ПРАВИЛЬНЫЙ DEVOPS ДЛЯ SPRING BOOT И JAVA

>_ man RustamKuramshin

- Занимаюсь backend-разработкой на Java, Kotlin и Spring
- Люблю DevOps и задачи инфраструктуры
- Увлекаюсь Linux, Docker, Kubernetes и сетями
- Развиваю свой кластер на Raspberry Pi

@zen-code

@KuramshinRustam

О чём пойдёт речь

- Эффективный Dockerfile, использование Maven/Gradle плагинов и Cloud Native Buildpacks.
- Управление Spring Application Properties с помощью ConfigMap или Spring Cloud Config.
- Запуск миграций Liquibase в CI-пайплайне.
- Сделаем выводы

Репозиторий с кодом из презентации

https://github.com/RustamKuramshin/right-devops

“Уставшие DevOps’ы деплоят Java-сервисы…”

ЭФФЕКТИВНЫЙ DOCKERFILE

Распаковка JAR, переиспользование слоёв, Spring’овые Maven/Gradle плагины и Cloud Native Buildpacks

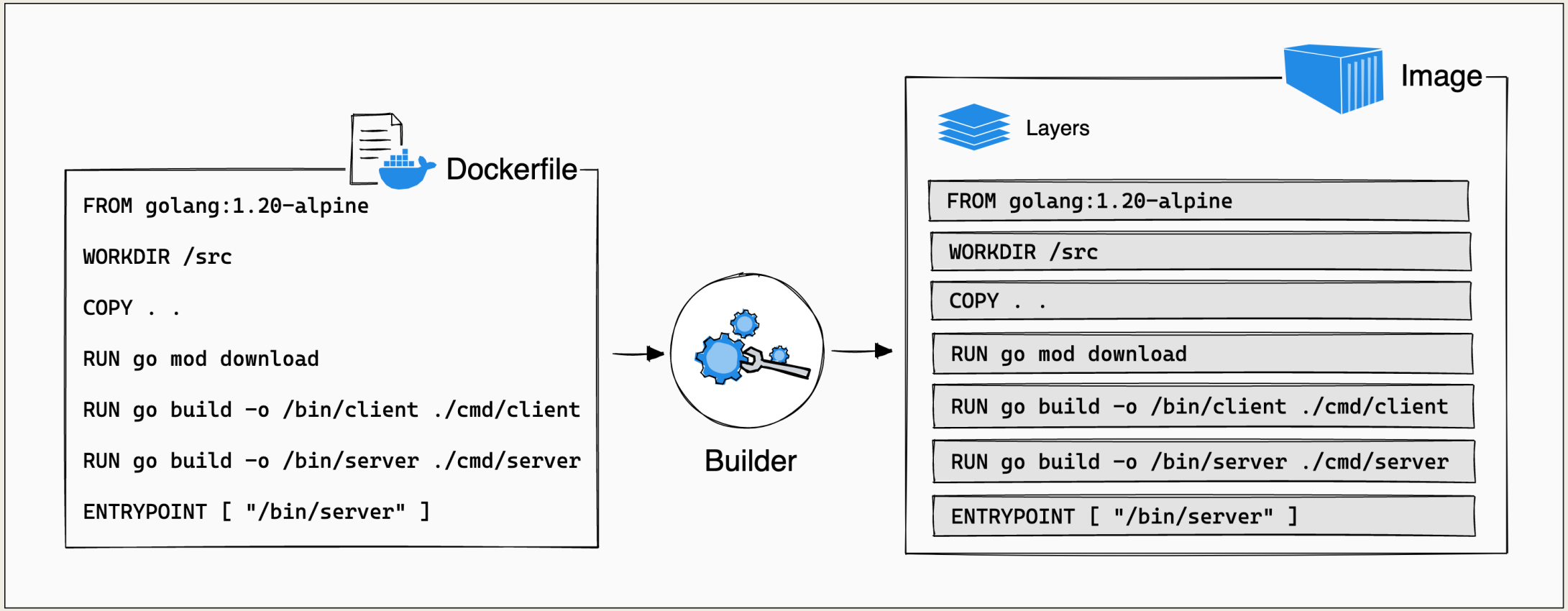
Обычный Dockerfile для Spring Boot

```dockerfile
FROM openjdk:17.0.2-slim

WORKDIR /app

COPY ./build/libs/right-devops-1.0.0.jar /app/right-devops.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "right-devops.jar"]
```

Обычный Dockerfile для Spring Boot

```dockerfile
FROM openjdk:17.0.2-slim

WORKDIR /app

COPY ./build/libs/right-devops-1.0.0.jar /app/right-devops.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "right-devops.jar"]
```

