# Helm for Continuous Delivery of Serverless Applications on Kubernetes

**Anila Gogineni**

Independent Researcher, USA
anila.ssn@gmail.com

**Abstract**

**This paper focuses on exploring the utilization of Helm in enabling coantinuous delivery of serverless applications that run on the Kubernetes. Helm helps in covering a package manager for Kubernetes that makes the packaging of serverless architecture more solid with the template Helm charts. By using these charts, developers are able to consolidate and orchestrate deployment pipelines over different environments. The problem addressed in the study is related to real-world issues of serverless applications, focusing on how serverless technologies have to work with Kubernetes, issues with multi-environment configuration, scaling, and dependency injection. It further details how Helm operates to overcome these issues and effectively interface with advanced CI/CD systems including Jenkins, GitLab CI, and ArgoCD. This paper also provides architectural recommendations that make Helm more suitable to serve in serverless orchestration, focusing on the scalability, reliability, and performance. Also, it showcases the techniques for extending the helm to meet the peculiarities of serverless technologies, such as OpenFaaS and Knative. These considerations can be supported by comparing the performance of Helm with the real-world use cases and marking its importance in increasing the deployment performance and reliability. This work, therefore, proposes the use of Helm-driven pipelines to fulfill the increasing demand for dependable and standardized tooling in serverless app development.**

**Keywords: Helm, Kubernetes, Serverless Applications, CI/CD Pipelines, Helm Charts, Resource Scaling, Prometheus, Grafana, DevOps, Application Monitoring.**

## I. INTRODUCTION

Kubernetes has emerged as the de facto standard for container orchestration, offering scalability, flexibility, and Kubernetes has quickly become the most popular platform for container orchestration, allowing for the efficiency as well as strong resilience. As a result, Kubernetes is considered one of the most effective systems for developing and deploying serverless workloads since the technology leans on infrastructure management and lightweight, event-based functions. However, serverless solutions entail challenging issues regarding dependencies, multiple environment support and versions, fairly challenging for the development and operationalization teams. These issues explain the importance of correct and effective tools and approaches for similar deployments across these environments. Helm solves these issues through the creation of reliable packages for Kubernetes known as Helm charts that ease the packaging, configuration, and deployment of an application. Helm helps to declare the resources and their relationships in a concise and idiom way which allows for not having configuration drift and being able to reliably update resources in a serverless environment. Moreover, Helm's compatibility with CI/CD tools such as Jenkins, GitLab CI, and ArgoCD makes it valuable in today's DevOps environment.

This paper aims to explore the potential of Helm in simplifying the deployment of serverless applications. This paper seeks to establish how Helm can be used to solve this problem of deploying serverless applications on Kubernetes. The thesis identifies the key issues in this field and discusses the potential approaches based on Helm. Furthermore, the paper also discusses architectural recommendations for enhancement to helm to achieve effective workflows hence enhancing deployment reliability and scalability. Consequently, this paper aims at employing the results from this study to offer practical recommendations on how to meet the emerging needs of serverless application scenarios in a Kubernetes environment.

## II. FOUNDATIONS AND CORE CONCEPTS

### Evolution of Kubernetes and Serverless Architectures

Kubernetes has become an essential platform for the deployment and orchestrations of containerized applications, providing strong capabilities regarding resource allocation, scalability, and distribution [1]. The progression of this services offering has allowed it to incorporate itself into serverless modes of architecture which allow extension of infrastructure optimization of developer resources, that centers its focus on running business logic. Serverless computing has been adopted by many due to its ability to self-manage workloads, take the best value from people through usage-based billing and offer more freedom when it comes to adding new features.

Nevertheless, managing serverless applications on Kubernetes is not without its difficulties. Kubernetes is rich but challenging to manage the lifecycle of serverless functions given their short-lived and trigger-based execution. Other problems like cold startup, scaling to zero and having similar configuration everywhere also contributes to this complexity. However, the combination of Kubernetes with serverless perspectives has brought a revolution for organizations to yield versatile, efficient, and cloud-based applications to manage the dynamic users' expectations.

