Передаем секреты в **Spring Boot** бины безопасно

Илья Николаев

МегаТех

МегаФон

SPRING АЙО

Чем же плохи ENV-ы?

ИИ:

1. Отсутствует шифрование
2. Доступны любому процессу
3. Доступны любому пользователю
4. Неочевидные места утечек, например через инструменты логирования

AWS

2024

> 90 000
переменных

~ 18 000
кредов к БД

~ 8 000
кредов к AWS

~ 1 500
кредов соц. сетям

Детали атаки

Docker-container

за $ 12 000

Детали атаки


НЕТ СЕКРЕТАМ
В ПЕРЕМЕННЫХ
ОКРУЖЕНИЯ
В ПРОДЕ!

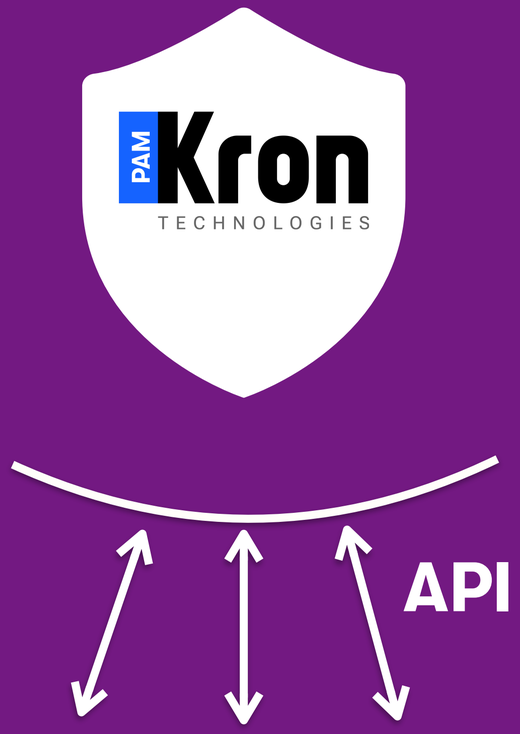
Часть 1

Корпоративное хранилище секретов

application.properties

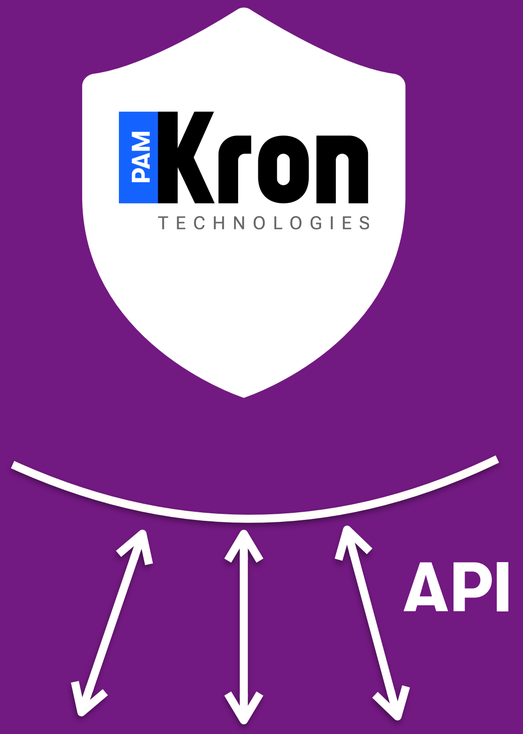
```
spring.datasource.url=jdbc:postgresql://\
  ${PG_HOST:localhost}:\
  ${PG_PORT:5432}/${PG_DB:demo}
spring.datasource.username=${PG_USER:demo}
spring.datasource.password=${PG_PASS:changeme}
```

Что такое PAM?




```
some_project
└── secrets
    ├── credentials
    │   ├── postgres
    │   ├── redis
    │   └── rabbit
    ├── tokens
    │   └── some_external_api_token
    ├── ssh_keys
    │   └── on_prem_customer_1
    └── ssl
```

Что такое PAM?



