Configuration Frameworks/Solutions

Yasin Tüzen[1]

May 16, 2025

Contents

Introduction

Why Configuration Matters

Modern software runs in multiple environments:

- Local development, staging, production – all need different settings.
- API URLs, credentials, feature flags, and timeouts must be environment-specific.
- Configuration should be **externalized from the codebase**.

Bad configuration practices cause:

- Hard-to-debug bugs in production.
- Secrets accidentally pushed to Git.
- Fragile deployments and inconsistent environments.

What Is Configuration Management?

Definition: The process of handling dynamic parameters that influence application behavior **without modifying the source code**.

Types of configuration:

- **Static:** loaded at app startup (e.g. from ‘application.properties‘)
- **Dynamic:** changed at runtime (e.g. via Config-UI or REST API)
- **Secrets:** API keys, passwords – must be handled securely

Goal: Decouple environment-specific data from the core logic, while ensuring security and flexibility.

Presentation Goals

This talk will help you understand:

- Core concepts of configuration and environment-based separation
- Tools: Spring Cloud Config, HashiCorp Vault, Consul, etc.
- How to manage secrets and version configuration
- Demo: Our own Angular Config-UI in the StayInSync project
- Best practices for secure and scalable config in Quarkus + Angular

Core Concepts of Configuration Management

Local Configuration

Definition

- Configuration is stored directly in application files
- Examples: ‘application.properties‘, ‘config.yaml‘, ‘environment.ts‘

Benefits

- Easy to understand and implement
- No external dependencies or network calls
- Works well in development or single-instance setups

Limitations

- Requires redeploy to change config
- Risk of secrets being committed to version control
- No visibility across environments or services

Typical Use Case

- Local Angular dev environment using ‘environment.ts‘
- Monolith service with hardcoded file-based configuration

Centralized Configuration

Definition

- Configuration is managed in an external system or service
- Fetched by applications at runtime via API or SDK

Benefits

- Single source of truth for multiple applications
- Supports dynamic config updates without redeploy
- Better separation of concerns and access control

Limitations

- Requires network connectivity and service availability
- Additional setup and maintenance overhead
- Potential performance bottleneck or security target

Typical Use Case

- Cloud microservices using Spring Cloud Config or Vault
- Frontend apps using ConfigCat for feature toggles

Static Configuration

Definition

- Configuration values are loaded once at application startup
- Remain fixed for the entire runtime session

Advantages

- Simple and predictable behavior
- Easy to test and debug
- Supported natively in most frameworks (e.g. Quarkus)

Limitations

- Changing config requires restart or redeploy
- Not suitable for user-driven or time-sensitive updates
- Lacks flexibility in dynamic environments

Typical Use Cases

- Application ports, database credentials, log levels
- Quarkus config via ‘application.properties‘
- Angular build-time ‘environment.ts‘

Dynamic Configuration

Definition

- Configuration values can be updated during runtime
- Changes take effect immediately without restarting the app

Advantages

- Enables real-time updates and flexibility
- Useful for UI-driven parameters and feature toggles
- Can adapt behavior based on user input or external systems

Limitations

- Requires extra logic (reload mechanisms, state watchers)
- Risk of inconsistency or performance overhead
- Must be secured to prevent unauthorized changes

Typical Use Cases

- Sync rule changes in Stay in Sync UI
- ConfigCat toggles or UI preferences
- REST-based runtime config in microservices

Why Environment-specific Configuration?

Applications behave differently in each environment:

- **Development** (dev): verbose logging, local test data
- **Testing/QA**: mocked services, stable datasets
- **Production** (prod): optimized performance, real credentials

Goals of environment separation:

- Avoid accidental deployment of dev settings to production
- Enable testing in safe and reproducible environments
- Allow team members to work independently in dev mode

Typical examples:

- API endpoints
- Logging level (‘debug‘ vs ‘error‘)
- Feature toggles and security settings

Implementing Environment Separation

Angular (Frontend)

- Uses ‘environment.ts‘ and ‘environment.prod.ts‘ files
- Values are replaced at build time using the Angular CLI
- Examples: base URLs, feature flags, debug flags

Quarkus (Backend)

- Uses profiles like ‘application-dev.properties‘, ‘application-prod.properties‘
- Profile is selected via CLI or environment variable (‘quarkus.profile‘)
- Examples: DB connections, log format, polling interval

CI/CD Tools

- Inject environment variables during deployment
- GitHub Actions / GitLab CI support secrets per stage
- Enable automated testing per environment

Secrets in Configuration

What are secrets?

- Sensitive values such as:
  - API keys
  - Database passwords
  - OAuth tokens and credentials
- Unlike general config, secrets must be handled with special care
- They can grant full access to systems, databases, or data

Why are secrets a risk?

