

Vault Kafka CSI Integration SDD

1. Purpose

This document describes the shared design for loading Kafka secrets from Vault through the Kubernetes Secrets Store CSI Driver across Avalara oneStudio workloads.

It covers:

- the existing `services` implementation used as the reference pattern
- the `runtime` rollout for the seven .NET services
- the namespace-aware Vault path behavior for `one-studio` and `one-studio-runtime`
- CI/CD safeguards that skip `SecretProviderClass` deployment when the Vault secret path does not exist
- fallback behavior to existing configuration sources when mounted secrets are unavailable
- the planned second-pass shape for `nodered_engine` and `system_tasks`

2. Scope

In scope

- `services` Kafka Vault CSI pattern used as the baseline
- `runtime` .NET services:
 - `scheduler`
 - `wf_executor`
 - `router_receiver`
 - `runtime_agent`
 - `connection`
 - `scheduledapp_worker`
 - `refreshtoken_worker`
- Phoenix deployment changes for CSI volume mounts and `SecretProviderClass`
- GitLab secrets deployment jobs
- fallback behavior when secrets do not exist

Out of scope for this phase

- `runtime/backend/dispatcher/system_tasks`
- `runtime/backend/nodered_engine`

Those two workloads do not use the same .NET `PlatformSetting` startup pattern as the seven runtime services above and need a different second-pass implementation.

3. Problem Statement

Historically, Kafka connection settings were loaded from application configuration sources such as `PlatformSetting` or environment variables. That model works, but it leaves Kafka credentials and bootstrap server management coupled to app config storage.

The target design is to:

1. let Kubernetes mount Kafka secrets from Vault as files
2. read those files at startup
3. overlay Kafka configuration into the application configuration object before Kafka clients are registered
4. preserve existing behavior when secrets are unavailable

4. High-Level Design

4.1 Shared pattern

The CSI driver mounts three files into the pod:

- `kafka-access-key`
- `kafka-secret-key`
- `kafka-bootstrap-server`

The application reads them from `/mnt/secrets` and, only when all three are present, replaces:

- `BootstrapServers`
- `SaslUsername`
- `SaslPassword`

on the in-memory Kafka configuration object.

If any file is missing, the application keeps the original values that came from `PlatformSetting`.

4.2 Runtime path rules

The runtime rollout uses namespace-aware Vault paths:

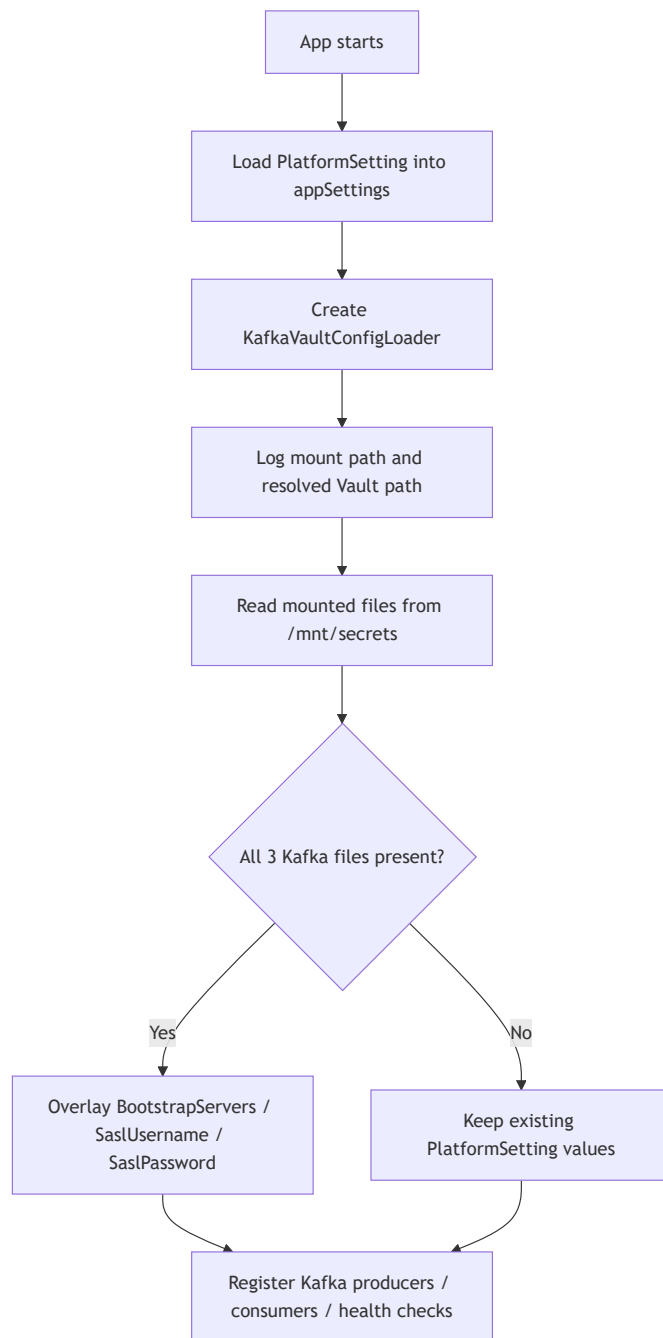
- `one-studio` -> `secrets/data/cip/integrationplatform/one-studio/<region>/services/kafka`
- `one-studio-runtime` -> `secrets/data/cip/integrationplatform/one-studio-runtime/<region>/runtime/kafka`