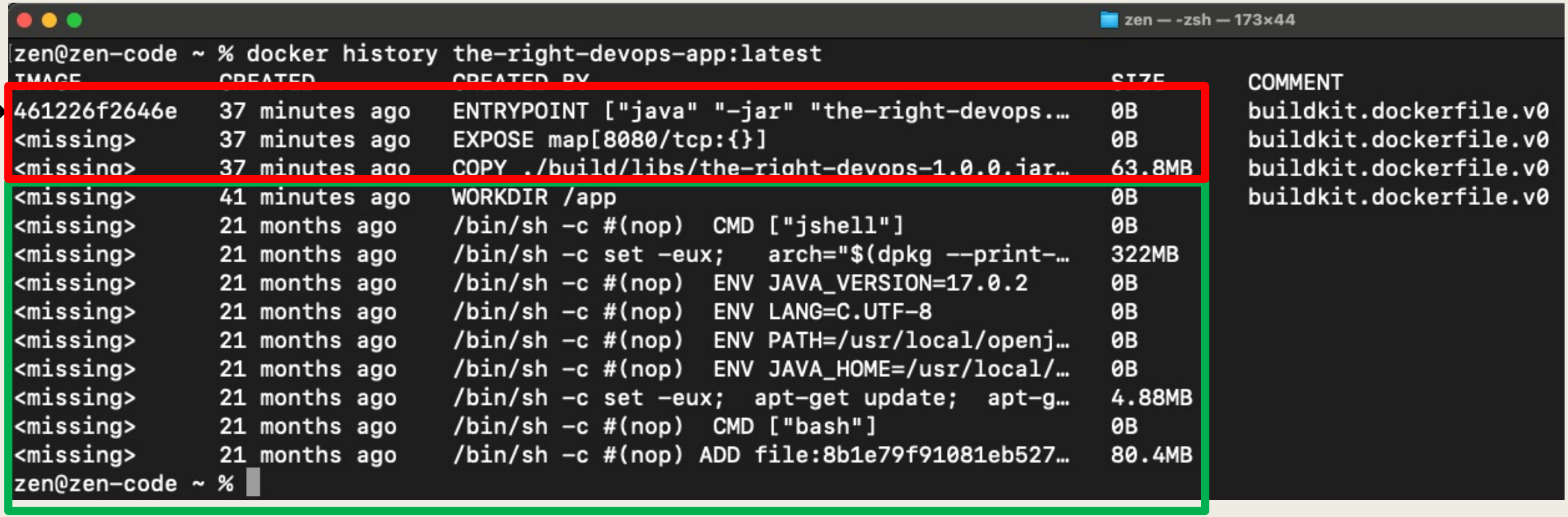
Что делает «docker build»



Правило кэширования слоёв docker образа

- При запуске сборке, билдер пытается повторно использовать слои из более ранних сборок.
- Если слой изображения не изменился, то билдер подхватывает его из кэша сборки.
- Если слой изменился с момента последней сборки, этот слой и все последующие слои необходимо перестроить.

Слои получившегося образа и их кэширование



НЕ будет кэшировано ~ 63.8 MB (на каждую сборку, push и pull)

Будет кэшировано ~ 407.28 MB

В чём здесь проблема? Оверхэды по времени, месту на диске и сетевому трафику.

- Скорость сборки: кэширование уменьшает время затраченное на docker build в CI-пайплайне.
- Отправка в Docker Registry: docker push передает образ в Registry по слоям.
- Кэшированные слои повторно не передаются.

В чём здесь проблема? Оверхэды по времени, месту на диске и сетевому трафику.

- Дисковое пространство под Docker Registry: кэширование слоёв экономит место на накопителях.
- Скорость получения образа из Docker Registry: при запуске сервиса в kubernetes происходит pull образа из Registry.
- Кэширование ускоряет pull.