### Role of Helm in Streamlining Kubernetes Operations

Helm is known as 'Kubernetes Package Manager', which is intended to help with the application and resources deployment into the Kubernetes clusters. Helm charts package an application and related dependency solutions in a way that makes it easy to do multiple identical applications [2]. A Helm chart pack contains charts, templates, and configuration files, which denote primitives such as pod, service, and ingress controller. Helm extends its capability with dynamic value files for flexibility onboard, allowing the developers to tweak deployments in line with application type using the template.

Helm helps manage application lifecycles with its versioning and rollback features. Helm charts and all application configurations can be stored in repositories for easy access by multiple teams. Helm chart releases in a cluster manage the state of deployed apps and can be updated or retracted. Helm simplifies various slow, human-interference routines while retaining application stability and uniformity.

### Advancing Continuous Delivery in Cloud-Native Environments

Continuous Delivery (CD) is now a synonym of efficient and effective release process, which allows an organization to deliver software frequently, reliably and speedily [3]. In cloud native architectures, CD pipelines are important in orchestrating the deployment of serverless applications which are complex architectures especially in how micro services integrate with the underlying infrastructure.

When implemented into CD pipelines, Helm is able to significantly reduce the impact of human factors and ensures correctness of the application configuration across the environments. Helm enables it to become

part of Continuous Integration (CI) since it would assist in building, testing, and deploying an application as soon as the code undergoes a modification.

## III. DESIGNING CONTINUOUS DELIVERY PIPELINES FOR SERVERLESS APPLICATIONS

### Architectural Overview

Kubernetes, Helm, and serverless frameworks like OpenFaaS and Knative have changed deployment of certain serverless applications. Kubernetes stands as a resilient run-time for containerized environments; conversely, serverless computing offers developers the ability to build functions that fire in response to specific events with no need to proportion the underlying infrastructure [4]. In this setup, Helm plays the role of matching application configurations with the essential tasks performed on Kubernetes.

