Federated Context Synchronization for Model Context Protocol Servers in Multi-Cluster Kubernetes

Huan Le, Young Han Kim*

Soongsil University

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

Abstract

In modern AI-driven applications, Model Context Protocol (MCP) servers maintain dynamic, stateful contexts—such as code, files, and configuration—that directly shape the behavior and decisions of AI agents. Deploying and managing MCP servers across multi-cluster Kubernetes environments introduces new challenges, particularly in ensuring consistent and synchronized context states. Existing Kubernetes federation tools (e.g., Karmada) can distribute static CRDs but lack support for real-time synchronization of evolving context, resulting in drift and inconsistent inference. We propose a Kubernetes-native architecture featuring two custom resources: FederatedContext on a central control plane and MCPContext on each workload cluster, and two controllers: a Federated Context Controller that propagates context updates via Karmada's PropagationPolicy, and an MCP Context Controller that applies them locally. By embedding context synchronization directly into Kubernetes CRDs and controllers, our solution eliminates external messaging layers while ensuring consistent context distribution across clusters, greatly simplifying multi-cluster MCP server management.

Ⅰ. Introduction

The Model Context Protocol (MCP) [1] is an open standard that defines a secure, two-way interface between AI models and external tools or data sources. An MCP deployment comprises hosts (e.g., IDE plugins or agent platforms), clients (LLM-backed applications), and servers, where MCP servers provide dynamic, stateful context in the form of executable code modules, configuration files, static resources such as documents and media assets, and customizable prompts that guide model interactions.

As mentioned in [5], one of the key challenges of MCP is scalability in multi-tenant environments. This means that when MCP servers are deployed across shared Kubernetes clusters, maintaining isolation, performance, and context consistency becomes increasingly difficult. Furthermore, the MCP server lifecycle, which includes creation, operation, and update phases, introduces additional complexity: during updates, unsynchronized changes to tools, resources, or prompts can lead to context drift across clusters. This drift can undermine model behavior and reliability. Although the MCP specification supports basic context operations, it does not natively address dynamic, multi-cluster synchronization, motivating the need for a federated Kubernetes-native solution.

Several open-source projects address multi-cluster orchestration for Kubernetes—most notably Karmada [2]. Karmada provides a central control plane with PropagationPolicy and ClusterPropagationPolicy CRDs to replicate static resources (Deployments, ConfigMaps, Secrets) across clusters. GitOps tools such as ArgoCD [3] and Flux [4] can likewise deploy identical manifests across clusters, but typically operate at commit cadence rather than supporting high-frequency updates. None of these solutions were designed to handle the dynamic, stateful context that MCP servers require; they lack event-driven synchronization for rapidly changing tools, resources, and prompts.

Therefore, to address the lack of dynamic, multi-cluster context synchronization for MCP servers, we propose a Kubernetes-native architecture. Our design introduces two custom resources: FederatedContext at the control plane and MCPContext within each workload cluster, and implements dedicated controllers that use Karmada's propagation engine to achieve real-time, bidirectional synchronization of MCP server contexts across clusters.

Ⅱ. Method

In this section, we present our Kubernetes-native method for real-time, bidirectional context synchronization across MCP servers in multi-cluster environments.

As illustrated in Figure 1, we first define two CRDs to represent federated and local contexts, then describe the Federated Context Controller and the MCP Context Controller. Finally, we explain how these components integrate with Karmada's propagation engine to deliver low-latency updates without external messaging layers.

A. Custom Resource Definitions

To encode context state declaratively, we introduce two Kubernetes CustomResourceDefinitions (CRDs): FederatedContext (Control Plane) and MCPContext (Workload Cluster).

The FederatedContext CRD serves as the global, source-of-truth representation of MCP context in the control-plane cluster. Its spec includes an "mcpServerName" (the target MCP server deployment), a clusters list or selector to scope propagation, a "contextType" field ("tools", "resources", "prompts"), and a "contextPayload" object containing the actual code modules, configuration files, binary artifacts, or prompt templates.

