TECHNICAL WHITE PAPER: February 2026

Sandbox MCP Servers on vSphere Kubernetes Service with Cosmonic Control

Reference Architecture

Table of contents

Executive Summary

As AI agents become increasingly integrated into enterprise workflows, organizations face critical security challenges in deploying Model Context Protocol (MCP) servers. MCP servers enable AI models to interact with external tools and data sources, but this capability introduces significant risks including prompt injection attacks, unauthorized data exfiltration, resource exhaustion, and cross-tenant vulnerabilities when servers operate on internal networks.

Cosmonic Control addresses these challenges by leveraging WebAssembly (Wasm) sandboxing to create secure, isolated execution environments for MCP servers. Wasm component binaries are portable, polyglot sandboxes that interact with the outside world via explicitly enabled, language-agnostic interfaces. This architecture ensures AI agents can only access approved tools and resources in approved ways, with virtualizable interfaces—such as virtual filesystems—that strictly control visibility and I/O behavior, preventing unauthorized access to sensitive resources.

When deployed on VMware vSphere Kubernetes Service (VKS), organizations gain the combined benefits of enterprise-grade Kubernetes infrastructure and secure agent execution:

- **Lower TCO**: Reduce operational silos by leveraging existing vSphere tools, skills, and processes. Unified lifecycle management keeps infrastructure current with the latest security patches while minimizing risk.
- **Operational Simplicity**: VKS streamlines Kubernetes management through automated cluster provisioning, upgrades, and lifecycle management—freeing teams to focus on innovation rather than infrastructure maintenance.
- **Scalable Security**: Deploy CNCF-certified Kubernetes clusters at scale with built-in security controls, while Cosmonic Control's capability-based security model ensures each MCP server operates within strictly defined boundaries.
- **Cloud-Native Integration**: Wasm components are highly portable and integrate seamlessly with Kubernetes environments and GitOps workflows, enabling consistent deployment patterns from development through production.

This reference architecture provides a validated approach for deploying sandboxed MCP servers on VKS with Cosmonic Control, enabling organizations to harness the power of AI agents while maintaining enterprise security standards.

vSphere Kubernetes Service

vSphere Kubernetes Service (VKS) is the Kubernetes runtime built directly into VMware Cloud Foundation (VCF). With CNCF certified Kubernetes, VKS enables platform engineers to deploy and manage Kubernetes clusters while leveraging a comprehensive set of cloud services in VCF. Cloud admins benefit from the support for N-2 Kubernetes versions, enterprise grade security, and simplified lifecycle management for modern apps adoption.

