Cavisson Application Agent Installer

For Kubernetes

Table of Content

Overview

This comprehensive guide outlines the complete process of authenticating with Azure, building container images, deploying applications, and configuring Cavisson webhook injector for automatic agent injection in a Kubernetes environment.

Agent to Controller Communication Architecture

The architecture demonstrates the communication flow between application pods and the Cavisson controller:

- **Webhook Injector**: Mutating admission controller that intercepts pod creation requests
- **Application Pods**: Receive injected environment variables and agent libraries
- **Communication Flow**: Pods → Cavisson Controller

The webhook automatically injects agent configuration during pod creation, enabling seamless monitoring without manual instrumentation.

ND Webhook Injector Setup

Prerequisites:

1. **Install Helm on Cavisson controller**

   Verify Helm installation or install if not present:

   curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
   helm version

2. **Connectivity between the Cavisson controller and the Kubernetes cluster where the application resides**

Installation Steps:

Step 1: Download cavisson helm chart on controller for installing webhook

tar -xvzf cav-helm-chart.tar.gz.
cd helm

Note: Updated tar file will be given by your Cavisson representative

Step 2: Configure cav-values.yaml

Navigate to the webhook-injector directory and edit cav-values.yaml. Update the env section of the appropriate agent, for e.g. nodeJS or Java with the following details:

env:
- name: CAV_APP_AGENT_NDCHOST
  value: "netcloud4-15-u24.cav-test.com" (mention the controller domain/IP)
- name: CAV_APP_AGENT_NDCPORT
  value: "4433" (mention the API gateway port of Cavisson controller)
- name: CAV_APP_AGENT_NDC_COMM_PROTOCOL
  value: "WSS"

The following snippet should be added in the matching rules in the cav-values.yaml file for specific agents as per the services

- name: cav-<agent_type>-agent
  matchRules:
    namespaceRegex: .*
    podNameRegex: ^(currencyservice|paymentservice)-
  injectionRules:
    template: nodejs-default

Note: Simply mention the deployment name of the service on which the application agent is to be injected under the podNameRegex field as highlighted above. You can enter regular expressions to specify multiple services.

Important Configuration Parameters:

Parameter	**Value**
Agent and injector images	Latest matching image versions

DockerHub agent - https://hub.docker.com/r/cavissonsystem/<certified-agent-image>

DockerHub Injector - https://hub.docker.com/r/cavissonsystem/nd_webhook_injector

Note: You can share your artifactory URL for uploading the agent related images.

Instrumentation and Injection rules

matching labels	Must match application labels
namespace	Target namespace for instrumentation

Table 1: cav-values.yaml Configuration Parameters

Step 3: Install Webhook Injector Using Helm

Option 1: Helm install cavisson webhook

helm install cavisson-wh . --namespace cavisson --values cav-values.yaml

Expected Output:
NAME: cavisson-wh
LAST DEPLOYED: Wed Dec 24 01:26:25 2025
NAMESPACE: cavisson
STATUS: deployed
REVISION: 1
TEST SUITE: None

Step 4: Verify Webhook Injector Deployment
Confirm all webhook components are running successfully:

kubectl get pods -n cavisson
kubectl get deployment -n cavisson
kubectl get svc -n cavisson

Expected Status: All pods should show STATUS as "Running"

Webhook Injector Logs Verification
Check webhook server logs to verify mutation operations:

kubectl logs -l app=webhook-server -n cavisson-wh

Application-Level Deployment Configuration

Deployment Rollout and Application Restart
Trigger a rolling restart to ensure webhook injection:

kubectl rollout restart deployment <app-deployment-name> -n <namespace>

Example:
kubectl rollout restart deployment eshopwebmvc-alpine -n cavisson

Why Restart is Required: The webhook injector only mutates pods during creation. Existing pods must be recreated for injection to occur.

