Gitops-based Framework for Kubernetes Cluster Deployment over Hybrid Cloud

Kiem Nguyen Trung*, Young Han Kim
Soongsil University

kiemnt@dcn.ssu.ac.kr*, younghak@ssu.ac.kr

Abstract

In the domain of cloud computing, the application of GitOps to Kubernetes enhances efficiencies in automating, managing, and deploying applications through a Git repository, which serves as a single source of truth. Nonetheless, managing multiple Kubernetes clusters in Hybrid Cloud environments manually with various configurations is prone to errors, including those due to human oversight, and demands substantial manual effort. This paper presents an innovative design to automate the GitOps deployment process on Kubernetes clusters within Hybrid cloud environments, utilizing the integration of Cluster API, GitOps, and custom controllers. This architecture aims to reduce the potential for errors and reduce manual involvement, thereby improving operational efficiency.

Ⅰ. Introduction

The evolution of Hybrid cloud computing reflects the growing complexity of managing distributed, containerized infrastructures that span both private and Hybrid environments. As demand for more flexibility and control over data and applications, the need for advanced tools that can ensure streamlined and scalable operations across diverse cloud providers becomes essential. This scenario underscores the importance of adopting transformative architectural frameworks. Particularly, technologies like Cluster API and GitOps have emerged as promising solutions, offering precise control and automation for Hybrid cloud management.

Cluster API, a Kubernetes subproject [1], introduces a standardized mechanism to manage the lifecycle of Kubernetes clusters. It leverages Kubernetes-style APIs and patterns to foster portability and agility across different environments. It encapsulates both the complexity and versatility of managing multiple Kubernetes clusters, promoting more efficient operations and facilitating easier scaling and maintenance. This provides a significant step towards a more unified and automated management of Kubernetes environments.

GitOps, on the other hand, is a way of implementing Continuous Deployment for cloud-native applications. It focuses on a developer-centric experience when operating infrastructure, by using tools developers are already familiar with[2]. Using Git as a single source of truth for declarative infrastructure and applications, GitOps simplifies the process of updating and maintaining cloud-native applications by automating deployment pipelines, thus it gives us many benefits, including improved developer productivity, enhanced clarity, and better edit configurations for infrastructure changes.

However, a significant challenge arises when clusters are provisioned by using Cluster API, which currently requires manual steps to install necessary components such as the Container Network Interface (CNI) and the GitOps platform (e.g., ArgoCD, FluxCD). Furthermore, setting up and configuring the repository for GitOps also involves manual processes that can slow down deployments and introduce human errors.

Therefore, in this paper, we designed the architecture using these concepts aimed at addressing this gap. Through automation, we can not only accelerate the deployment of workload clusters but also improve consistency, minimize potential errors, and guarantee that the clusters are ready for immediate use. The following section dives into a detailed examination of our approach designed explicitly for managing Kubernetes over the Hybrid Cloud.

Ⅱ. Method

In this section, we explore the architecture in detail, concentrating on each component's specific functions and roles within the system.

The architecture is strategically divided into two core components: the Management Cluster and the Workload Cluster. The Management Cluster acts as the central hub of the architecture, where the principal software components are executed as Pods. This cluster integrates essential GitOps tools and Cluster API services, providing seamless connectivity with a broad array of cloud providers, such as AWS, Google Cloud, and others. The primary function of the Management Cluster is to administer and orchestrate the infrastructure, encompassing monitoring and control processes essential for initiating and seamlessly integrating GitOps practices across clusters that are provisioned on demand.

On the other hand, the Workload Clusters are instantiated on these cloud platforms and are directly managed by the configurations and policies pre-established in the Management Cluster[3]. These clusters are where the actual deployment of applications and services occurs, managed through the GitOps methodology. The configurations are synchronized continuously from the Management Cluster, ensuring that each Workload Cluster operates in alignment with the defined operational and security standards. This ensures consistent configurations and enhances security by centralizing control.