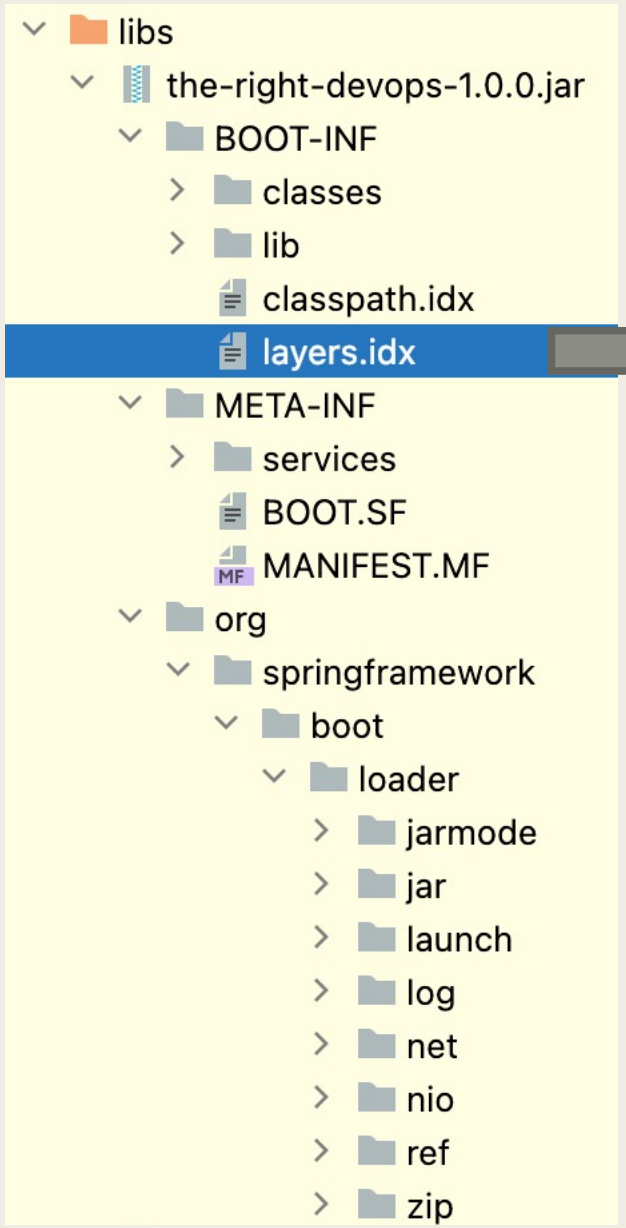
Как Spring решает проблему с кэшированием слоёв.

- В jar-архив добавляется индексный файл layers.idx
- layers.idx содержит сведения о путях в jar-архиве и слоях, на которые нужно разделить классы Spring приложения.

Как Spring решает проблему с кэшированием слоёв.

- Уровни определяются в зависимости от вероятности изменения между сборками приложения.
- Режим запуска Spring Boot приложения ”layertools” и его команда «extract» позволяют разобрать jar-архив в соответствии с layers.idx.

layers.idx



layers.idx

```
1  - "dependencies":
2    - "BOOT-INF/lib/"
3  - "spring-boot-loader":
4    - "org/"
5  - "snapshot-dependencies":
6  - "application":
7    - "BOOT-INF/classes/"
8    - "BOOT-INF/classpath.idx"
9    - "BOOT-INF/layers.idx"
10   - "META-INF/"
```

Как это работает

```dockerfile
FROM gradle:7.4.0-jdk17 as build
WORKDIR /app
COPY --chown=gradle:gradle . /app
RUN gradle clean build --no-daemon

FROM openjdk:17.0.2-slim as builder
WORKDIR /app
COPY --from=build /app/build/libs/right-devops-1.0.0.jar /app/right-devops.jar
RUN java -Djarmode=layertools -jar right-devops.jar extract

FROM openjdk:17.0.2-slim
WORKDIR /app
COPY --from=builder app/dependencies/ ./
COPY --from=builder app/spring-boot-loader/ ./
COPY --from=builder app/snapshot-dependencies/ ./
COPY --from=builder app/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.launch.JarLauncher"]
```

Как это работает

```dockerfile
FROM gradle:7.4.0-jdk17 as build
WORKDIR /app
COPY --chown=gradle:gradle . /app
RUN gradle clean build --no-daemon

FROM openjdk:17.0.2-slim as builder
WORKDIR /app
COPY --from=build /app/build/libs/right-devops-1.0.0.jar /app/right-devops.jar
RUN java -Djarmode=layertools -jar right-devops.jar extract

FROM openjdk:17.0.2-slim
WORKDIR /app
COPY --from=builder app/dependencies/ ./
COPY --from=builder app/spring-boot-loader/ ./
COPY --from=builder app/snapshot-dependencies/ ./
COPY --from=builder app/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.launch.JarLauncher"]
```

Как это работает

```dockerfile
FROM gradle:7.4.0-jdk17 as build
WORKDIR /app
COPY --chown=gradle:gradle . /app
RUN gradle clean build --no-daemon

FROM openjdk:17.0.2-slim as builder
WORKDIR /app
COPY --from=build /app/build/libs/right-devops-1.0.0.jar /app/right-devops.jar
RUN java -Djarmode=layertools -jar right-devops.jar extract

FROM openjdk:17.0.2-slim
WORKDIR /app
COPY --from=builder app/dependencies/ ./
COPY --from=builder app/spring-boot-loader/ ./
COPY --from=builder app/snapshot-dependencies/ ./
COPY --from=builder app/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.launch.JarLauncher"]
```