Post-Deployment Validation

Step 1: Access Application Pod
Execute shell inside the running application pod:

kubectl exec -it <app-pod-name> -n cavisson -- sh

Example:
kubectl exec -it eshopwebmvc-alpine-7d8f9c6b5d-abc12 -n cavisson -- sh

Step 2: Verify Environment Variable Injection

Run the following commands inside the pod to verify successful agent injection:

Check Cavisson Variables:
env | grep CAV_

Expected Output:
CAV_APP_AGENT_NDCHOST=netcloud4-15-u24.cav-test.com
CAV_APP_AGENT_NDCPORT=4433
CAV_APP_AGENT_NDC_COMM_PROTOCOL=WSS

Validation Success Criteria

Successful validation confirms:

- Webhook injector correctly configured and operational
- Application pods successfully mutated during creation
- Cavisson proxy connection parameters properly injected
- .NET profiler enabled with correct library paths
- Application ready for performance monitoring

Critical Configuration Rules

Rule 1: The Restart Requirement

The webhook injector **only mutates pods at creation time**. After changing labels or `cav-values.yaml`:

1. Delete existing deployment
2. Reapply deployment YAML

kubectl delete deployment <app-name> -n <namespace>
kubectl apply -f <deployment-yaml>

Or use rollout restart:
kubectl rollout restart deployment <app-name> -n <namespace>

Rule 2: Namespace Consistency
Ensure webhook instrumentation rules target the correct namespace where applications are deployed.

Troubleshooting Guide

Common Issues and Solutions

Issue	Cause	Solution
No environment variables injected	Label mismatch	Verify labels in app YAML match cav-values.yaml
Webhook not mutating pods	Webhook not running	Check webhook pod status: kubectl get pods -n cavisson-wh
Changes not taking effect	Pods not recreated	Run kubectl rollout restart deployment

Table 2: Common Issues and Resolutions

Debugging Commands
Check webhook logs:
kubectl logs -l app=webhook-server -n cavisson-wh

List all services to find proxy IP:
kubectl get svc -A

Describe pod for detailed events:
kubectl describe pod <pod-name> -n cavisson

Cleanup and Uninstallation

Remove the webhook injector Helm release via:

helm uninstall cavisson-wh --namespace cavisson

Use rollout restart:
kubectl rollout restart deployment <app-name> -n <namespace>

Delete Namespaces
kubectl delete namespace cavisson

Complete Deployment Flow Summary

1. **Webhook Injector Installation**
    - Install Helm (if needed)
    - Download webhook-injector.tar.gz
    - Extract and navigate to directory
    - Configure cav-values.yaml (NDController section)
    - Install via Helm with --create-namespace
    - Verify webhook pod status
2. **Deploy and Validate**
    - Apply deployment: kubectl apply
    - Rollout restart: kubectl rollout restart
    - Verify logs: kubectl logs
    - Exec into pod: kubectl exec
    - Check environment variables: env | grep CAV_, CORECLR, DOTNET

Key Takeaways

- **kubectl logs** → Debug pods and webhook operations
- **kubectl exec** → Enter pod to verify injection
- **Restart** → Required after configuration changes
- **Webhook** → Automatically injects agent during pod creation
- **NDController** → Configuration section in cav-values.yaml
- **Validation** → Always verify environment variables post-deployment

Useful Commands Reference

Namespace Operations

kubectl create namespace <namespace-name>
kubectl get namespaces
kubectl delete namespace <namespace-name>

Deployment Operations

kubectl apply -f <yaml-file>
kubectl get deployment -n <namespace>

```
kubectl delete deployment <name> -n <namespace>
kubectl rollout restart deployment <name> -n <namespace>
kubectl rollout status deployment <name> -n <namespace>
```

Pod Operations

```
kubectl get pods -n <namespace>
kubectl describe pod <pod-name> -n <namespace>
kubectl logs <pod-name> -n <namespace>
kubectl exec -it <pod-name> -n <namespace> -- sh
```

Service Operations

```
kubectl get svc -A
kubectl get svc -n <namespace>
kubectl describe svc <service-name> -n <namespace>
```

Helm Operations

```
helm list -n <namespace>
helm install <release-name> <chart> -n <namespace> --values <values-file>
helm upgrade <release-name> <chart> -n <namespace> --values <values-file>
helm uninstall <release-name> -n <namespace>
```

Conclusion

This guide provides a comprehensive workflow for deploying applications with automatic Cavisson agent injection in Kubernetes. Following these steps ensures proper configuration of the webhook injector, successful agent injection, and reliable communication between application pods and the Cavisson controller for performance monitoring and diagnostics.

Remember: Always verify environment variables after deployment to confirm successful agent injection. Label matching between application deployments and webhook configuration is critical for successful instrumentation.