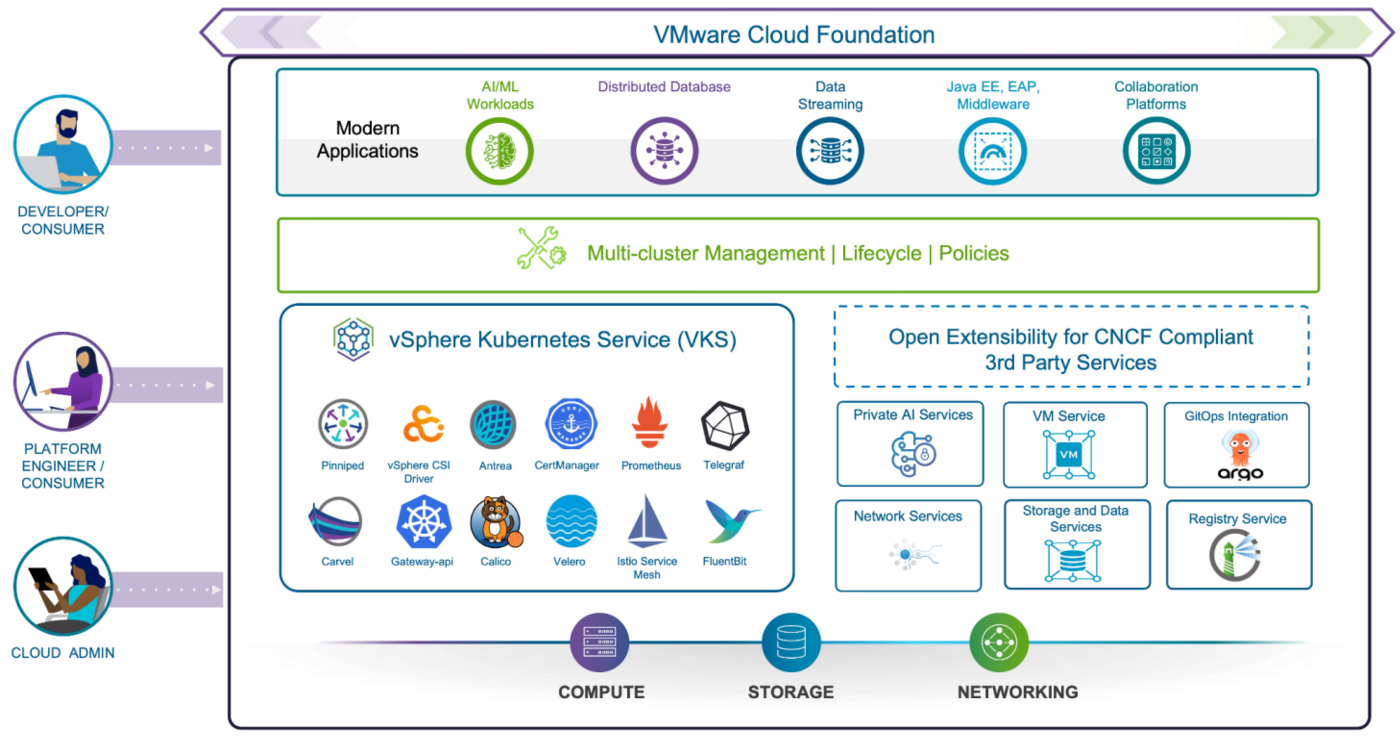

Figure 1: VKS on VCF Ecosystem

Cosmonic Control

Cosmonic Control is a Kubernetes-native platform for deploying and managing WebAssembly workloads alongside traditional container-based applications. Built on the Cloud Native Computing Foundation (CNCF) wasmCloud project, Cosmonic Control enables organizations to run secure, sandboxed workloads that leverage Wasm's architectural security guarantees while integrating seamlessly with existing Kubernetes infrastructure.

Key Capabilities

WebAssembly Sandboxing: Cosmonic Control leverages the inherent security properties of WebAssembly components. Unlike traditional containers, Wasm components can only interact with the outside world via explicitly granted capabilities. This "deny-by-default" security model prevents unauthorized operations and ensures that workloads cannot access resources beyond their defined scope.

Capability-Based Security: The platform implements strict capability enforcement through language-agnostic interfaces. Administrators define precisely which tools, APIs, and resources each workload can access. Virtualizable interfaces—including virtual filesystems and network policies—strictly control visibility and I/O behavior, preventing unauthorized access to sensitive resources.

Kubernetes Integration: Cosmonic Control deploys as a Helm chart and integrates with standard Kubernetes primitives. Workloads are deployed using custom resources (such as HTTPTrigger) that map to familiar Kubernetes patterns. The platform includes an Envoy-based ingress for routing traffic to Wasm workloads via xDS configuration.

Observability: Built-in support for OpenTelemetry enables comprehensive monitoring and tracing of Wasm workloads. The Cosmonic Perses UI provides visualization and querying capabilities for operational insights.

GitOps Ready: Wasm components are distributed as OCI artifacts, enabling version-controlled deployments through standard GitOps workflows with tools like ArgoCD.