За счет чего безопасно?

1. Только из внутренней сети
2. Белые списки клиентских адресов
3. Свой токен на каждую группу секретов
4. Токен связан с приложением
5. Токен выдается под подпись
6. Логирование всех обращений

Шаг 1

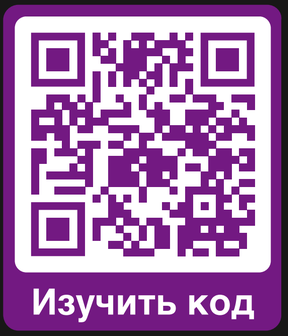
Пишем MVP

Шаг 1

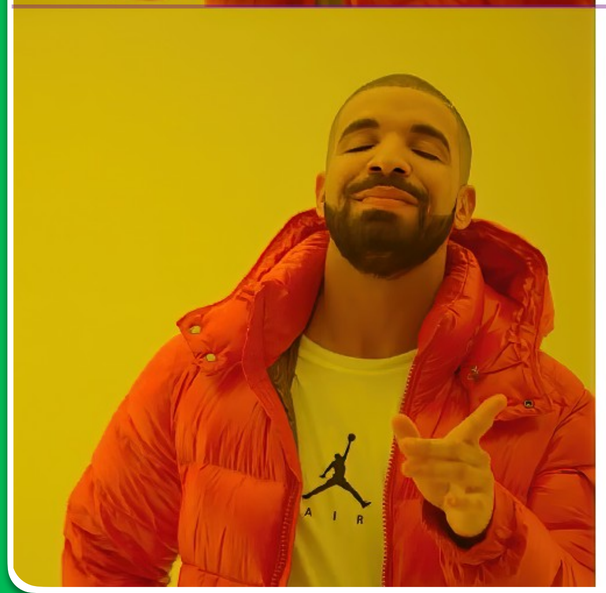
MVP

PamConfig.java

```
31. @Bean
32. public DataSource dataSource(
33.         …
34.     ) throws Exception {
35.     StaticUserCredentials credentials
36.             = readCredentials(pamPath, pamSecret, pamToken);
37.     return DataSourceBuilder.create().url(jdbcUrl)
38.             .username(credentials.username())
39.             .password(credentials.password()).build();
40. }
```



Раскопировать кусок кода между 10 микросервисами за час



Потратить целый день чтобы написать Spring Boot Stareter

Шаг 2

Spring Boot Starter

application-pam.properties

```
pam.db.secret.path=/path/to/secret/changeme
pam.db.secret.name=demo
pam.krontech.token=some_token
pam.krontech.client-id=${spring.application.name}
```

Шаг 2

Spring boot starter

PamConfig.java

```
21. @EnableKrontech
22. @Profile("pam")
23. @Configuration
24. public class PamConfig {
25.     @Bean
26.     public DataSource dataSource(…
27.                          KronTechClient kronTechClient) {
            …
32.     }
33. }
```

Изучить стартер

Изучить код

Шаг 2

Spring boot starter

PamConfig.java

```
31. @Bean
32. public DataSource dataSource(
33.         …
34.     ) throws Exception {
35.     StaticUserCredentials credentials
36.             = kronTechClient.read(pamPath, pamSecret);
37.     return DataSourceBuilder.create().url(jdbcUrl)
38.             .username(credentials.username())
39.             .password(credentials.password()).build();
40. }
```

Изучить стартер

Изучить код

Шаг 2

Spring boot starter

- Больше бинов - больше класс конфигурации
- Нельзя работать со стартерами через пропсы
- Разная инициализация бинов в DEV и PROD

Шаг 3

ApplicationContextInitializer

Вызов SpringApplication.run()