The runtime code logs which path was selected and whether it came from:

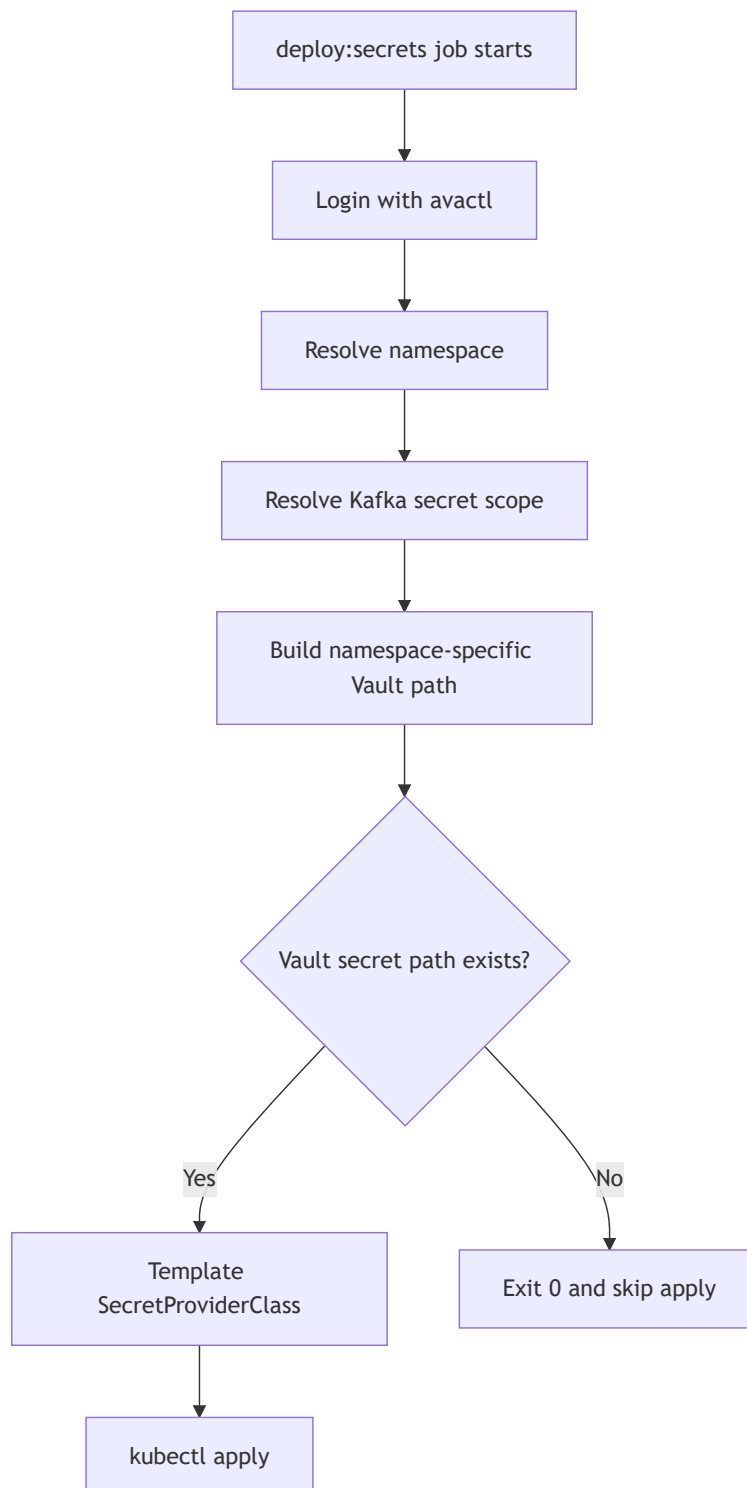
- direct override via `VAULT_KAFKA_SECRET_PATH`, or
- derived namespace/region/scope logic

5. Architecture Diagrams

5.1 Runtime startup flow



5.2 CI/CD secrets deployment flow



6. Services Repository Pattern

The `services` repository established the baseline pattern:

- a shared `KafkaVaultConfigLoader`
- Kafka-only scope
- CSI mount at `/mnt/secrets`
- `SecretProviderClass` templated with:
 - namespace
 - AWS region
- CI job validation that checks the Vault path before applying the SPC

The `services` path shape remains:

- `secrets/data/cip/integrationplatform/<namespace>/<region>/services/kafka`

That repository does not need the `runtime` vs `services` split because it only targets the `services/kafka` scope.