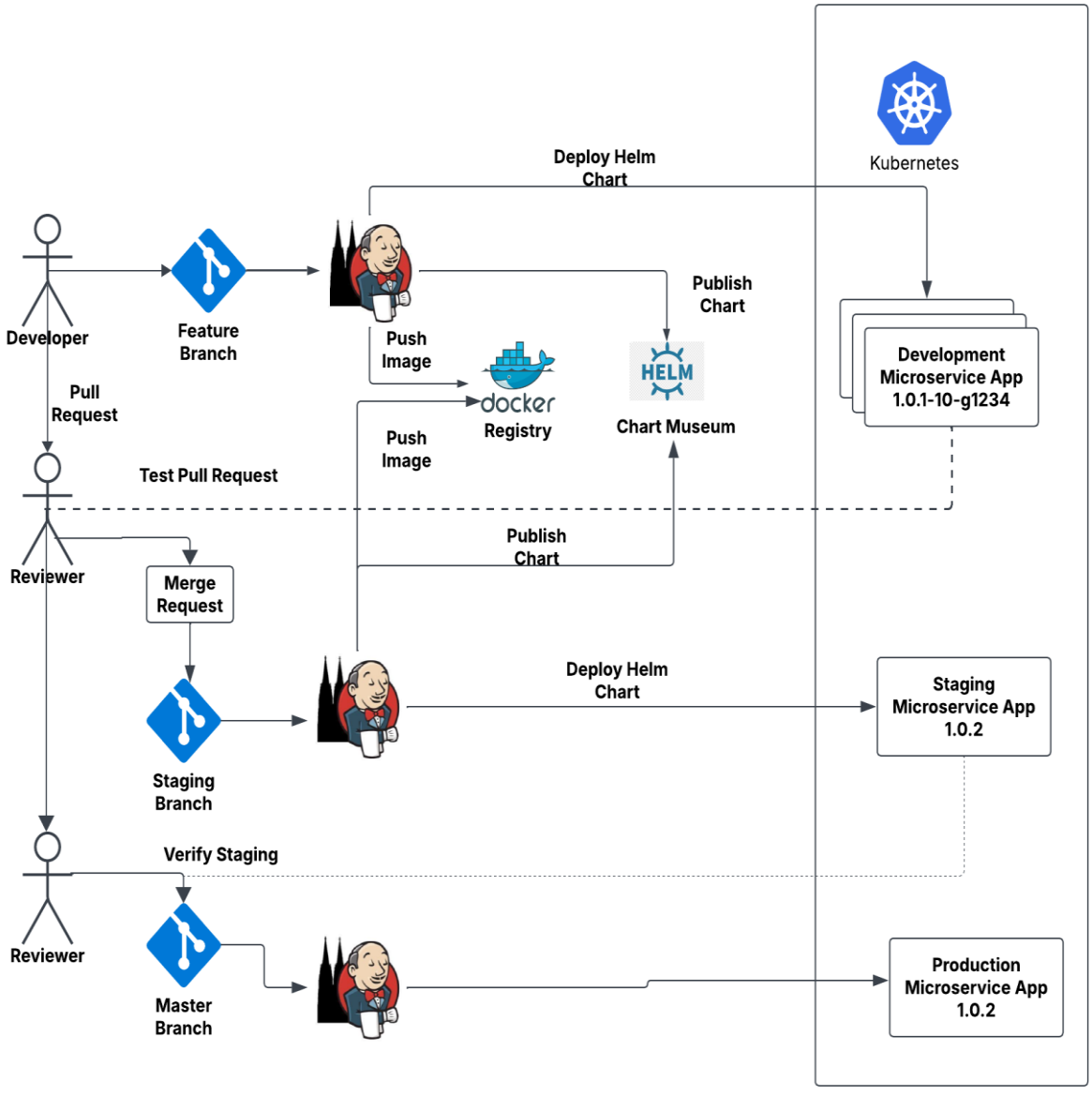


**FIG 1: System Architecture Showing Helm as a Key Orchestrator in CI/CD Pipelines**

The proposed architecture makes use of Helm charts to manage applications and its dependencies in order to create a consistent and easily reproducible environment for each application level. It also includes CI/CD solutions in this architecture to have an automated deployment process to minimize the potential error or human intercession.

### Helm Charts for Serverless Deployment

Kubernetes deploy and expose serverless applications on Kubernetes; Helm charts help with the creation of a structured and deployed valuable layout that defines the Kubernetes configurations needed to run the application. They fix the interfaces for passing functions and for the modules they depend on [5]. With Helm, it can be specified how much resources should be provided for the serverless functions, event processing and the environment besides other values in a single chart. For instance, Helm charts containing serverless functions may contain templates reflecting Kubernetes objects like Deployments, Services or ConfigMaps, containing parameters that signify certain environment conditions. This particular templating feature makes it easy for an organization to use the same chart in the creation of the different kinds of architectures for the applications needed in serverless environments in staging, testing, and production settings with just a little tweaking.

### Integration with CI/CD Tools

The integration of Helm with CI/CD such as Jenkins, GitLab CI, and ArgoCD helps to improve the chain of its deployment for serverless applications. Such tools serve as the CI/CD enablers as they are responsible for tasks like compiling, compiling and running tests, and deploying source code. When used together with Helm, they allow for the creation of a smooth code committed to the production process. For example, Jenkins can be set to alert Helm for the creation of deployments right after the successful build and test of the code.