- Easily leaked via:
  - Git commits (‘.env‘, config files)
  - Frontend bundles (Angular)
  - Misconfigured server environments
- Can lead to data breaches or full system compromise

Secure Handling of Secrets

Best practices for managing secrets

- **Do not hardcode secrets** in files or source code
- **Use environment variables** in deployment environments
- **Use secret managers** such as:
  - HashiCorp Vault
  - AWS Secrets Manager / Azure Key Vault
  - Kubernetes Secrets
- **Use temporary tokens** (e.g. OAuth2) instead of static keys

Project example (Stay in Sync)

- Frontend (Angular): never stores or handles secrets
- API Key field is passed securely to backend only
- Future use of Vault possible to store EDC-related credentials

Evaluated Frameworks

Spring Cloud Config

Overview

- Centralized configuration service for distributed applications
- Stores versioned config in Git repositories
- Exposes config via REST API to connected services

Capabilities

- Environment-specific profiles (dev/test/prod)
- Git-based versioning and rollback
- Live refresh for Spring Boot apps
- Ideal for Spring microservices using shared config

Limitations

- Tightly coupled with Spring ecosystem
- Setup requires both config server and Git repo
- No Angular or Quarkus integration

HashiCorp Vault

Overview

- Secure storage and access for secrets like credentials and API tokens
- Central tool in zero-trust and cloud-native environments

Capabilities

- Dynamic secrets (auto-expiring credentials)
- Role-based access policies with fine-grained control
- Audit logs for full traceability
- Quarkus extension for integration
- Ideal for securing EDC tokens, CI/CD secrets, and backend services

Limitations

- Requires setup of Vault server and policies
- Backend-only – no Angular/front-end usage
- High complexity for small teams or projects

SmallRye Config (Quarkus)

Overview

- Built-in configuration framework in Quarkus
- Supports ‘.properties‘, ‘.yaml‘, and environment variables
- Designed for simple, efficient backend configuration

Capabilities

- Profile-based configs (‘application-dev.properties‘, etc.)
- Type-safe injection via ‘@ConfigProperty‘
- Startup configuration for ports, retries, log levels
- Load priority: env vars $>$ config file $>$ default

Limitations

- Static – values can’t change at runtime
- No support for dynamic config via UI or REST

ConfigCat

Overview

- Cloud-based configuration and feature flag platform
- Designed for real-time frontend configuration updates
- Works well with Angular, mobile, and web apps

Capabilities

- Runtime config updates without redeployments
- Feature toggles (e.g. enable/disable features per user or region)
- Angular SDK and REST API integration
- No server-side setup required

Limitations

- SaaS-only – external dependency and vendor lock-in
- Limited support for backend config (Quarkus, CLI, etc.)
- Free plan restricts environments and usage

Not Suitable for Angular + Quarkus

Several frameworks were evaluated but excluded for technical reasons:

- **Spring Cloud Config:** tightly coupled to Spring Boot → not usable with Quarkus
- **Consul:** designed for service discovery, lacks simple integration with Angular
- **Configu:** JavaScript-first tool, limited backend support
- **Dynaconf:** Python-based, not compatible with JVM projects
- **Rudder / opsi:** focused on system-level config management, not application settings

Focus was placed on tools that are lightweight, cross-platform, and easily integrated with Angular and Quarkus.

HashiCorp Vault Demo

Vault Demo: Goal and Setup

Goal: Securely manage secrets — e.g., EDC tokens should not be hardcoded in the source code.

Setup:

- Start Vault in `dev` mode:

```
vault server -dev
```

- Store a secret:

```
vault kv put secret/edc token=abcd1234
```

Vault Test via curl

Retrieve the secret using curl:

```
export VAULT_ADDR=http://127.0.0.1:8200

curl -H "X-Vault-Token: <root-token>" \
  http://127.0.0.1:8200/v1/secret/data/edc
```

Expected response:

```
{
  "data": {
    "data": {
      "token": "abcd1234"
    }
  }
}
```

Note: Verified live before integrating into Quarkus.

Vault Integration in Quarkus (Pseudocode)

What does the endpoint do?

```
// GET /vault/edc-token

1. Send HTTP GET to:
   http://127.0.0.1:8200/v1/secret/data/edc
   with header "X-Vault-Token"

2. Parse the JSON response

3. Extract: "token": "abcd1234"

4. Return:
   HTTP 200 OK + Body: abcd1234
```

Implemented as a Quarkus REST resource using a basic HTTP client.

End-to-End Demo: Vault Quarkus curl/Postman

1. **Start Vault and store secret:**

```
vault server -dev
vault kv put secret/edc token=abcd1234
```

2. **Quarkus endpoint:**
   - `GET /vault/edc-token`
   - Fetches secret from Vault and returns it as plain text
3. **Call via curl:**

```
curl http://localhost:8080/vault/edc-token
    Response: abcd1234
```

4. **Alternative:** Use Postman to call the same endpoint

Result: Secure, centralized secret access without hardcoding.