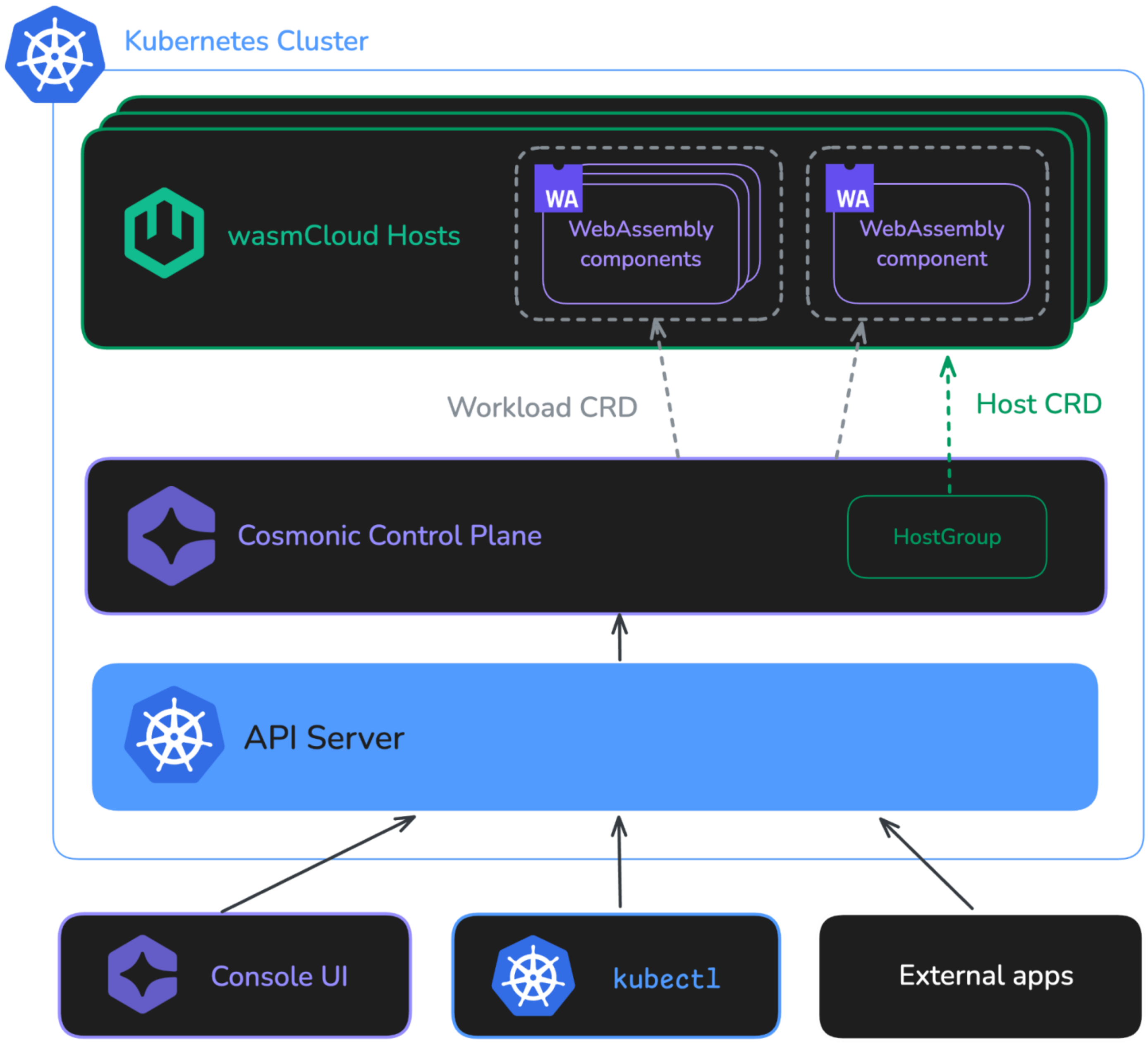

Figure 2: Cosmonic Control Workflow Example

Solution Architecture

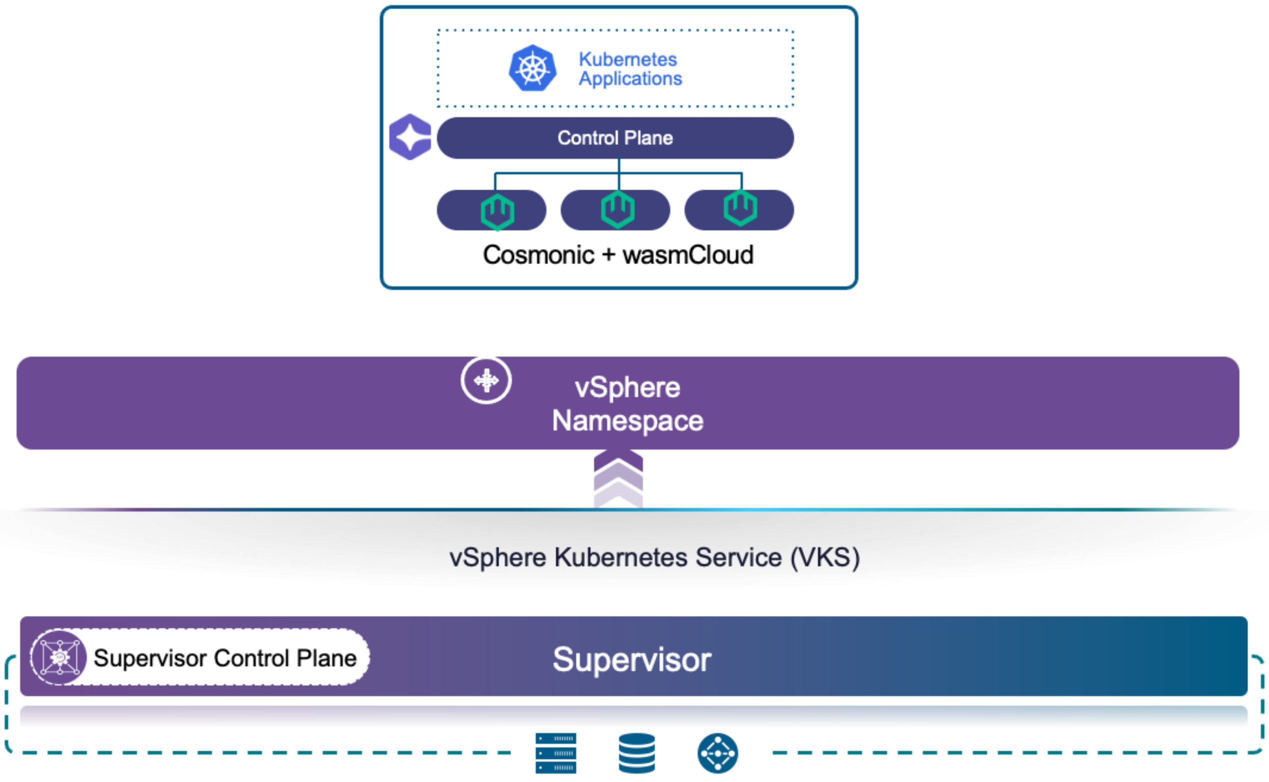

Figure 3: Cosmonic Control on VKS

Component	Version	Notes
vSphere Kubernetes Service	3.5	
vSphere Cloud Foundation	9.0	
Kubernetes	1.34	vKR Version
Cosmonic Control	1.0	
Contour	1.33.0+vmware.1-vks.1	Ingress controller
Cert-manager	1.18.2+vmware.2-vks.2	TLS certificate manager

Solution Validation

Install Cosmonic Control on Kubernetes

This section provides instructions for installing Cosmonic Control on a VKS cluster, enabling the cluster to run both traditional pod deployments and WebAssembly workloads.

Prerequisites

Before installing Cosmonic Control, ensure the following components are configured:

- **VKS 3.5 or later**: Required for add-on support
- **VCF CLI**: VKS cluster management with add-ons
- **Contour**: Configured as the ingress controller
- **Cert-Manager**: Installed for TLS support (recommended)
- **Cosmonic License Key**: Obtain a trial or production license from Cosmonic

Step 1: Configure VKS Add-ons Repository

Add the VKS standard add-ons repository to enable installation of required components:

```
vcf addon repository add vks-repo \
  --url projects.packages.broadcom.com/vsphere/supervisor/packages/2025.10.22/vks-standard-packages:3.5.0-
20251022 \
  -n packages
```

Step 2: Install Cert-Manager

Install Cert-Manager for automated TLS certificate management:

```
vcf addon install cert-manager \
  -p cert-manager.kubernetes.vmware.com \
  --version 1.18.2+vmware.2-vks.2 \
  -n packages
```

Step 3: Install Contour Ingress Controller