Что там со слоями образа

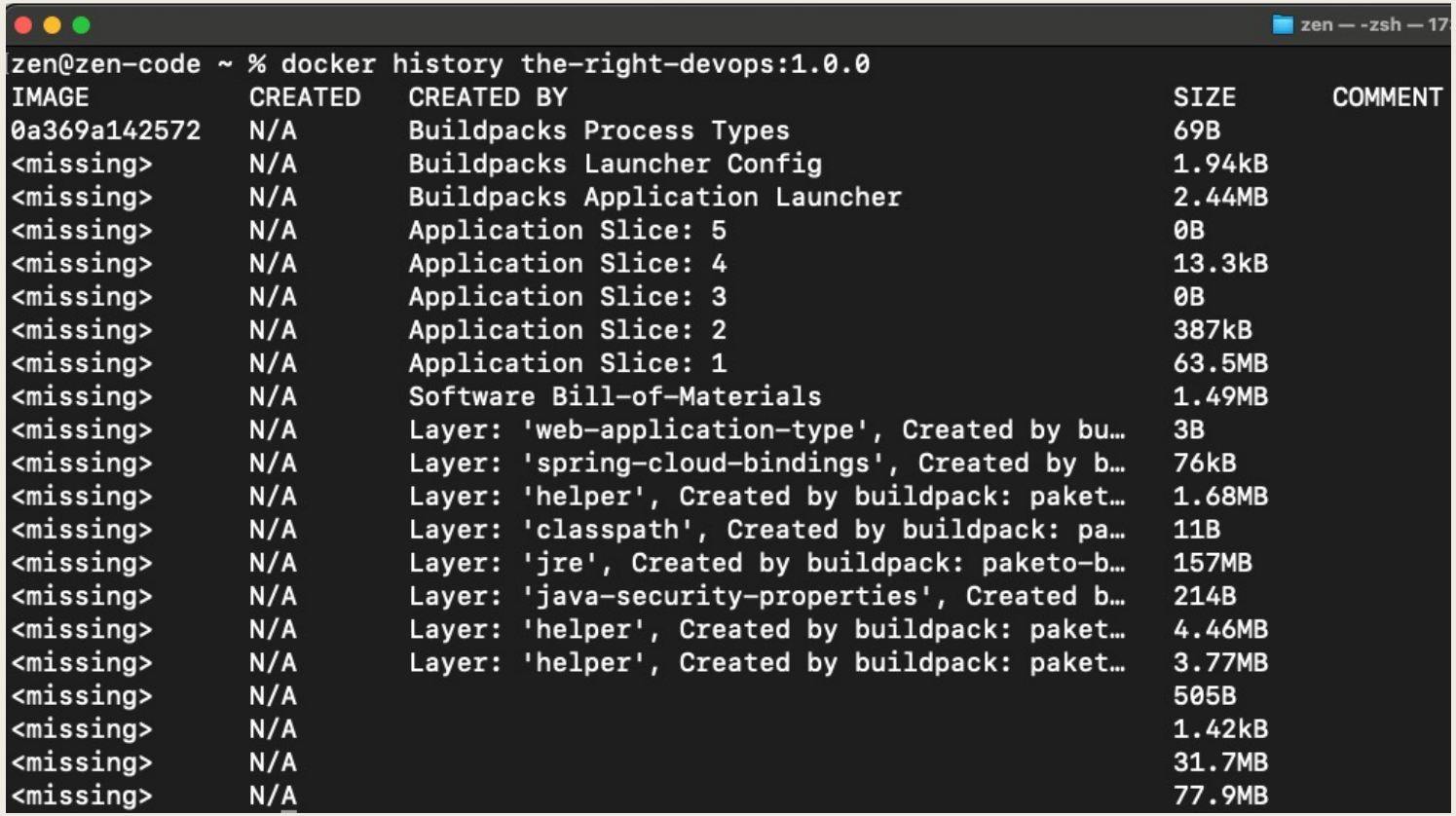
```
zen@zen-code ~ % docker history the-right-devops-app:latest
IMAGE          CREATED         CREATED BY                                      SIZE      COMMENT
d8c76685a8cc   6 minutes ago   ENTRYPOINT ["java" "org.springframework.boot…   0B        buildkit.dockerfile.v0
<missing>      6 minutes ago   COPY app/application/ ./ # buildkit             13.3kB    buildkit.dockerfile.v0
<missing>      6 minutes ago   COPY app/snapshot-dependencies/ ./ # buildkit   0B        buildkit.dockerfile.v0
<missing>      6 minutes ago   COPY app/spring-boot-loader/ ./ # buildkit      387kB     buildkit.dockerfile.v0
<missing>      6 minutes ago   COPY app/dependencies/ ./ # buildkit            63.5MB    buildkit.dockerfile.v0
<missing>      2 hours ago     WORKDIR /app                                    0B        buildkit.dockerfile.v0
<missing>      21 months ago   /bin/sh -c #(nop)  CMD ["jshell"]               0B
<missing>      21 months ago   /bin/sh -c set -eux;   arch="$(dpkg --print-…   322MB
<missing>      21 months ago   /bin/sh -c #(nop)  ENV JAVA_VERSION=17.0.2      0B
<missing>      21 months ago   /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0B
<missing>      21 months ago   /bin/sh -c #(nop)  ENV PATH=/usr/local/openj…   0B
<missing>      21 months ago   /bin/sh -c #(nop)  ENV JAVA_HOME=/usr/local/…   0B
<missing>      21 months ago   /bin/sh -c set -eux;  apt-get update;  apt-g…   4.88MB
<missing>      21 months ago   /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>      21 months ago   /bin/sh -c #(nop) ADD file:8b1e79f91081eb527…   80.4MB
zen@zen-code ~ %
```

