A Design of KPT based Continuous Delivery for Kubernetes Workloads over Distributed Edge Clouds

Huy Thang Tran*, Young Han Kim
Soongsil University

trhthang0401@dcn.ssu.ac.kr*, younghak@ssu.ac.kr

Abstract

In the distributed cloud environment of Kubernetes, the manual deployment of tailored clusters alongside the deployment of applications on each edge cluster can result in errors and demand significant efforts. Navigating the intricacies of managing individual edge clusters becomes particularly challenging, especially in the context of diverse applications. This paper introduces a design that harnesses the power of the KPT in tandem with extended Kubernetes Operator and GitOps. The focus of this study is to tailor cluster management, elevate automation, and enhance reproducibility specifically for Kubernetes Workloads over Distributed Edge Clouds.

Ⅰ. INTRODUCTION

Technologies like distributed cloud enable on-demand, API-driven access to the edge. Unfortunately, existing brittle, imperative, fire-and-forget orchestration methods struggle to take full advantage of the dynamic capabilities of these new infrastructure platforms. To succeed at this, Nephio uses new approaches that can handle the complexity of provisioning and managing a multi-vendor, multi-site deployment of interconnected network functions across on-demand distributed cloud [1]. Nephio aims to simplify and enhance the deployment of multi-cluster workloads [2], emphasizing GitOps principles for secure, adaptable, and robust networking solutions across diverse clusters supporting various applications.

While Nephio offers streamlined management, the absence of an automated method for developing and configuring Kubernetes clusters remains a notable gap. In response, the Kubernetes Package Tool (KPT) [3] enters the scene, designed to automate editing tasks and promote a Configuration as Data approach. By fostering a clear separation between configuration and current state, KPT simplifies the management of Kubernetes configurations and aligns with the Kubernetes Resource Model (KRM). This approach contributes to a unified ecosystem, facilitating consistent and adaptable configuration management.

Within the KPT ecosystem, a flexible and unified deployment process for Kubernetes configurations involves the integration of various tools. Package Orchestration, represented by Porch, plays a pivotal role in managing the entire lifecycle of configuration packages. Porch, functioning as a service, seamlessly aligns with workflows, offering a user-friendly experience akin to the Backstage plugin's proof-of-concept.

Cluster API, another essential component in the multi-cluster environment, streamlines Kubernetes cluster management by automating provisioning and lifecycle tasks through familiar manifests. This open-source tool enhances scalability and consistency across diverse environments.

Nevertheless, the creation of custom clusters and the deployment of software in a step-by-step manner within a multi-cluster environment pose significant challenges in terms of effort and time. Ensuring accurate and swift deployment across various clusters is a complex task.

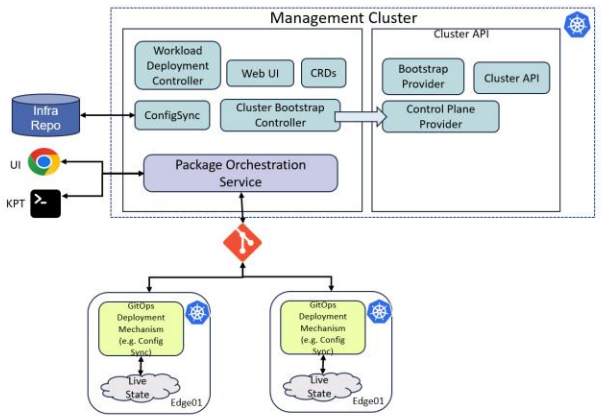
Therefore, this paper introduces a novel design for efficiently deploying custom Kubernetes clusters based on user preferences. The proposed approach leverages the KPT methodology, Cluster API, and extended Kubernetes Operators to support continuous deployment workflows. The subsequent section provides an in-depth exploration of our innovative approach, specifically tailored for Kubernetes workloads over distributed edge clouds.

Ⅱ. THE PROPOSED ARCHITECTURE

In this section, we delineate a range of components and detail their specific functions in the context of an efficient, automated framework for the deployment and ongoing management of multiple clusters. This includes the automated installation and updating of software across new and existing clusters.

Clusters are differentiated into two main types: Management Clusters and Workload Clusters. The management cluster is the hub where we coordinate the development of package declarations and the use of advanced Kubernetes operator solutions, along with leveraging the Cluster API project. This cluster acts as the control plane, directing and managing the operations and lifecycle of workload clusters.

In contrast, a Workload Cluster is where deployed workloads are coordinated by the Management Cluster. In these Workload Clusters, your software is actively run and maintained, ensuring efficient performance in real-world environments. A clear division of responsibilities between these clusters is critical to establishing a seamless, scalable architecture that competently manages and executes multiple complex workloads with relative ease.


