9.2 Helm Chart Errors

Helm charts generate errors primarily because of structural problems in the Chart itself and mistakes in the Kubernetes resources that the Chart contains. Helm generates error messages that provide helpful information concerning the nature of the problem. Users need to understand error messages when troubleshooting different problems. Three common Helm errors stem from lacking dependencies or misnamed dependencies with absent values in the values: yaml configuration or template formatting errors. The incorrect Kubernetes resource definitions, including pod services and deployments in Helm charts, will trigger specific error messages from Helm pointing to the impacted resources. A dependency chart failure occurs when the used version does not match the needed version or the Chart is missing altogether. Helm returns an error message when it detects a missing dependency, which states the absent dependency chart. It becomes essential to check if required dependency charts exist and if they properly integrate into the Helm chart configuration. Template rendering errors frequently appear when there are YAML structure or syntax issues in the Helm chart. The deployment failure of Helm charts stems from problems encountered during the Go templating generation of Kubernetes YAML, which manifests when syntax errors emerge, variables remain undefined, or vital values lack definition. Users should examine template files of the

Helm chart for syntax issues while confirming that all defined values are present (Helm et al, 1991).

9.3 Troubleshooting Helm Release Failures

The initial sources of Helm release failure require a detailed method for detection because different execution problems can lead to them. To execute Helm releases properly, assessing their current state during diagnosis is essential. The helm status command generates comprehensive release information, including resource lists, present-state status, and failure reports. Failed release status reveals the precise resource problems involving pods, service unavailability, and pod startup failures. Users can examine the Helm release deployment history through the Helm history command execution. Users can evaluate past deployment results through this command while identifying when recent modifications started causing issues. Users can identify failure sources by inspecting successful Helm release implementation against their unsuccessful counterparts, thus enabling better analysis of deployment or update failures. Users who need to find and resolve Helm release execution failures rely on the critical resource combination of Kubernetes logs and Helm commands. When applied to specific pods and containers, the command kubectl logs helps users find runtime problems beyond Helm error detection. The analysis of platform logs shows the system has absent environmental parameters, insufficient resource limits, and interconnection problems between components (Ali et al., 2015). Users must analyze Kubernetes events when undertaking the last step of fixing failed Helm releases. Kubernetes events provide specific causes of deployment failure because they store a chronological record of system operational logs. Users can check system events through kubectl to get events and locate failure indicators. Approval of deployment success demands the resolution of external system components, resource inadequacy, and network configuration problems before a solution can be achieved. The systematic execution of maintenance procedures helps users solve typical Helm chart problems, which supports application reliability and Kubernetes deployment quality (Štefanič et al., 2017).

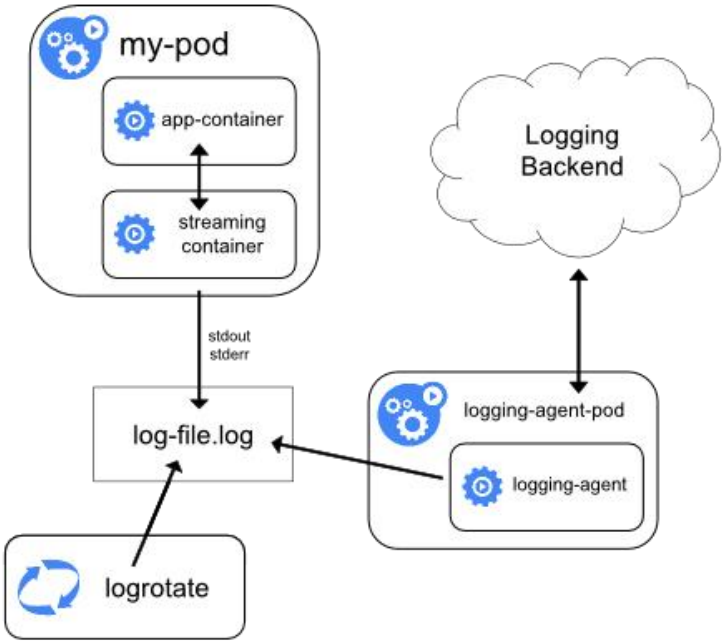

Figure 10: Logging Architecture

10. Conclusion

The Helm charts provide users with fundamental functionalities to simplify Kubernetes deployment of applications and maintain operational control functions. Helm is an essential management tool for Kubernetes application deployment systems since it attracts more users with its increased popularity. The Kubernetes application management system Helm operates on the platform through the same mechanisms that Linux package managers apt and yum use on their platforms. Developers using Helm chart development tools create application packages that automatically deploy applications while reducing mistakes in manual installation procedures. Helm charts' main benefit during Kubernetes deployments is their ability to make complex configurations easier to manage. Kubernetes management systems present significant obstacles to teams because experts must continually handle the complex configuration files that need updating when application requirements change. Helm allows users to translate Kubernetes configurations into flexible charts, which users can easily rester. The predetermined templates in these charts enable one-command deployment for applications and include all Kubernetes objects, such as Pods Services and Deployments. This deployment method standardizes the process so development stages match production stages, resulting in improved procedure control.