НЕ будет кэшировано ~ 13.3 КВ

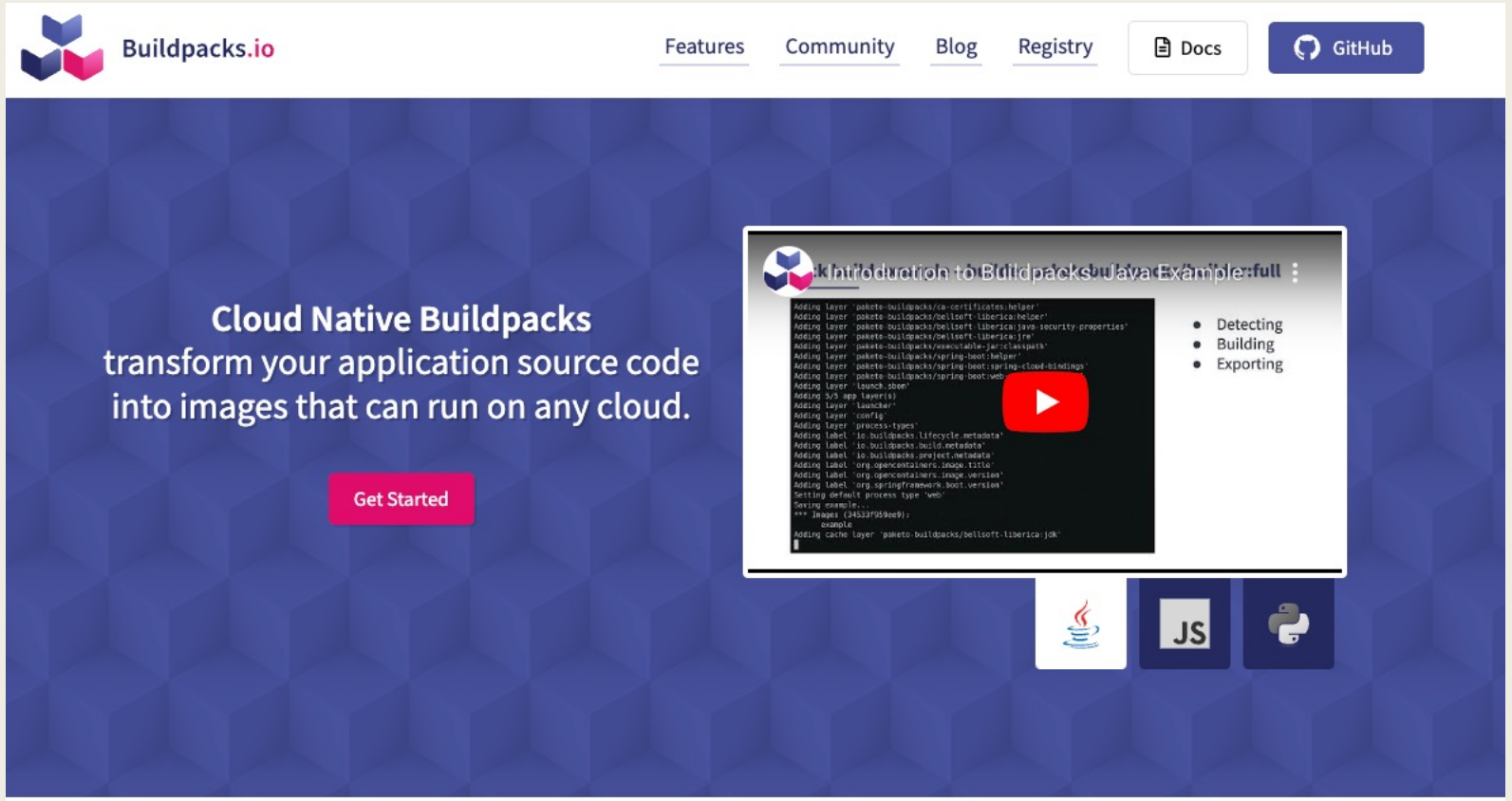
Будет кэшировано ~ 471.17 MB

Как перестать писать Dockerfiles и полюбить DevOps

>_ gradle bootBuildImage **>_ mvn spring-boot:build-image**



Spring-плагины для Gradle и Maven делают это через **Cloud Native Buildpacks** от CNCF