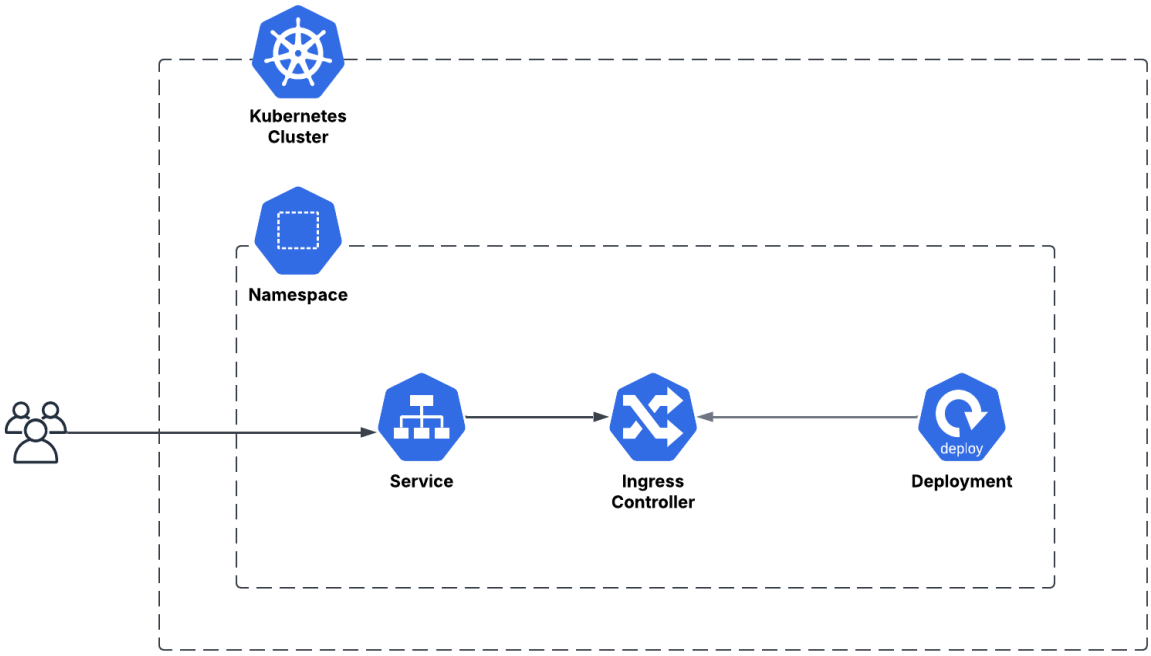


**FIG 2: Workflow Diagram of Helm with GitOps for Managing Serverless Deployment Pipelines**

With the help of Jenkins pipelines, it is possible to state the series of steps that include linting Helm charts, the validation of configurations, as well as applying Helm charts to the Kubernetes clusters. GitLab CI adds to this with integrated Helm-based GitOps support, wherein changes to Helm charts residing in a Git repository cause a deployment [6]. ArgoCD integrates with Helm and is also a declarative GitOps tool that automatically tracks Helm chart updates in the Git repository and applies them to the cluster. This approach guarantees that the state of the serverless application deployed in the cloud is always in sync with that of the repository.

### Addressing Challenges

One of the possible issues arising from the deployment of serverless in Kubernetes is configuration management, scale, and conflicts of dependencies. Helm solves these challenges by Templating, Packaging, and Versioning it's services. Another important issue is handling configuration in development, testing and production environments. This Helm does by letting one build templates with parameters, where the environment-specific values go into their own files. This separation enables teams to employ the same Helm chart used in one or the other environment, which makes it easier and reduces the tendencies for reinventing the wheel or making mistakes [7]. Another important aspect, as we know that in serverless use requests could be more or less frequently, is scalability. It also supports scalability by providing the ownership of the CPU and memory resource parameters within the chart. This flexibility ensures that serverless functions have the right resources that will enable them to run without hitches.