Helm Version Control allows developers to view deployment configurations made after applications go live. The deployment record system manages versions of records that enable easier troubleshooting and verification protocols. The deployment rollbacks available through Helm charts create an easy user interface to maintain Kubernetes application reliability throughout maintenance operations. Helm enables service

continuity through its dependency management system and automatic application update feature, thus producing better operational effectiveness. The upcoming deployment and management control process will gain direction from numerous Kubernetes and Helm development plans. Helm serves as the key attraction for GitOps organizations because it lets them implement Git repository management strategies for their operational technique. Teams can establish infrastructure declarations in Git version controllers by storing Helm charts in these repositories as part of the GitOps process. Operation efficiency improves in present-day methods because better teamwork implementation allows teams to modify following standard code review processes.

Business organizations have made Helm their enhancement strategy by integrating it with Continuous Integration/Continuous Deployment (CI/CD) interfaces. CI/CD pipelines achieve automatic deployment capabilities through Helm charts, allowing fast application launch with built-in deployment automation features. Cheaper Kubernetes updates will improve Helm management through new features that let users handle numerous clusters, serverless deployments, and hybrid cloud solutions. The management operations for Kubernetes applications reach peak efficiency when employing Helm charts. Organizations benefit from Helm deployment by obtaining pre-configured management systems that create secure environments for smooth updates and easy restore operations. Kubernetes management features have been strengthened continuously due to GitOps integration and improved CI/CD capabilities, making Helm the default platform management solution. All organizations seeking to enhance their Kubernetes deployment method need Helm charts as an essential organizational requirement. The installation of Hub creates an efficiency boost by simplifying complex operations while allowing teams to maintain complete control over stable Kubernetes environments for deployments.

References

[1] *Ali, S., Qaisar, S. B., Saeed, H., Farhan Khan, M., Naeem, M., & Anpalagan, A. (2015). Network challenges for cyber physical systems with tiny wireless devices: A case study on reliable pipeline condition monitoring. Sensors, 15(4), 7172-7205.*

[2] *Bekas, E. (2017). Service Management in NoSQL Data Stores via Replica-group Reconfigurations (Doctoral dissertation, University of Ioannina).*

[3] *Block, A., Dewey, A., & Mocevicius, R. (2022). Managing Kubernetes Resources Using Helm: Simplifying how to build, package, and distribute applications for Kubernetes. Packt Publishing Ltd.*

[4] *Bogdan, R. (2023). Automated System for Managing and Deploying Cloud-based Demo Tests.*

[5] *Burns, B., & Tracey, C. (2018). Managing Kubernetes: operating Kubernetes clusters in the real world. O'Reilly Media.*

[6] *Chavan, A. (2023). Managing scalability and cost in microservices architecture: Balancing infinite scalability with financial constraints. Journal of Artificial Intelligence & Cloud Computing, 2, E264. http://doi.org/10.47363/JAICC/2023(2)E264*

[7] *Chavan, A., & Romanov, Y. (2023). Managing scalability and cost in microservices architecture: Balancing infinite scalability with financial constraints. Journal of Artificial Intelligence & Cloud Computing, 5, E102. https://doi.org/10.47363/JMHC/2023(5)E102*

[8] *Dustin, E., Rashka, J., & Paul, J. (1999). Automated software testing: introduction, management, and performance. Addison-Wesley Professional.*

[9] *Estrem, W. A. (2003). An evaluation framework for deploying Web Services in the next generation manufacturing enterprise. Robotics and Computer-Integrated Manufacturing, 19(6), 509-519.*

[10] *Filer, S. M. (2009). Managing hostile environments: journalists and media workers: learning to survive the world's difficult, remote and hostile environments (Doctoral dissertation, Queensland University of Technology).*

[11] *Fischer-Hellmann, K. P. (2012). Information Flow Based Security Control Beyond RBAC: How to enable fine-grained security policy enforcement in business processes beyond limitations of role-based access control (RBAC) (Vol. 1). Springer Science & Business Media.*