Подготовка Environment

Создание ApplicationContext

Подготовка BeanFactory

Вызов BeanFactoryPostProcessors

Регистрация BeanPostProcessors

Инициализация MessageSource

Инициализация EventMulticaster

Создание Singleton бинов

Вызов Initialisation Callbacks

Публикация ContextRefreshedEvent

Запуск раннеров или веб-сервера

Приложение готово к работе

application.properties

```
spring.datasource.url=jdbc:postgresql://\
  ${PG_HOST:localhost}:\
  ${PG_PORT:5432}/${PG_DB:demo}
spring.datasource.username=${PG_USER:demo}
spring.datasource.password=${PG_PASS:changeme}
```

application-pam.properties

```
pam.db.secret.path=/path/to/secret/changeme
pam.db.secret.name=demo
pam.krontech.token=some_token
pam.krontech.client-id=${spring.application.name}
```

```
pam.db.secret.path
pam.db.secret.name
```

```
spring.datasource.username
spring.datasource.password
```

Шаг 3

ApplicationContextInitializer

application-pam.yaml

```yaml
pam:
  overrides:
    db:
      keys:
        user: spring.datasource.username
        pass: spring.datasource.password
      secret:
        path: /path/to/secret
        name: db_credentials
```

Шаг 3

ApplicationContextInitializer

```
22. public record PamProps(Map<String, Override> overrides, …) {
23.     public record Override(Keys keys, Secret secret) {
24.     }
25.
26.     public record Secret(String name, String path) {
27.     }
28.
29.     public record Keys(String user, String pass) {
30.     }
31. }
```

Шаг 3

ApplicationContextInitializer

PamContextInitializer.java

```
21. public class PamContextInitializer implements
22. ApplicationContextInitializer<ConfigurableApplicationContext>
23. {
24.     @Override
25.     public void initialize(…) {
26.       // TODO: implement
27.     }
28. }
```

Шаг 3

ApplicationContextInitializer

```
25. void initialize(ConfigurableApplicationContext context) {
26.     ConfigurableEnvironment env = context.getEnvironment();
27.
28.     if (!env.matchesProfiles("pam")) return;
29.
30.     PamProps pamProps = Binder.get(env)
31.         .bind("pam", Bindable.of(PamProps.class))
32.         .orElseThrow(() -> new ApplicationContextException());
33.     …
45. }
```

Шаг 3

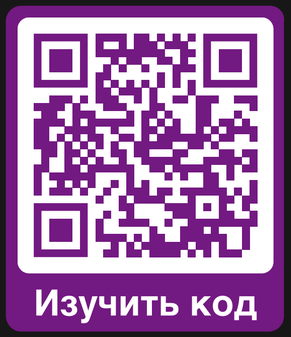
ApplicationContextInitializer

```
25. void initialize(ConfigurableApplicationContext context) {
26.     …
27.     pamProps.overrides().forEach((item, override) → {
28.         Secret secret = override.secret();
29.         Keys keys = override.keys();
30.         StaticUserCredentials credentials =
31.             pamClient.read(secret.path(), secret.name());
32.         env.getPropertySources().addFirst(
33.             overrideProps(item, keys, credentials));
34.     });
45. }
```

Шаг 3

ApplicationContextInitializer

```
49. MapPropertySource overrideProps(
50.         String name,
51.         Keys keys,
52.         StaticUserCredentials credentials
53. ) {
54.     Map<String, Object> props = HashMap.newHashMap(2);
55.     props.put(keys.user(), credentials.username());
56.     props.put(keys.pass(), credentials.password());
57.     return new MapPropertySource(name + "Props", props);
58. }
```