Retrieve the default Contour configuration values:

```
vcf addon available get contour.kubernetes.vmware.com/1.33.0+vmware.1-vks.1 \
  --default-values-file-output contour-data-values.yaml \
  -n packages
```

Customize the contour-data-values.yaml file as needed for your environment, then install Contour:

```
vcf addon install contour \
  -p contour.kubernetes.vmware.com \
  --version 1.33.0+vmware.1-vks.1 \
  --values-file contour-data-values.yaml \
  -n packages
```

For detailed Contour configuration options, refer to the VMware VCF documentation.

Step 4: Install Cosmonic Control

Deploy Cosmonic Control using Helm. For a basic installation:

```
helm install cosmonic-control oci://ghcr.io/cosmonic/cosmonic-control \
  --version 0.3.0 \
  --namespace cosmonic-system \
  --create-namespace \
  --set envoy.service.type=ClusterIP \
  --set cosmonicLicenseKey="<your-license-key>"
```

Step 5: Install Cosmonic Control Hostgroup

Deploy a Hostgroup to provide compute resources for Wasm workloads:

```
helm install hostgroup oci://ghcr.io/cosmonic/cosmonic-control-hostgroup \
  --version 0.3.0 \
  --namespace cosmonic-system
```

Step 6: Configure Ingress Routes

Create ingress configurations to route traffic to Cosmonic Control services. Example configurations include:

- Console and Perses UI Ingress: Routes traffic to the management interfaces

- **Wasm Workload Ingress**: Routes application traffic to deployed Wasm workloads via the Cosmonic Envoy service

For production environments, configure specific hostnames rather than wildcards to reduce invalid hostname requests.