Dependency management is often a challenge in complex serverless architectures, where multiple services and Dependency management is always an issue in many serverless applications, especially where there are many services and libraries. Helm addresses this problem by permitting dependencies within a single chart; this way, teams can write dependencies. This feature of Pike ensures that all the required components are deployed at once meaning that there is no conflict as well as compatibility issues. From the performance standpoint, Helm cuts down deployment time compared to manual deployment by a huge margin.

## IV. ENHANCING HELM FOR SERVERLESS WORKFLOWS

### Extending Helm's Capabilities

Helm is highly flexible and has the ability to set templates, which are highly valuable for writing serverless applications; however, more developments could make it even better in serverless environments. One possible addition is the development of specific Helm templates designed for serverless platforms like OpenFaaS, Knative, and AWS Lambda. These templates can define resources that are specific to a service or pattern, such as event trigger, auto scaler, and function specific configurations which can help in faster and consistent deployment. Further, improving Helm's parameterization provides more flexibility in specifying 'what' can be customized without reducing the level of reuse. For example, the support for workload metric-facilitated dynamic resource allocation can be integrated to address this problem in serverless environments. Another enhancement is the ability to connect Helm charts with monitoring and logging solutions, including Prometheus and Grafana, by using custom hooks and annotations that help increase monitorability of serverless applications.

### Case Study

A real-world example of deploying serverless possesses the sorted logic of why Helm outperforms its competitor. Consider an organization that utilizes Knative on Kubernetes to run several functions based on events. With the help of Helm these functions can be wrapped into charts so they can be easily redeployed and there will be no difference between the development, staging and production environments. In this regard, Helm greatly enhances capabilities of the deployment process and concerns with such problems as configuration drift and errors that are known to occur in manual deployment. With Helm, this process is not manual as every time it makes the deployment it goes back to the known configuration state. Overall, integration of The Helm with CI/CD tools improves the pipeline experience as indicated below [8]. It can also avoid configuration problems before they emerge and guarantee the stability of serverless applications at the cluster level. This contributes to a higher deployment success rate as Helm simplifies management,

resolves dependency conflicts and makes rolling back in case of failure easier. Helm is shown to be a crucial tool for managing and scaling serverless operations in this example, which significantly emphasizes its importance in today's deployment processes.

## V. IMPLEMENTATION DETAILS

### Deployment Workflow

The process of installing a serverless application through the aid of Helm has various steps that are clearly defined to support deployment symmetry across different environments. The first one is to create the Helm chart specific to the serverless application. This entails charting generation which entails creating templates for various Kubernetes objects such as Deployment, Service and ConfigMap [12]. These templates are parameterized to provide dynamic ways of defining environment bound parameters like the resource's limits, environment variables and the function specific configurations. Following that, the developers enter the values.yaml file to include application-specific and environment-specific configurations. This file keeps 'config' out of 'code,' making a stronghold and guaranteeing reuse and enhancement. This Helm chart is then limited to check on the validity of the helm chart structure and the syntax of the different helm templates.