Шаг 3

ApplicationContextInitializer

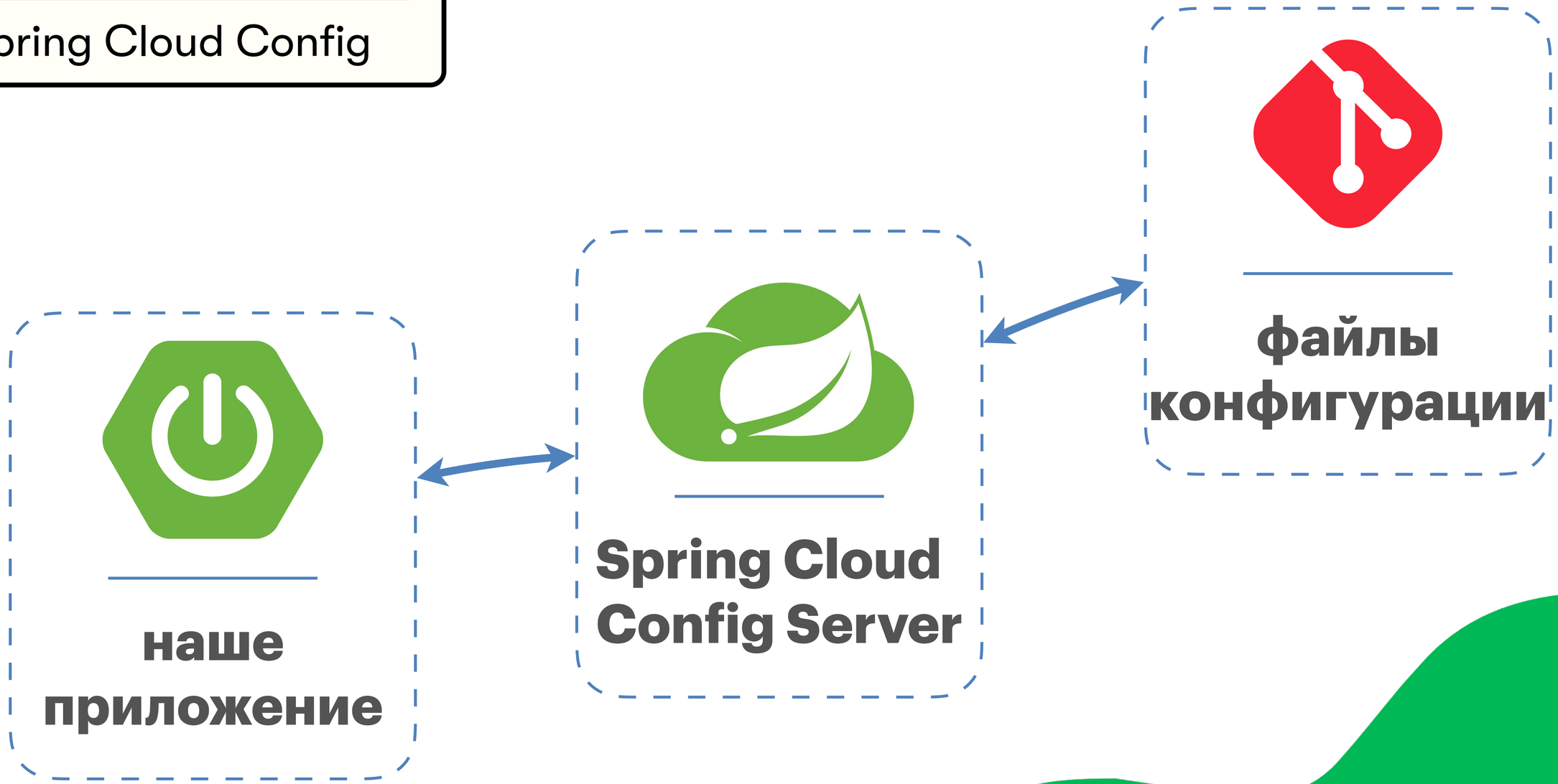
- Фиксированный код подмены пропсов
- Можем конфигурировать стартеры через пропсы
- Единая инициализация бинов в DEV и PROD

Часть 2

Скрываем конфиги для on-premise решений

Претендент № 1

Spring Cloud Config



Претендент № 2

HashiCorp Vault

1. Open Source
2. Хранит разные типы данных
3. Транзитный сервис шифрования и подписания данных
4. Есть динамические секреты
5. CLI и WEB для настройки
6. API для интеграции

Кворум

Защищаем мастер ключ

Как общаться?