ConfigCat-Demo

ConfigCat Demo: Goal Concept

Goal: Enable or disable features without code changes — configured live via the ConfigCat dashboard.

Objective:

- The backend checks the flag `syncEnabled` in ConfigCat
- The endpoint decides whether to start sync or return an error
- Changes are reflected immediately without redeployment

Technology: Uses the `ConfigCatClient` library in Quarkus

Backend Logic for Feature Toggle

What does the endpoint do?

```
// GET /sync/start

1. Check ConfigCat: is "syncEnabled" active?

2. If NO      return:
   HTTP 403 Forbidden + "Sync is disabled"

3. If YES      return:
   HTTP 200 OK + "Sync started successfully!"
```

Advantage: Feature status can be changed at any time — without restarts or config files.

Live Demo: Toggle Feature in ConfigCat

Steps:

1. Disable the `syncEnabled` flag in the ConfigCat dashboard
2. Call: `GET /sync/start` in browser or Postman
3. → Response: `403 Forbidden`, sync is blocked
4. Re-enable the flag in ConfigCat
5. → Call again: `/sync/start`
6. → Response: `200 OK` + "Sync started successfully!"

Note: Depending on caching, a short delay might occur — use LazyLoad to avoid it.

Configuration Needs in Stay in Sync

Configuration Needs in "Stay in Sync"

Core requirements for configuration management:

- **Runtime control:** Enable or disable sync features at any time
- **Environment-specific settings:** Different values for dev, test, prod
- **Secure token management:** API credentials (e.g., EDC access tokens)
- **UI-based rule setup:** Non-developers need a simple configuration view
- **Good separation of concerns:** Config should be external to code

Goal: Select tools that match these needs while being lightweight and easy to maintain.

Tool Comparison: ConfigCat, Vault, SmallRye Config

Use Case	ConfigCat	Vault	SmallRye Config
Feature toggling at runtime	✓	✗	✗
Secret/token storage	✗	✓	✗
Profile-specific settings	✗	✗	✓
External API integration	✓	✓	✗
Supports UI-based config	**Dashboard**	✗	✗
Live value updates	**(with LazyLoad)**	✗	✗

Conclusion: Each tool is suited for a specific class of configuration needs.

Final Tool Selection for Our Project

Recommended stack:

- **ConfigCat** for:
  - UI feature toggles (sync button control)
  - Quick enable/disable logic during runtime
- **HashiCorp Vault** for:
  - Secure storage of tokens and credentials
  - Centralized and protected backend access
- **SmallRye Config (Quarkus)** for:
  - Static application settings
  - Environment-specific values (e.g., dev/prod config)

Final Recommendation: ConfigCat or not?

Vault is essential for secure token management.

SmallRye Config covers most of our backend needs:

- Environment-specific configuration (dev/prod)
- Startup-only values like sync intervals
- Fully integrated into Quarkus

ConfigCat is not required:

- Suitable for live toggling of UI features
- Useful mainly for demos or A/B testing
- Adds external dependency and complexity

Conclusion: SmallRye Config is sufficient. ConfigCat can be skipped unless runtime UI control becomes essential.

Best Practices for Secure and Scalable Configuration

Do:

- Keep secrets out of code and Git
- Use external tools (Vault, ConfigCat) for sensitive values
- Separate configs per environment (dev / prod)
- Use defaults + overrides (env vars > files > code)
- Document config changes (versioning / Git)

Don't:

- Hardcode credentials or feature flags
- Use the same config across environments
- Trust the frontend with security logic

Thank you for your attention!

Questions are welcome.

Stay in Sync – Configuration Frameworks
Bosch StuPro SS 2025

References – Concepts and Best Practices

- Heroku. *The Twelve-Factor App – Configuration Principle*. `https://12factor.net/config`
- Microsoft Docs. *Best practices for application configuration*. `https://learn.microsoft.com/en-us/dotnet/core/extensions/configuration`
- Martin Fowler. *Configuration Management Article*. `https://martinfowler.com/articles/configuration.html`
- Bell, James. *Modern DevOps Practices*. O'Reilly Media, 2021.

References – Tools and Official Docs

- ConfigCat Team. *ConfigCat Feature Flag Service*. `https://configcat.com`
- HashiCorp. *Vault – Secrets Management*. `https://www.vaultproject.io/`
- Red Hat. *Quarkus Configuration Guide*. `https://quarkus.io/guides/config-reference`
- SmallRye Project. *SmallRye Config Specification*. `https://smallrye.io/smallrye-config/`
- Spring Team. *Spring Cloud Config Docs*. `https://spring.io/projects/spring-cloud-config`