ChatGPT

Vault Backend in Composite – Understanding the Error and Fixing It

What the Error Means

When you see an exception like `No thread-bound request found` in the Config Server logs (as shown in your stack trace), it indicates that **Spring Cloud Config Server's Vault integration is trying to access an HTTP request attribute when no HTTP request is present**. In your case, this happens during the Config Server's *health check* for the Vault backend. The Config Server is attempting to retrieve the Vault token from an HTTP request (via a `RequestContextHolder`), but since the health check runs in a background thread (not within an actual client HTTP request), there is no request context. This triggers the `IllegalStateException` you observed [1]. In short, the Vault environment repository is attempting to find a Vault token in the incoming request, and none is available during that internal operation.

This error is a known behavior of Spring Cloud Config when using a Vault backend. **By default, Config Server expects clients to supply the Vault token via an HTTP header (`X-Config-Token`)** on each request [2]. If no token is provided in the request (as is the case during a server-initiated health check or for clients that don't need Vault), the Vault repository component ends up with no token and fails. As noted in a Spring Cloud issue, if the Vault token isn't provided, the Vault backend health check will respond as *failed* [3] – which is exactly what you're seeing in the logs. The warning in your log (`ConfigServerHealthIndicator : Health check failed`) is indicating that the Vault portion of the composite config could not be accessed due to missing token context.

Cause: Missing Custom ConfigTokenProvider and Request Context

The root cause is that **you have not defined a custom `ConfigTokenProvider` bean**, so Spring Cloud Config Server falls back to its default behavior of using an **HTTP request-based token provider**. Internally, Spring Cloud Config Server's Vault support uses a `ConfigTokenProvider` to obtain the Vault token. In the absence of a custom implementation, it uses `HttpRequestConfigTokenProvider`, which tries to get the token from the `X-Config-Token` header of the current `HttpServletRequest` [4]. This works when a client call supplies that header, but it breaks outside a web request. During startup or periodic health checks (which run in a separate thread with no HTTP request), there is no request bound to the thread, hence the `RequestContextHolder.currentRequestAttributes()` call throws an exception [5].

In your configuration, you did set a static token (`token: root`) for the Vault backend and even set `request-token-header: false` (which is intended to tell the server not to expect the header). Intuitively, this should make the Config Server use the provided token for Vault calls. **However, the health check mechanism still attempted to use the default request-scoped token provider**, likely because no alternative was registered and the code wasn't aware of the static token in the context of that check. This is a quirk (or bug) in Spring Cloud Config's handling of Vault in a composite setup: the

VaultEnvironmentRepository tries the request-based token lookup even when a token is configured, at least in the health indicator flow. The Spring Cloud team has acknowledged that the Vault repository should *gracefully handle the absence of a token* (for example, skip Vault if no token is available, instead of throwing) [6], but in the versions around Spring Cloud 2022.0.x it still throws an error in this scenario.

In summary, having Vault in the composite without a custom token provider causes the Config Server to attempt token retrieval from the HTTP request context by default. When that context is missing (as during the composite health check), you get the `IllegalStateException` seen in your logs. The presence of `spring.cloud.config.server.vault.request-token-header=false` was supposed to avoid relying on the header, but the health check didn't honor that setting, resulting in the warning.

Version Compatibility (Spring Cloud Config 4.x with Spring Boot 3.x)

You also inquired about version compatibility. **Yes, Spring Cloud Config Server 4.x (Spring Cloud 2022.0 release train) is designed to work with Spring Boot 3.x.** In fact, Spring Cloud 2022.0.3 (which brings Config Server 4.0.3) specifically added compatibility with Spring Boot 3.1.x [7]. According to the official Spring Cloud compatibility matrix, the 2022.0.x “Kilburn” release train supports Spring Boot 3.0.x, and starting with 2022.0.3 it supports 3.1.x as well [8]. Your POM shows Spring Boot 3.1.11 and Spring Cloud 2022.0.3, which is a correct and supported combination. So there's no version mismatch here contributing to the issue – the error is not due to an incompatibility, but rather due to how Vault token is handled in this setup.