7. Runtime Repository Implementation

7.1 Shared loader

Runtime uses a shared file:

- `runtime/common/Vault/KafkaVaultConfigLoader.cs`

Responsibilities:

- log startup using `Step-1` , `Step-2` , etc.
- log the mount path
- log the Kafka Vault path source
- read mounted Kafka secret files
- apply Kafka settings only when all required files exist
- log missing files without exposing secret values
- fall back to the already-loaded `PlatformSetting` values when files are missing

7.2 Injection point

For each .NET runtime service, the loader is invoked immediately after `appSettings` deserialization and before Kafka registration and Kafka health check wiring.

That placement is important because downstream code continues to use the same `appSettings` object.

7.3 Runtime .NET services updated

The following services were updated to overlay both producer and consumer config:

- `scheduler`
- `wf_executor`
- `router_receiver`
- `runtime_agent`
- `connection`
- `scheduledapp_worker`
- `refreshtoken_worker`

7.4 What gets overridden

When the mounted secrets are present, the loader updates:

- `MessageQueueProducerConfig.BootstrapServers`
- `MessageQueueProducerConfig.SaslUsername`
- `MessageQueueProducerConfig.SaslPassword`
- `MessageQueueConsumerConfig.BootstrapServers`
- `MessageQueueConsumerConfig.SaslUsername`
- `MessageQueueConsumerConfig.SaslPassword`

This ensures all existing Kafka usage that reads from `appSettings` transparently picks up Vault-backed values.

8. Kubernetes and Phoenix Changes

8.1 CSI mount

Each targeted runtime service received:

- a CSI-backed volume
- a mount at `/mnt/secrets`

8.2 SecretProviderClass

Runtime defines:

- `runtime/sre/deployments/phoenix/secret-provider-class.yml`

with templated values for:

- `__NAMESPACE__`
- `__GT_AWS_REGION_NAME__`
- `__KAFKA_SECRET_SCOPE__`

The Kafka scope is:

- `services` for `one-studio`
- `runtime` for `one-studio-runtime`

9. GitLab Secrets Deployment Guard

Runtime now mirrors the `services` repository guard in:

- `runtime/sre/deployments/phoenix/.gitlab_jobs.yml`

Behavior:

1. determine namespace
2. determine Kafka secret scope
3. build the expected Vault secret path
4. use `avactl secrets read` to verify the path exists
5. if the path exists, apply the `SecretProviderClass`
6. if the path does not exist, print a message and exit successfully without applying the SPC

This prevents deploying an SPC that points at a nonexistent Kafka path.

10. Fallback Behavior

10.1 Pipeline fallback

If the Vault path does not exist:

- the secrets deploy job exits `0`
- the pipeline continues
- the `SecretProviderClass` is not applied

10.2 Application fallback

If the mounted Kafka files do not exist or are incomplete:

- the application logs which files are missing
- `KafkaVaultConfigLoader.TryApply(...)` returns `false`
- the original `PlatformSetting` values remain in effect

This means Kafka continues to work with existing configuration sources even when Vault-backed files are absent.

11. Logging Design

Runtime keeps the step-based startup log format for easy diagnosis.

Examples:

- `Step-1` : initialize Vault CSI loader and mount path
- `Step-2` : log Kafka Vault path source
- later steps: read each secret file and report presence or absence

The log messages intentionally expose:

- mount path
- derived or overridden Vault path
- file existence
- string length and trimming metadata

They intentionally do **not** expose actual secret values.

12. Test Coverage

Focused unit tests were added for the runtime shared loader to cover:

- direct `VAULT_KAFKA_SECRET_PATH` override
- `one-studio-runtime` resolving to `runtime/kafka`
- `one-studio` resolving to `services/kafka`
- successful apply when all Kafka secrets exist
- fallback and missing-file logging when one or more secrets are absent

13. Verification Summary

Verified during implementation:

- all seven targeted runtime .NET services build successfully
- focused loader tests pass
- runtime secrets deployment jobs support both:
 - `one-studio`
 - `one-studio-runtime`
- runtime secrets job now skips SPC apply when the namespace-specific Kafka path is absent

14. Second-Pass Design Notes

14.1 system_tasks

`system_tasks` still reads Kafka config directly from environment variables rather than `PlatformSetting`.

Second-pass work will need:

- CSI mount and namespace-aware SPC behavior
- a startup overlay or environment export approach appropriate for that service

14.2 nodered_engine

`nodered_engine` is JavaScript-based and cannot use the .NET loader.

Expected second-pass shape:

1. keep the same CSI mount and namespace-aware secret path
2. read mounted files before Node-RED starts
3. export them into Kafka env vars such as:
 - `KAFKA_BROKERS`
 - `KAFKA_SASL_USERNAME`
 - `KAFKA_SASL_PASSWORD`
4. preserve current env-based fallback when files are missing

The likely implementation point is a startup wrapper around the current `node-red -s ./settings.js` launch flow.

15. Key Invariants

- Kafka secret loading must never block the app from starting when Vault files are absent.
- Missing Vault secret paths must not fail the secrets deploy pipeline.
- Namespace determines Kafka scope in runtime:
 - `one-studio` -> `services`
 - `one-studio-runtime` -> `runtime`
- No log line may expose raw Kafka secret values.
- The overlay must happen before Kafka clients, Kafka health checks, or downstream services consume `appSettings`.