Spring Cloud Vault

наше приложение

+

Spring Cloud Vault Config

Как общаться?

Spring Cloud Vault

```
51. <dependency>
52.  <groupId>org.springframework.cloud</groupId>
53.  <artifactId>spring-cloud-starter</artifactId>
54.  <version>${spring-cloud.version}</version>
55. </dependency>
56. <dependency>
57.  <groupId>org.springframework.cloud</groupId>
58.  <artifactId>spring-cloud-starter-vault-config</artifactId>
59.  <version>${spring-cloud.version}</version>
60. </dependency>
```

Как общаться?

Spring Cloud Vault

```
51. <dependency>
52.  <groupId>org.springframework.cloud</groupId>
53.  <artifactId>spring-cloud-vualt-config-database</artifactId>
54.  <version>${spring-cloud.version}</version>
55. </dependency>
56. <dependency>
57.  <groupId>org.springframework.cloud</groupId>
58.  <artifactId>spring-data-keyvalue</artifactId>
59.  <version>${spring-cloud.version}</version>
60. </dependency>
```

Как общаться?

Spring Cloud Vault

```
15. spring:
16.   cloud.vault:
17.     uri: ${VAULT_HOST:http://localhost:8200}
18.     authentication: TOKEN
19.     token: ${vault.app-token}
20.   config.import:
21.       - vault://
22.       - vault://secret/${spring.application.name}
```

Как общаться?

Spring Cloud Vault

```
13. spring:
14.   cloud.vault:
15.     uri: ${VAULT_HOST:http://localhost:8200}
16.     authentication: TOKEN
17.     token: ${vault.app-token}
18.     database:
19.       enabled: true
20.       role: ${spring.application.name}
21.       backend: database
22.   config.import:
23.       - vault://
24.       - vault://secret/${spring.application.name}
```

Ротация кредов

Spring Cloud Vault

```
30. VaultConfig(SecretLeaseContainer leaseContainer,
31.                 ContextRefresher contextRefresher) {
32.     String dbCredsPath = "jpoint_demo/creds/database";
33.     leaseContainer.addLeaseListener(evt → {
34.         if (dbCredsPath.equals(evt.getSource().getPath())) {
35.             if (evt instanceof SecretLeaseExpiredEvent lee) {
36.                 configDataContextRefresher.refresh();
37.             }
38.         }
39.     });
40. }
```

Ротация кредов

Spring Cloud Vault

```
@Bean
@RefreshScope
DataSource dataSource(DataSourceProperties properties) {
    return DataSourceBuilder
            .create()
            .url(properties.getUrl())
            .username(properties.getUsername())
            .password(properties.getPassword())
            .build();
}
```

Ротация кредов

Spring Cloud Vault

Изучить код


```
INFO: Starting JPointDemoApplication v0.0.1-vault-sandbox using Java 25 with PID 1 (demo.jar …
INFO: The following 1 profile is active: "vault"
INFO: Bootstrapping Spring Data Vault repositories in DEFAULT mode.
INFO: Finished Spring Data repository scanning in 51 ms. Found 1 Vault repository interface.
INFO: Root WebApplicationContext: initialization completed in 2562 ms
INFO: Creating DataSource with user: v-token-jpoint-d-78a0x5ZFPZ8HaE1RwOEA-1775552356
INFO: Secret loaded: this!is*my#secret@from~vault
INFO: Started JpointDemoApplication in 6.876 seconds (process running for 8.035)
INFO: HikariPool-1 - Starting...
INFO: HikariPool-1 - Added connection org.postgresql.jdbc.PgConnection@2d4bcb6f
INFO: HikariPool-1 - Start completed.
INFO: Lease expired! Event: SecretLeaseExpiredEvent. Context refreshed!
INFO: HikariPool-1 - Shutdown initiated...
INFO: HikariPool-1 - Shutdown completed.
INFO: Creating DataSource with user: v-token-jpoint-d-pjm1sCMzNK1rz9VBShDx-1775552396
INFO: HikariPool-2 - Starting...
INFO: HikariPool-2 - Added connection org.postgresql.jdbc.PgConnection@8b9eb7cd
INFO: HikariPool-2 - Start completed.
```

За счёт чего безопасность API?

