CHECKLIST

Checklist: Secure .NET API Secrets with Azure Key Vault Integration

A practical checklist for validating secret handling, access control, rotation, and production readiness for .NET APIs using Azure Key Vault or an equivalent managed secret store.

Secure .NET API Secrets with Azure Key Vault Integration

Checklist

Use this checklist to validate a .NET API secret-management pattern before production rollout.

---

1) Confirm what should be treated as a secret

- [ ] Identify every value the API uses to authenticate to another system.
- [ ] Classify each value as secret or non-secret.
- [ ] Keep ordinary settings out of the vault when they do not require secrecy.
- [ ] Remove any secret values from source control, appsettings files, container images, and build artifacts.

2) Define the runtime access path

- [ ] Choose the identity the application will use at runtime.
- [ ] Prefer a managed identity where the hosting environment supports it.
- [ ] If managed identity is not available, document the approved fallback identity method.
- [ ] Confirm the application can authenticate without embedding credentials in code.
- [ ] Verify the vault endpoint, secret name, and environment-specific configuration are all non-secret.

3) Verify access control

- [ ] Grant the application only the minimum permission needed to read the required secret.
- [ ] Avoid broad vault access if secret-level access is sufficient.
- [ ] Confirm developers, operators, and automation each have the right level of access.
- [ ] Review whether access is controlled by policy, role-based access control, or both.
- [ ] Document who can create, rotate, read, and delete secrets.

4) Validate secret retrieval at runtime

- [ ] Start the application with only non-secret configuration present locally.
- [ ] Confirm the service resolves the secret from the vault at runtime.
- [ ] Verify the application uses the secret in memory and does not write it back to disk.
- [ ] Test startup behavior if the vault is unavailable.
- [ ] Decide whether the service should fail closed or tolerate cached values for a limited time.

5) Check rotation behavior

- [ ] Rotate one secret in a non-production environment.
- [ ] Confirm whether the application reads the secret once, refreshes periodically, or reloads on configuration change.
- [ ] Verify the app picks up the rotated value according to its intended behavior.
- [ ] Confirm old secret values are retired on schedule.
- [ ] Ensure rotation does not require rebuilding or repackaging the application.

6) Review deployment and environment hygiene

- [ ] Verify no secret appears in CI logs, release notes, or deployment manifests.
- [ ] Confirm environment variables are not being used as long-lived secret storage unless explicitly required.
- [ ] Ensure container images do not contain secrets baked into layers.
- [ ] Check local developer workflows for accidental copying of production secrets.
- [ ] Validate that test, staging, and production use separate secret values and access boundaries.

7) Inspect logging and telemetry

- [ ] Search application logs for connection strings, client secrets, tokens, and vault responses.
- [ ] Confirm error messages do not reveal secret values.
- [ ] Review telemetry pipelines for sensitive data redaction.
- [ ] Verify debug logging is safe to enable in lower environments.
- [ ] Ensure failed secret lookups are observable without exposing the secret itself.

8) Validate operational resilience

- [ ] Confirm the vault is treated as a critical runtime dependency.

- [ ] Document what happens if identity authentication fails.
- [ ] Document what happens if network access to the vault is interrupted.
- [ ] Determine whether cached values are allowed and for how long.
- [ ] Confirm startup, health checks, and readiness checks reflect real secret dependency status.

9) Review governance and auditing

- [ ] Confirm access to secrets is logged and auditable.
- [ ] Validate that rotation events can be traced.
- [ ] Review who can change secret values and under what approval process.
- [ ] Confirm stale or unused secrets are removed.
- [ ] Set a schedule to review secret inventory and access permissions.

10) Decide whether the pattern is a fit

- [ ] The API handles meaningful secrets such as API keys, database passwords, client secrets, or signing material.
- [ ] The application runs in an environment that can authenticate cleanly to the vault.
- [ ] Secret rotation without redeployment is important.
- [ ] Separation of duties between development and operations is required.
- [ ] The runtime dependency on the vault is acceptable for this service.

Quick production readiness test

Before approving rollout, confirm the following:

- [ ] The application starts successfully without secrets embedded in code or config files.

- [ ] The application can retrieve the secret from the vault using the approved identity.
- [ ] The service continues to behave correctly after a secret rotation.
- [ ] Logs, artifacts, and telemetry do not expose secret values.
- [ ] Access is least-privilege and documented.
- [ ] A rollback or recovery plan exists if the vault becomes unavailable.

Common mistakes to avoid

- [ ] Storing the secret in the vault and also copying it into appsettings files.
- [ ] Reusing one secret across every environment without a clear reason.
- [ ] Giving the application broader vault permissions than necessary.
- [ ] Assuming secret refresh works without testing it.
- [ ] Treating non-secret configuration as if it must live in the vault.
- [ ] Failing to test what happens when the vault cannot be reached.

Final sign-off

Use this section to record a release decision.

- Service name: _____
- Environment: _____
- Identity method: _____
- Secret(s) validated: _____
- Rotation tested on: _____
- Reviewer: _____
- Approval date: _____

Go-live decision

- [ ] Approved
- [ ] Approved with follow-up actions
- [ ] Not approved

Follow-up actions

- _____
- _____
- _____