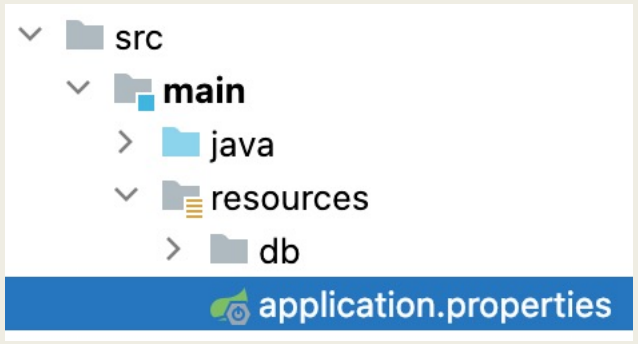


https://buildpacks.io/

УПРАВЛЕНИЕ SPRING APPLICATION PROPERTIES

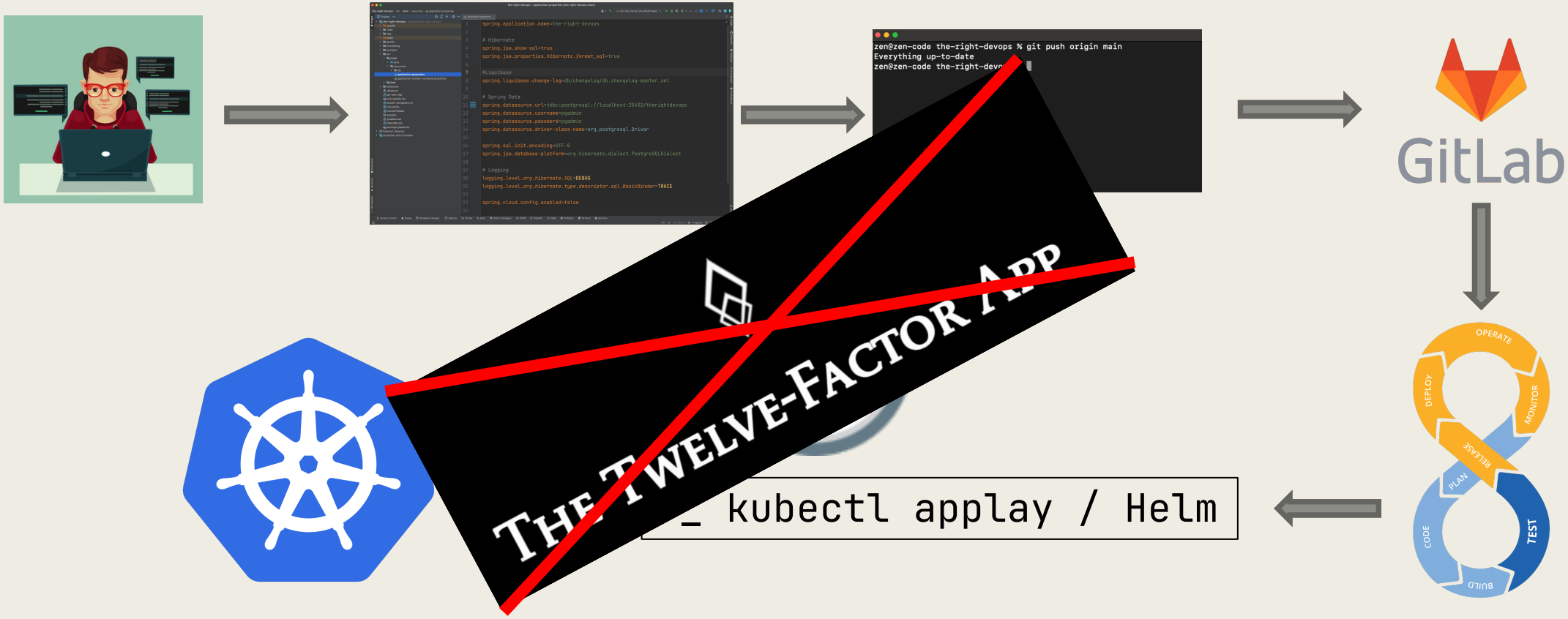
Kubernetes ConfigMap или Spring Cloud Config Server

Обычно Spring Boot приложение «настраивается» через application.properties файл




```properties
spring.application.name=the-right-devops

# Hibernate
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

#Liquibase
spring.liquibase.change-log=db/changelog/db.changelog-master.xml

# Spring Data
spring.datasource.url=jdbc:postgresql://postgres:5432/therightdevops
spring.datasource.username=pgadmin
spring.datasource.password=pgadmin
spring.datasource.driver-class-name=org.postgresql.Driver

spring.sql.init.encoding=UTF-8
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

# Logging
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

spring.cloud.config.enabled=false
```

И тут вы захотели переключить пропертю в приложении развернутом в Kubernetes



Spring предлагает Externalized Configuration

- Особый PropertySource порядок для разумного переопределения значений пропертей.
- Поиск пропертей: проперти-файлы, переменный среды, System.getProperties(), JNDI, контекст и конфигурация сервлетов, аргументы командной строки и прочее.