А минусы будут?

- Open Source
- Сложность настройки
- Не всегда применимы все механизмы защиты
- Нужен кластер
- Автоматическая ротация кредов имеет свои нюансы

Итоги

Нужны секреты?

Сходи за ними сам!

наше приложение

СЕКРЕТЫ

Вызов SpringApplication.run()

Подготовка Environment

Создание ApplicationContext

Подготовка BeanFactory

Вызов BeanFactoryPostProcessors

Регистрация BeanPostProcessors

Инициализация MessageSource

Инициализация EventMulticaster

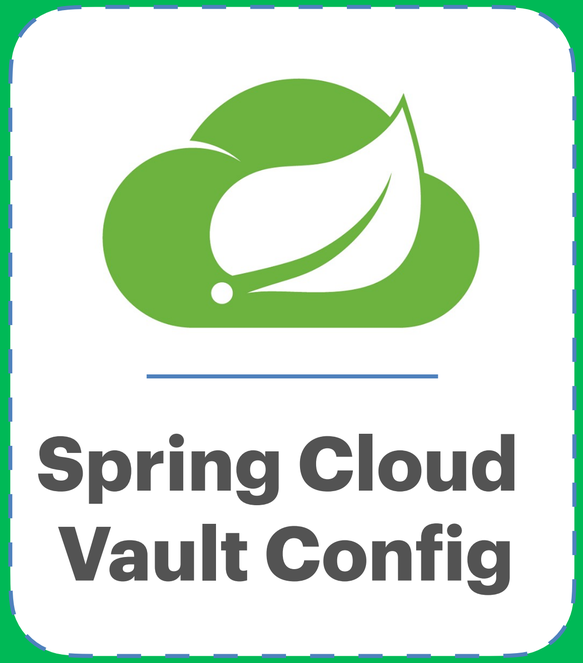
Создание Singleton бинов

Вызов Initialisation Callbacks

Публикация ContextRefreshedEvent

Запуск раннеров или веб-сервера

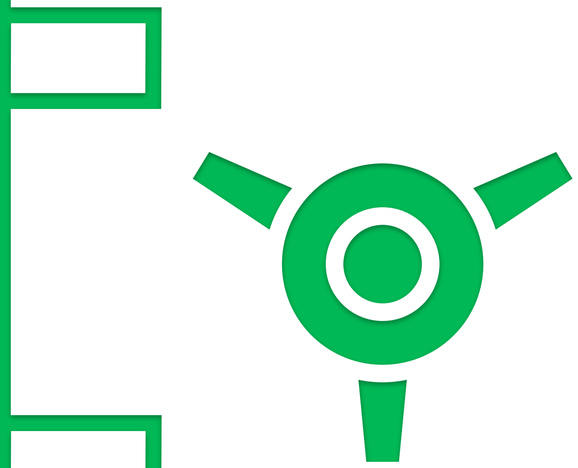
Приложение готово к работе


Spring Cloud
Vault Config

+


HashiCorp
Vault

SecretsProvider<T>



+

```
interface
ApplicationContextInitializer
```

```
@RefreshScope
```

Q&A

Примеры кода

Spring Boot Starter

Spring Cloud Vault

HashiCorp Vault

https://gitverse.ru/realimp/JPoint_demo-2026
https://gitverse.ru/realimp/JPoint_demo-2026-spring-starter
https://docs.spring.io/spring-cloud-vault/reference/index.html
https://developer.hashicorp.com/vault/docs