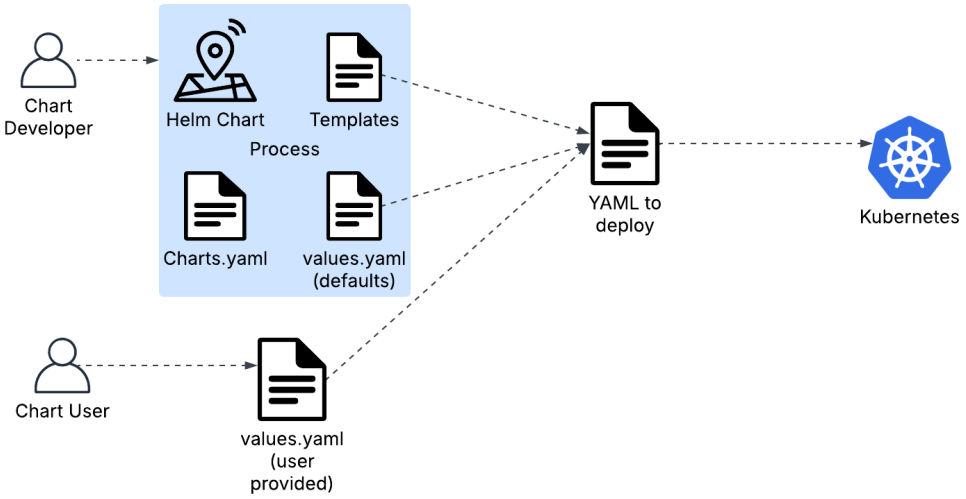


**FIG 3: The Deployment Workflow for Creating And Deploying A Helm Chart**

After that, to install Helm's chart, the helm install command is used with the Helm chart as the argument and the target Kubernetes cluster as value. While doing so, Helm transforms these templates to Kubernetes manifests, except for the values, and then applies them to the cluster. In case of a problem, a specific form in Helm allows developers to roll back to a previous release, thus making Helm more reliable and reducing the chances of having to spend a lot of time in down time [9]. This way the specific deployment standard and growth remains realistic and comprehensible which is helpful in keeping operations linear.

### Monitoring and Troubleshooting

Managing and especially detecting problems in serverless applications is crucial for their stability and performance. If Helm is deployed, the most important aspect is to monitor the various applications in order to have a good check on the health of the system. Helm collaborates well with tools such as Prometheus and Grafana that help teams monitor the application behavior and utilization of resources in real-time. Helm allows Prometheus to monitor the Kubernetes resources it creates, including CPU and

memory consumption, request response times, and function duration. These are useful metrics for evaluating the behavior of serverless applications under different load conditions. Grafana then puts those metrics on the dashboard to allow developers and operation teams to be able to manipulate that system.

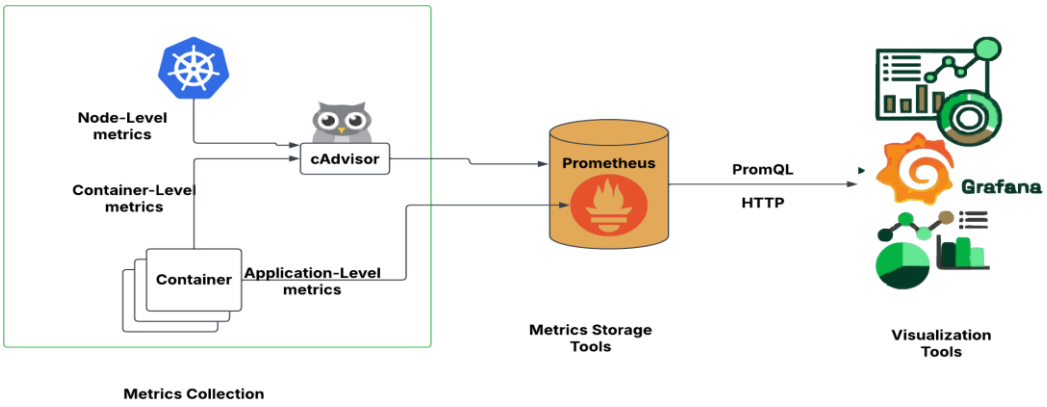


**FIG 4: Visualization of Serverless Function Performance Metrics**

Difficulty monitoring serverless applications' problems like failed deploys, imbalance in scales, and problems in configurations. Release tools such as helm status and helm get values helps in diagnosing problems by providing the current Helm deployment and configuration information. The Helm hooks benefits are apparent since the developers can conduct pre-deployment and post-deployment checks to establish if there are shortcomings and fix them at this stage of the deployment pipeline [10].

## VI. CHALLENGES AND BEST PRACTICES

### Resource Management and Autoscaling

While developing and hosting serverless applications on Kubernetes, some of the known issues include resource utilization and scaling that happens automatically. Serverless workloads are inherently event-driven and experience workloads that can be variable and therefore, Kubernetes clusters that are used to support application needs must be scaled [11]. However, to configure auto scaling policies, especially for temporary serverless functions, different features may be challenging. Incorrect policies may lead to inadequate resource capacity resulting in poor system performance or provision of excessive resources that are uneconomical.