Verification

Verify the installation by checking that all pods in the cosmonic-system namespace are running:

```
kubectl get pods -n cosmonic-system
```

Sandbox MCP Servers with Cosmonic Control

This section describes how to create, build, and deploy sandboxed MCP servers to Cosmonic Control on your VKS cluster.

Prerequisites

Ensure the following tools are installed on your development workstation:

- **Node.js and npm**: Required for the MCP server template
- **wash v2.0.0-rc.6 or later**: WebAssembly Shell for building and managing components
- **kubectl**: Configured to access your VKS cluster
- **Helm v3.8.0 or later**: For deploying workloads

Step 1: Install OpenAPI2MCP

Install the OpenAPI2MCP tool, which generates TypeScript-based MCP servers from OpenAPI specifications:

```
npm install -g openapi2mcp
```

Step 2: Create an MCP Server Project

Clone the MCP server template:

```
git clone https://github.com/cosmonic-labs/mcp-server-template-ts.git my-mcp-server
cd my-mcp-server
```

If you have an OpenAPI specification for your API, generate MCP tools from it:

```
openapi2mcp your-api-spec.json --project-path .
```

Step 3: Local Development and Testing

Start the local development server:

```
wash dev
```

This launches the MCP server at http://127.0.0.1:8000/v1/mcp and opens the MCP Model Inspector in your browser for testing.

To test the MCP server with an AI model locally, you can use tools such as:

- **Goose**: Configure a remote extension pointing to http://127.0.0.1:8000/v1/mcp
- **ngrok**: Create a tunnel to expose your local server to cloud-based AI services

Step 4: Build the WebAssembly Component

Compile your MCP server to a WebAssembly component:

```
wash build
```

The compiled component appears in the ./dist directory.

Step 5: Publish to a Container Registry

Push the compiled Wasm component to an OCI-compatible registry:

```
wash oci push ghcr.io/<your-namespace>/<project-name>:0.1.0 ./dist/component.wasm
```

Note: If using GitHub Container Registry (GHCR), the uploaded artifact defaults to private visibility. Update the package settings to make it public if needed for deployment.

Step 6: Deploy to Cosmonic Control

Deploy your MCP server to the VKS cluster using the HTTPTrigger Helm chart. Create a values file (values.yaml) with your configuration:

```
components:

  - name: my-mcp-server
    image: ghcr.io/<your-namespace>/<project-name>:<tag>

ingress:
  host: my-mcp-server.your-domain.com

pathNote: v1/mcp
```

Deploy the workload:

```
helm install my-mcp-server \
  --version 0.1.2 \
  oci://ghcr.io/cosmonic-labs/charts/http-trigger \
  -f values.yaml
```

Step 7: Connect and Verify

Test connectivity to your deployed MCP server using the MCP Model Inspector:

```
npx @modelcontextprotocol/inspector
```

Configure the connection with:

- **Transport Type**: Streamable HTTP
- **URL**: http://my-mcp-server.your-domain.com/v1/mcp

Verify that tools and resources are accessible through the inspector interface.

Security Considerations

Wasm component binaries deployed through Cosmonic Control operate within a secure sandbox:

- Components can only access capabilities explicitly granted during deployment
- Virtual filesystems prevent access to host system resources
- Network access is controlled through capability configuration
- Resource consumption is bounded by the Wasm runtime

This architecture ensures that even if an MCP server is compromised through prompt injection or other attacks, the blast radius is contained within the sandbox boundaries.

Conclusion

Deploying MCP servers on vSphere Kubernetes Service with Cosmonic Control provides a robust, enterprise-ready solution for organizations seeking to leverage AI agents while maintaining stringent security controls.

By combining VKS's operational simplicity and enterprise-grade Kubernetes infrastructure with Cosmonic Control's WebAssembly sandboxing, organizations achieve:

- **Defense in depth**: Multiple layers of security—from Kubernetes RBAC to Wasm capability-based isolation—protect against the evolving threat landscape of agentic AI systems.
- **Operational efficiency**: Platform teams can manage MCP server deployments using familiar Kubernetes tooling and GitOps workflows, reducing the learning curve and operational overhead.
- **Scalable architecture**: The solution scales from development environments to production deployments, with consistent security guarantees across all stages.
- **Future-ready infrastructure**: As AI agent capabilities expand, the capability-based security model ensures new tools and resources can be safely integrated without compromising existing security postures.

This reference architecture provides a validated foundation for organizations ready to deploy AI agents in production environments, balancing the transformative potential of MCP-enabled AI with the security requirements of enterprise infrastructure.

For additional resources and examples, refer to:

- Cosmonic Control Documentation
- Cosmonic Labs Control Demos Repository
- VMware VKS Documentation