Spring предлагает Externalized Configuration

- Загрузка application.properties и application.yaml из разных источников.
- classpath root и пакет /config там же.
- Текущий каталог и каталог config в нём, подкаталоги внутри config.
- Либо укажите каталог через spring.config.location.

Здесь то и приходит на помощь ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: the-right-devops-config
data:
  application.properties: |
    spring.application.name=the-right-devops
    spring.jpa.show-sql=true
    spring.jpa.properties.hibernate.format_sql=true
    spring.liquibase.change-log=db/changelog/db.changelog-master.xml
    spring.datasource.url=jdbc:postgresql://localhost:25432/therightdevops
    spring.datasource.driver-class-name=org.postgresql.Driver
    spring.sql.init.encoding=UTF-8
    spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
    logging.level.org.hibernate.SQL=DEBUG
    logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

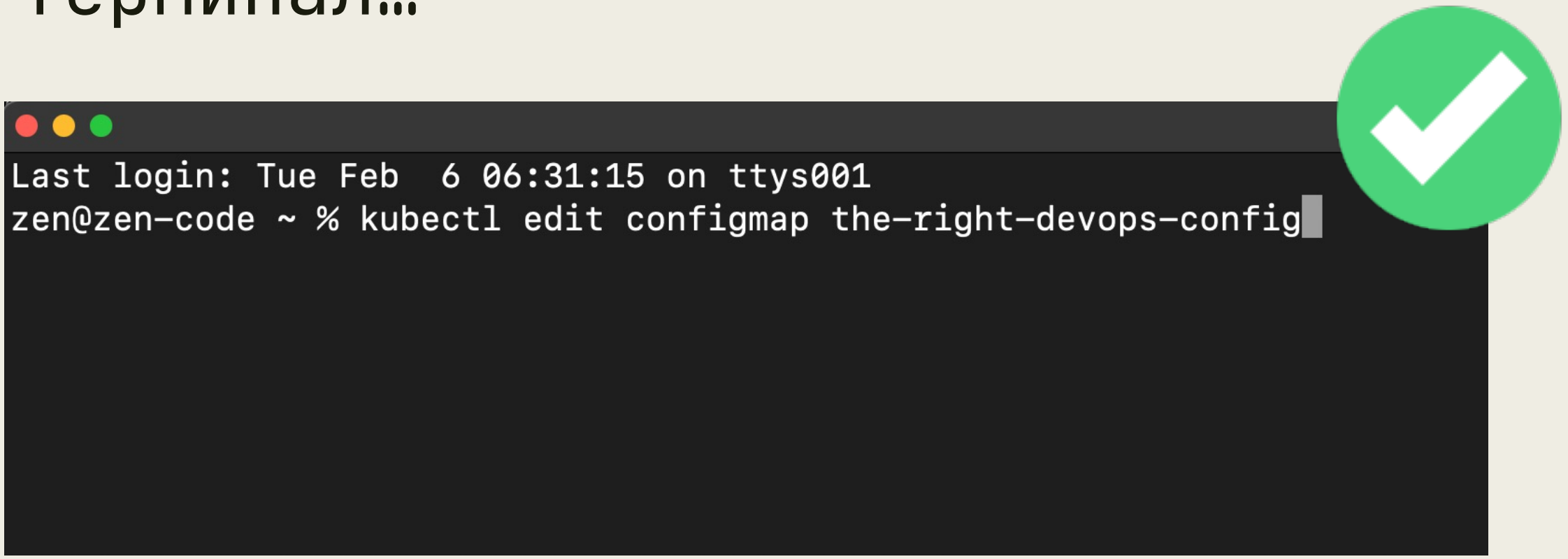
Монтируется ConfigMap через Deployment

```yaml
spec:
  containers:
    - name: right-devops
      image: localhost:5000/right-devops:1.0.0
      ports:
        - containerPort: 8080
      volumeMounts:
        - name: config-volume
          mountPath: /config
      env:
        - name: SPRING_DATASOURCE_USERNAME
          valueFrom:
            secretKeyRef:
              name: right-devops-secrets
              key: datasource.username
        - name: SPRING_DATASOURCE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: right-devops-secret
              key: datasource.password
  volumes:
    - name: config-volume
      configMap:
        name: right-devops-config
```

Монтируется ConfigMap через Deployment

```yaml
spec:
  containers:
    - name: right-devops
      image: localhost:5000/right-devops:1.0.0
      ports:
        - containerPort: 8080
      volumeMounts:
        - name: config-volume
          mountPath: /config
      env:
        - name: SPRING_DATASOURCE_USERNAME
          valueFrom:
            secretKeyRef:
              name: right-devops-secrets
              key: datasource.username
        - name: SPRING_DATASOURCE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: right-devops-secret
              key: datasource.password
  volumes:
    - name: config-volume
      configMap:
        name: right-devops-config
```

Теперь когда вам нужно будет изменить настройки Spring Boot приложения просто откройте терминал…



(убедитесь, что pod'ы вашего приложения были перезапущены!)