How to Resolve or Work Around the Issue

To eliminate the error (and properly supply the Vault token in a composite configuration), consider the following solutions:

- **Define a Custom `ConfigTokenProvider` Bean:** Since you confirmed you haven't provided one, adding a bean that implements `ConfigTokenProvider` will override the default `HttpRequestConfigTokenProvider`. For example, you can create a bean that always returns your Vault token (or fetches it from config):

```
import org.springframework.cloud.config.server.environment.ConfigTokenProvider;
import org.springframework.context.annotation.Bean;

@Bean
public ConfigTokenProvider vaultConfigTokenProvider() {
    // return a ConfigTokenProvider that supplies the Vault token (e.g., from
properties)
    return () -> "your-root-token-here";
}
```

Registering such a bean in the Config Server application context will ensure that Spring Cloud Config uses it to get the Vault token for all operations (including health checks), instead of looking at the HTTP request.

This is especially important in composite setups where not every request will carry a Vault token header. By supplying the token programmatically, you decouple it from the incoming request. (Make sure to **not** hard-code the token in real scenarios – you can fetch it from a secure source or environment variable).

- **Double-Check Vault Configuration:** Ensure that your composite Vault config includes all necessary properties. For a static token, you should have `spring.cloud.config.server.vault.authentication=TOKEN` (the default is TOKEN if a token is provided) and `spring.cloud.config.server.vault.token=<token value>` configured. In YAML, this was implicit in your composite section (`token: root` under the vault repo). Generally, this is sufficient. The `request-token-header: false` is meant to indicate that clients may not always send the header, which is fine. The key issue is the absence of a token provider for internal use, which the above bean addresses.

- **Upgrade or Patch (if available):** Check if there's an updated patch version of Spring Cloud Config in the 2022.0.x line (for example, 2022.0.7 with Config Server 4.0.7) where this behavior is improved. There have been discussions in the Spring Cloud community about making the Vault health check more lenient when no token is present [6]. Upgrading to the latest service release might include a fix where the health indicator no longer throws an exception if the Vault token is absent, but instead reports a healthier state or skips Vault. If upgrading isn't an option, the custom provider approach remains the direct fix.

- **Ignore/Handle the Health Indicator Warning:** It's worth noting that this error **does not shut down the application** and the Config Server can still serve properties from Vault when a proper request with a token (or the configured token) is present [9]. In other words, the warning is mainly indicating that the health check couldn't authenticate to Vault (because no token was provided in that context). If you cannot easily change the code or configuration, you might choose to ignore this warning or disable the Vault health check. Setting `fail-on-composite-error: false` (which you already did) is why the server continues to start – it prevents a single backend failure from stopping the whole composite. You could also consider customizing the health indicator to exclude Vault, though that's usually not necessary unless the noise is problematic.

In conclusion, you get this error because **Spring Cloud Config Server (4.x) with a Vault backend in a composite environment is attempting to retrieve the Vault token from the HTTP request context by default, and no request is available during certain operations (like health checks)**. The fix is to supply the token through a different mechanism (e.g., a ConfigTokenProvider bean or proper server config), so that the Vault environment repository doesn't depend on an incoming request. Once you do that, the health check should pass without errors, and your Vault-backed configuration should work seamlessly alongside the other composite sources.

Sources:

- Spring Cloud Config Server Vault backend documentation – explains default token via `X-Config-Token` header and alternative authentication configuration [2].
- Spring Cloud issue discussion – notes that a Vault health check fails if no token is provided, and suggests handling missing token more gracefully [3] [6].
- Stack Overflow question with identical error – confirms that the exception appears on startup with Vault and that it's related to token handling (no request bound) [1].

- Spring release notes – Spring Cloud 2022.0.3 is compatible with Spring Boot 3.1.x, confirming your version setup is correct [7] [8].

---

[1] [4] [5] [9] java - Spring Cloud Config + HashiCorp Vault Cannot connect - Stack Overflow
https://stackoverflow.com/questions/74982117/spring-cloud-config-hashicorp-vault-cannot-connect

[2] Vault Backend :: Spring Cloud Config
https://docs.spring.io/spring-cloud-config/reference/server/environment-repository/vault-backend.html

[3] [6] Support Vault health endpoint in health check · Issue #1536 · spring-cloud/spring-cloud-config · GitHub
https://github.com/spring-cloud/spring-cloud-vault/issues/378

[7] Spring Cloud 2022.0.3 (aka Kilburn) Is Available
https://spring.io/blog/2023/05/25/spring-cloud-2022-0-3-aka-kilburn-is-available/

[8] Spring Cloud
https://spring.io/projects/spring-cloud/