As illustrated in Figure 1, our design architecture introduces four critical components that enhance the management and operational capabilities of GitOps. The GitOpsCluster Controller plays a pivotal role, overseeing the monitoring and reconciliation processes for the GitOpsCluster CRD(Custom

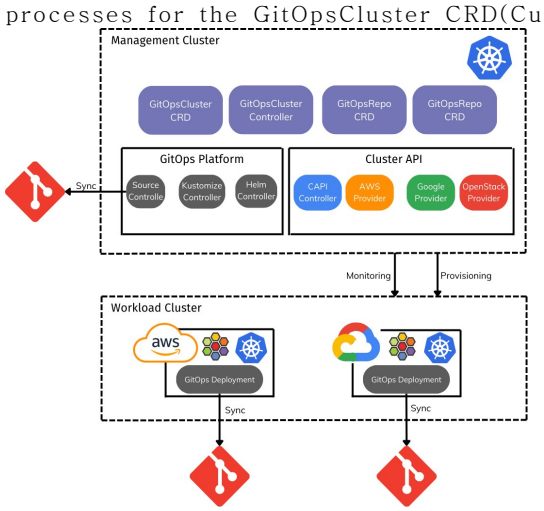

Figure 1: Diagram of our system architecture

Resource Definition). Similarly, the GitOpsRepo Controller manages GitOps deployment and Git Repository in the GitOpsRepo CRD. Detailed descriptions of these new components are provided below

```yaml
apiVersion: example/v1
kind: GitOpsCluster
metadata:
  name: mycluster
  namespace: default
spec:
  cluster:
    - name: aws-cluster
      cloudProvider:
        type: AWS
        region: ap-northeast-2
      network:
        pods:
          cidrBlocks:
          - 192.168.0.0/16
        service:
          cidrBlocks:
          - 10.50.0.0/16
      sshKeyName: kiem
    - name: google-cluster
      cloudProvider:
        cloudProvider:
        type: Google
        region: us-west1
      network:
        pods:
          cidrBlocks:
          - 172.18.0.0/16
        service:
          cidrBlocks:
          - 10.10.0.0/16
      sshKeyName: kiem
  cni:
    type: Cilium
    version: v1.15.0
  GitOps:
    type: ArgoCD
    repository: "your-url-git-repo"
    path: "/"
```

Figure 2: GitOpsCluster CRD

The GitOpsCluster controller operates by extracting essential data from the *spec.cluster* section of the GitOpsCluster Custom Resource Definition (CRD), where basic details of the clusters and associated cloud providers are specified. Each cluster configuration is under *spec.cluster* includes critical parameters such as the cloud provider type (*cloudProvider.type*), which can be AWS, Google, or other supported platforms, and regional deployment specifics (region). These configurations guide the GitOpsCluster controller in constructing resource manifests appropriate for the Cluster API and CAPI to deploy all of the clusters.

By default, when Workload Clusters is successfully created, ClusterAPI saves the *kubeconfig* file into Secret in the Management Cluster. The GitOpsCluster controller utilizes this access to coordinate with the Helm Controller to deploy the specified Container Network Interface (CNI) solution—Cilium in this case, as specified by the *cni* block of the CRD. Following the network setup, the GitOpsCluster controller deploys and config GitOps to the Workload Clusters. It does this by generating additional resources defined in the GitOpsRepo CRD as described in Figure 3

```yaml
apiVersion: example/v1
kind: GitOpsRepo
metadata:
  name: repo-mycluster
  namespace: default
  ownerReferences:
    - apiVersion: example/v1
      kind: GitOpsCluster
      name: mycluster
spec:
  type: "ArgoCD"
  repository:
    authenticate:
      secretRef:
        name: "default-credential"
    type: "Github"
      url: "your-url-git-repo"
      path: "/"
  interval: 1m0s
```

Figure 3: GitOpsRepo CRD

The GitOpsRepo Controller, managing the GitOpsRepo resource, contacts with Git repositories like GitHub or GitLab to create a folder named after the GitOpsCluster with *spec.repository.type.url*. If successful, the GitOpsRepo Controller utilizes the *kubeconfig* and Helm Controller to deploy GitOps tools, such as ArgoCD, FluxCD, or any other kind of GitOps platform, using the defined configurations.

Ⅲ. Conclusion

Our architectural design provides an automated approach to the management of Kubernetes cluster deployment. By automating cluster provisioning and configuration through custom controllers, the system ensures enhanced operational efficiency and alignment with best practices in application deployment. The architecture supports seamless multi-cloud operations, reducing complexity and human error while promoting a more secure and scalable infrastructure.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (RS-2024-00398379), Development of High Available and High Performance 6G Cross Cloud Infrastructure Technology)

REFERENCES

[1] The Cluster API Book, Kubernetes Cluster API May 16th, 2024. URL: https://cluster-api.sigs.k8s.io/
[2] GitOps, What is GitOps? May 16th, 2024. URL: https://www.gitops.tech/#what-is-gitops
[3] The Cluster API Book, Management cluster May 16th, 2024.URL: https://cluster-api.sigs.k8s.io/user/concepts