The MCPContext CRD, represents the localized context state for an MCP server within a workload cluster. It closely reflects the structure of FederatedContext, including fields such as mcpServerName (identifying the associated Deployment or StatefulSet), contextType ("tools", "resources", "prompts"), contextPayload (containing the JSON-encoded data such as code modules, configurations, binaries, or prompts), version (to track changes), and timestamp (to mark the latest update time).

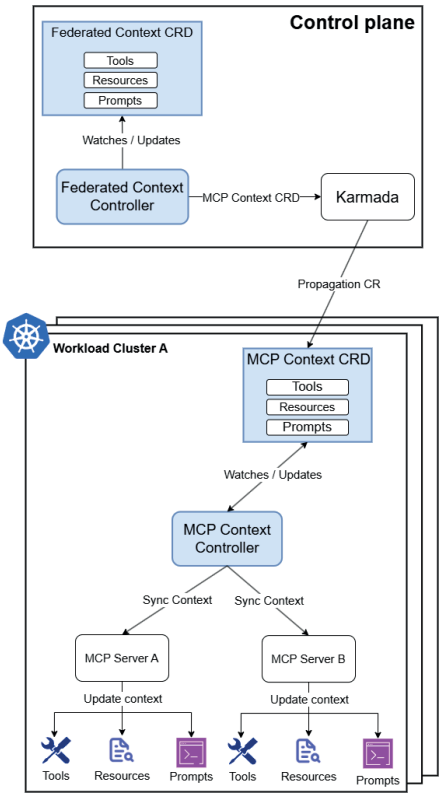

Figure 1. Proposed Architecture for MCP Server Context Synchronization.

B. Federated Context Controller

The Federated Context Controller watches Federated Context objects in the control-plane cluster and translates them into MCPContext instances for propagation. Its responsibilities are: (1) watch for changes to FederatedContext resources and translate each spec into a corresponding MCPContext CR, (2) generate or update a PropagationPolicy CR that selects the new MCPContext and targets the specified clusters, and (3) monitor propagation progress and update the FederatedContext status (Pending, Applied, or Error) accordingly.

C. MCP Context Controller

The MCP Context Controller runs in each workload cluster and is responsible for managing local MCPContext resources. Its primary tasks include: (1) applying the MCPContext data to the corresponding MCP server instance, such as updating local files, configurations, or runtime states; (2) monitoring the MCP server for any local context changes during the server's operation or update phases; and (3) when a local update occurs, using a remote Kubernetes API client to patch the FederatedContext in the control-plane cluster with the new context version and timestamp. This ensures bidirectional synchronization and maintains consistency across all clusters.

Ⅲ. Conclusion

In this paper, we addressed the challenge of dynamic, multi-cluster synchronization of MCP server contexts in Kubernetes environments. Building on existing federation tools, we proposed a Kubernetes-native architecture featuring two custom resources—FederatedContext and MCPContext—and designed dedicated controllers for each layer. By leveraging Karmada's propagation engine and implementing a remote patching mechanism, our architecture ensures real-time, bidirectional synchronization of MCP server contexts across clusters. In future work, we aim to extend our design with conflict resolution strategies, consistency guarantees, and security enhancements for cross-cluster synchronization.

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] *Introduction - Model context protocol. (n.d.-b). Model Context Protocol.* https://modelcontextprotocol.io/

[2] *Open, Multi-Cloud, Multi-Cluster Kubernetes Orchestration | karmada.* (n.d.). https://karmada.io/

[3] *ARGO CD - Declarative GITOPS CD for kubernetes.* (n.d.). https://argo-cd.readthedocs.io/en/stable/

[4] *Flux - the GitOps family of projects.* (n.d.). https://fluxcd.io/

[5] Hou, X., Zhao, Y., Wang, S., & Wang, H. (2025, March 30). *Model Context Protocol (MCP): landscape, security threats, and future research directions.* arXiv.org. https://arxiv.org/abs/2503.23278

[6] *Custom resources.* (2024, October 31). Kubernetes. https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/