Reducing resource management issues, Helm enables developers to declare the resources that a particular component of an application will require and the maximum limit that will be allowed when using the charts with Kubernetes. Also, Kubernetes' Horizontal Pod Autoscaler (HPA) can be linked to Helm for scaling depending on such parameters as CPU, memory or any other parameter. However, these tools alone are not sufficient to fully resolve the resource contention issues within a shared environment [13]. Those efficient methods are performance testing, using Helm Values for autoscaling tuning by workloads, using predictive scaling to learn about more heavy workloads.

### Security and Monitoring

Challenges like Security and observability, which are sensitive areas when running serverless applications on Kubernetes, need to be solved, especially when employing the Helm Continuous Delivery. In terms of security risks, some of the areas of concern include the handling of some restricted information like API keys and database credentials. Keeping such data in plaintext within Helm charts or Kubernetes manifests is not advisable due to the following reasons. Helm addresses this issue with solutions like secrets

management, which allows Helm secrets to be encrypted and securely integrated into application configurations. However, it is still a challenge to ensure these secrets are periodically rotated and protected from unauthorized access.

Supervising serverless applications is not an easy task owing to the stateless and distributed structure of the applications. Other kinds of tools like Prometheus, Grafana, and Jaeger can be incorporated into Kubernetes clusters to meter application performance and to track and analyze the execution of functions. To use these tools, Helm eases the task of installing these charts. The developers need to determine the right metrics and logging standards that will be effective to gather. Security means to use RBAC for Helm deployments, to check Helm charts with linters for misconfigurations and using centralized logging and distributed tracing for monitoring.

## VII.CONCLUSION

Helm has been established to be a useful tool in the ongoing stream of serverless applications on Kubernetes, which makes the work of maintaining the workflow of the contemporary delivery pipeline simplified and efficient. One is that it deploys application configuration encapsulated in reusable Helm charts, thus reducing the level of manual operations and configuration divergence. In that way, Helm increases developers' efficiency and guarantees the uniformity of environments and important tasks. They are scalability and serverless application management not only in Helm itself but also in various large and variable organizational settings. The integration of Helm with other CI/CD tools brings even more automation so that releases occur quickly and efficiently, while still being manageable and versioned. Furthermore, the ability to template, version and parameterize effectively enables teams to handle issues such as configuration in multiple environments and dependency management. With the advancements in serverless architectures, having Helm-based pipelines is crucial for companies and organizations aiming to improve their deployment processes. Helm, therefore, enables teams to launch higher iterations, better performance, and increased scalability leading to innovation in operations. The use of Helm for serverless workflow is not limited to solving the issues related to the deployment but also helps organizations prepare for the modern landscape of software delivery.