Management Cluster
Cluster API
Workload Deployment Controller
Web UI
CRDs
Bootstrap Provider
Cluster API
Infra Repo
ConfigSync
Cluster Bootstrap Controller
Control Plane Provider
UI
Package Orchestration Service
KPT
GitOps Deployment Mechanism (e.g. Config Sync)
Live State
Edge01
GitOps Deployment Mechanism (e.g. Config Sync)
Live State
Edge01
Figure 1 Diagram of supporting functionalities

As depicted in Figure 1, the essential elements comprise Package Orchestration (Porch), expanded Custom Resource Definitions (CRDs) known as WorkloadDeployment CRD and WorkloadCluster, controllers named WorkloadDeployment controller and Cluster Bootstrap Cluster, and WebUI. The specifics of these functionalities are outlined below.

WorkloadDeploymentOperator: defines customized clusters and outlines the detailed deployment of specific packages intended for use on these newly created clusters. Users have the flexibility to specify one or multiple packages from various repositories for deployment on the recently established cluster. The WorkloadDeployment controller harmonizes the WorkloadDeployment CRD and generates two corresponding CRD types: WorkloadCluster CRD and PackageVariant CRD. Figure 2 provides an example of a WorkloadDeployment CRD.

```
apiVersion: example/v1alpha1
kind: WorkloadDeployment
metadata:
  name: workload01
spec:
  clusters:
    - cluster:
        id: 1
        name: RegionalCluster
       metadata:
         labels:
           type: regional
           region: us-west1
       spec:
         packages:
           - upstream:
               id: 1
               repo: nginx-packages
               package: pkg-nginx
               revision: v1
           - upstream:
               id: 2
               repo: redis-packages
               package: pkg-redis
               revision: v1
     - cluster:
         id: 2
         name: Edge01
       metadata:
         labels:
           type: edge
           region: us-west1
       spec:
         packages:
           - upstream:
               id: 1
               repo: prometheus-package
               package: pkg-prometheus
               revision: v1
         cni: calico
```

Figure 2 WorkloadDeployment CRD

WorkloadCluster Operator : The WorkloadCluster CRD, produced by the WorkloadDeployment controller, contains infrastructure details for a customized cluster. The Cluster Bootstrap controller collaborates with the Cluster API controller to create a cluster, currently a KinD cluster. It then independently initializes the linked Git repository for the newly created cluster, deploying ConfigSync automatically for synchronization with the cluster's Git repository. Figure 3 shows a WorkloadCluster CRD example:

```
apiVersion: example/v1alpha1
kind: WorkloadCluster
metadata:
  name: RegionalCluster
spec:
  labels:
    type: regional
    region: us-west1
cni: flannel
```

Figure 3 WorkloadCluster CRD

PackageVariant CRD and Porch: The WorkloadDeployment controller creates the PackageVariant CRD, which holds data packets orchestrated from the upstream Git repository to the downstream Git repository. This orchestration is automated by Porch. Figure 4 illustrates a PackageVariant CRD example:

```
apiVersion: config.kpt.dev/v1alpha2
kind: PackageVariant
metadata:
  name: nginx-package
spec:
  upstream:
    repo: nginx-packages
    package: pkg-nginx
    revision: v5
  downstream:
    package: pkg-nginx
    repo: RegionalCluster
```

Figure 4 PackageVariant CRD

WebUI: In addition to the CLI, providing a UI to suppport users in interacting with the system.

Ⅲ. CONCLUSION

Starting with user interaction via the WorkloadDeployment CRD, the deployment pipeline smoothly transitions to the Cluster Bootstrap phase. Led by the Cluster Bootstrap Controller and Cluster API, this phase not only sets up clusters but also automatically initializes a Git repository and deploys ConfigSync. This holistic process optimizes cluster management and automation, inherently advancing reproducibility.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. 2022-0-01015, Development of Candidate Element Technology for Intelligent 6G Mobile Core Network)

REFERENCES

[1] Nephio. About Nephio. Mar. 15, 2023. url: https://nephio.org/about

[2] Dina, Laurent, Lukas Schlunegger, and Marc Eberhard. Cloud Native Intent Automation. Diss. OST Ostschweizer Fachhochschule, 2023.

[3] KPT. KPT Concepts. Apr. 9, 2023. url: https://kpt.dev/book/02-concepts/.

[4] KPT. KPT Config Sync. June 15, 2023. url: https://kpt.dev/gitops/configsync/.