[12] *Garfinkel, S. (2005). Design principles and patterns for computer systems that are simultaneously secure and usable (Doctoral dissertation, Massachusetts Institute of Technology).*

[13] *Giraldo Moreno, D. (2020). Rolling update and monitoring deployment strategy for edge layer IoT devices.*

[14] *Gokhale, S., Poosarla, R., Tikar, S., Gunjawate, S., Hajare, A., Deshpande, S., ... & Karve, K. (2021, September). Creating helm charts to ease deployment of enterprise application and its related services in kubernetes. In 2021 international conference on computing, communication and green engineering (CCGE) (pp. 1-5). IEEE.*

[15] *Helm, R., Marruitt, K., & Odersky, M. (1991, March). Building visual language parsers. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 105-112).*

[16] *Imadali, S., & Bousselmi, A. (2018, November). Cloud native 5g virtual network functions: Design principles and use cases. In 2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2) (pp. 91-96). IEEE.*

[17] *Kommera, A. R. (2013). The Role of Distributed Systems in Cloud Computing: Scalability, Efficiency, and Resilience. NeuroQuantology, 11(3), 507-516.*

[18] *Konneru, N. M. K. (2021). Integrating security into CI/CD pipelines: A DevSecOps approach with SAST, DAST, and SCA tools. International Journal of Science and Research Archive. https://ijsra.net/content/role-notification-scheduling-improving-*

patient

[19] *Krochmalski, J. (2017). Docker and Kubernetes for Java Developers. Packt Publishing Ltd.*

[20] *Kumar, A. (2019). The convergence of predictive analytics in driving business intelligence and enhancing DevOps efficiency. International Journal of Computational Engineering and Management, 6(6), 118-142. https://ijcem.in/wp-content/uploads/THE-CONVERGENCE-OF-PREDICTIVE-ANALYTICS-IN-DRIVING-BUSINESS-INTELLIGENCE-AND-ENHANCING-DEVOPS-EFFICIENCY.pdf*

[21] *Loeliger, J., & McCullough, M. (2012). Version Control with Git: Powerful tools and techniques for collaborative software development. " O'Reilly Media, Inc.".*

[22] *Matteson, R. (2010). DepMap: Dependency Mapping of Applications Using Operating System Events. California Polytechnic State University.*

[23] *McBride, B., & Reynolds, D. (2020). Survey of time series database technology.*

[24] *Oppenheimer, D., Ganapathi, A., & Patterson, D. A. (2003). Why do Internet services fail, and what can be done about it?. In 4th Usenix Symposium on Internet Technologies and Systems (USITS 03).*

[25] *Sellami, R., Zalila, F., Nuttinck, A., Dupont, S., Deprez, J. C., & Mouton, S. (2020, September). Fadi-a deployment framework for big data management and analytics. In 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) (pp. 153-158). IEEE.*

[26] *Sharma, R., & Singh, A. (2019). Getting Started with Istio Service Mesh: Manage Microservices in Kubernetes. Apress.*

[27] *Singh, V. (2022). Visual question answering using transformer architectures: Applying transformer models to improve performance in VQA tasks. Journal of Artificial Intelligence and Cognitive Computing, 1(E228). https://doi.org/10.47363/JAICC/2022(1)E228*

[28] *Singh, V., Unadkat, V., & Kanani, P. (2019). Intelligent traffic management system. International Journal of Recent Technology and Engineering (IJRTE), 8(3), 7592-7597. https://www.researchgate.net/profile/Pratik-Kanani/publication/341323324_Intelligent_Traffic_Management_System/links/5ebac410299bf1c09ab59e87/Intelligent-Traffic-Management-System.pdf*

[29] *Spillner, J. (2019). Quality assessment and improvement of helm charts for kubernetes-based cloud applications. arXiv preprint arXiv:1901.00644.*

[30] *Spillner, J. (2019). Quality assessment and improvement of helm charts for kubernetes-based cloud applications. arXiv preprint arXiv:1901.00644.*

[31] *Štefanič, P., Kimovski, D., Suciu, G., & Stankovski, V. (2017, August). Non-functional requirements optimisation for multi-tier cloud applications: An early warning system case study. In 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing &*

Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI) (pp. 1-8). IEEE.

[32] Torkura, K. A., Sukmana, M. I., Cheng, F., & Meinel, C. (2018). Cavas: Neutralizing application and container security vulnerabilities in the cloud native era. In Security and Privacy in Communication Networks: 14th International Conference, SecureComm 2018, Singapore, Singapore, August 8-10, 2018, Proceedings, Part I (pp. 471-490). Springer International Publishing.