## VIII. APPENDIX

```
# Helm Chart Configuration for Serverless Application Deployment

apiVersion: v2

name: serverless-application

description: Helm chart for deploying serverless applications on Kubernetes

type: application

version: 1.0.0

appVersion: "1.0.0"


# Default Values Configuration

values:

  global:
```

```
    namespace: serverless-namespace
    imagePullPolicy: IfNotPresent
  serverlessApp:
    image:
      repository: myregistry/serverless-function
      tag: latest
    replicas: 3
    resources:
      requests:
        memory: "128Mi"
        cpu: "250m"
      limits:
        memory: "256Mi"
        cpu: "500m"
    environmentVariables:
      - name: DATABASE_URL
        value: "mysql://db-service.default.svc.cluster.local:3306/db"

# Kubernetes Deployment Template
templates:
  deployment.yaml: |
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: {{ .Release.Name }}-deployment
      namespace: {{ .Values.global.namespace }}
    spec:
      replicas: {{ .Values.serverlessApp.replicas }}
      selector:
        matchLabels:
          app: {{ .Release.Name }}
```

```
    template:
      metadata:
        labels:
          app: {{ .Release.Name }}
      spec:
        containers:
          - name: {{ .Release.Name }}-container
            image: "{{ .Values.serverlessApp.image.repository }}:{{ .Values.serverlessApp.image.tag }}"
            imagePullPolicy: {{ .Values.global.imagePullPolicy }}
            env:
              {{- range .Values.serverlessApp.environmentVariables }}
              - name: {{ .name }}
                value: {{ .value }}
              {{- end }}
            resources:
              requests:
                memory: {{ .Values.serverlessApp.resources.requests.memory }}
                cpu: {{ .Values.serverlessApp.resources.requests.cpu }}
              limits:
                memory: {{ .Values.serverlessApp.resources.limits.memory }}
                cpu: {{ .Values.serverlessApp.resources.limits.cpu }}


# Service Template for External Access
  service.yaml: |
    apiVersion: v1
    kind: Service
    metadata:
      name: {{ .Release.Name }}-service
      namespace: {{ .Values.global.namespace }}
    spec:
      selector:
```

```
  app: {{ .Release.Name }}
 ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

## IX. REFERENCES


[1] S. Mohanty, "Evaluation of Serverless Computing Frameworks Based on Kubernetes," *Evaluation of Serverless Computing Frameworks Based on Kubernetes*, Aug. 2018.

[2] Perez, Alfonso & Moltó, Germán & Caballer, Miguel & Calatrava, Amanda. (2018). Serverless computing for container-based architectures. Future Generation Computer Systems. 83. 10.1016/j.future.2018.01.022.

[3] Yousefpour et al., "All one needs to know about fog computing and related edge computing paradigms: A complete survey," Journal of Systems Architecture, vol. 98, pp. 289–330, Feb. 2019.

[4] A. Perez, S. Risco, D. M. Naranjo, M. Caballer, and G. Molto, "On-Premises Serverless Computing for Event-Driven Data Processing Applications," *On-premises Serverless Computing for Event-driven Data Processing Applications*, pp. 414–421, Jul. 2019.

[5] A. Palade, A. Kazmi, and S. Clarke, "An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge," *An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge*, Jul. 2019.

[6] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, "Horizontal pod autoscaling in kubernetes for elastic container orchestration," Sensors, vol. 20, no. 16, p. 4621, Aug. 2020.

[7] P. Benedetti, M. Femminella, G. Reali, and K. Steenhaut, "Experimental analysis of the application of serverless computing to IoT platforms," *Sensors*, vol. 21, no. 3, p. 928, Jan. 2021.

[8] A. Pham, "Building Continuous Delivery Pipeline for Microservices," Building Continuous Delivery Pipeline for Microservices, Jan. 2018.

[9] M. D'Amore, "GitOps and ArgoCD: Continuous deployment and maintenance of a full stack application in a hybrid cloud Kubernetes environment," *GitOps and ArgoCD: Continuous Deployment and Maintenance of a Full Stack Application in a Hybrid Cloud Kubernetes Environment*, Apr. 2021.

[10] S. Bobrovskis and A. Jurenoks, "A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment.," *A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment.*, pp. 314–322, Jan. 2018.

[11] Poniszewska-Marańda and E. Czechowska, "Kubernetes Cluster for automating software production environment," Sensors, vol. 21, no. 5, p. 1910, Mar. 2021.

[12] Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: migration to a Cloud-Native architecture," IEEE Software, vol. 33, no. 3, pp. 42–52, Mar. 2016.

[13] M. C. Borges, S. Werner, and A. Kilic, "FaaSter Troubleshooting - Evaluating Distributed Tracing Approaches for Serverless Applications," *Faaster Troubleshooting-evaluating Distributed Tracing Approaches for Serverless Applications*, pp. 83–90, Oct. 2021.