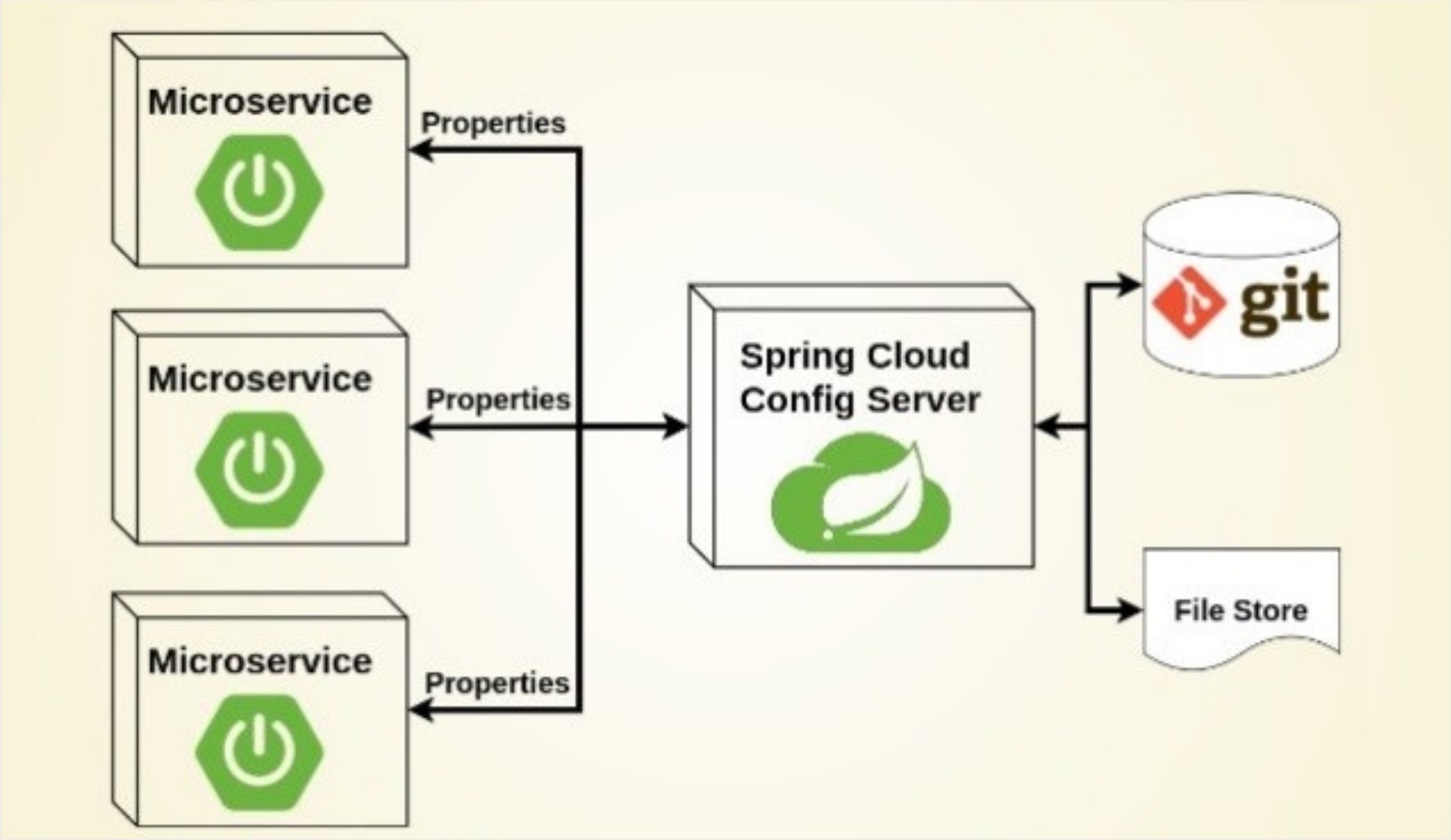
Однако Spring предлагает еще более изящное решение – **Spring Cloud Config**

- Клиент-серверное решение для распространения Spring Application Properties среди ваших сервисов.
- Config Server предоставляет HTTP API для получения Application Properties.

Однако Spring предлагает еще более изящное решение – **Spring Cloud Config**

- Config Client, который позволяет получать properties от Config Server.
- Возможность обновления Application Properties в реальном времени без перезапуска Spring Boot приложения (есть особенности).
- Источником для Config Server могут быть: *Git-репозиторий*, *файловая система*, *HashiCorp Vault*.

Схема работы Cloud Config



Поднять Config Server очень просто

```java
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

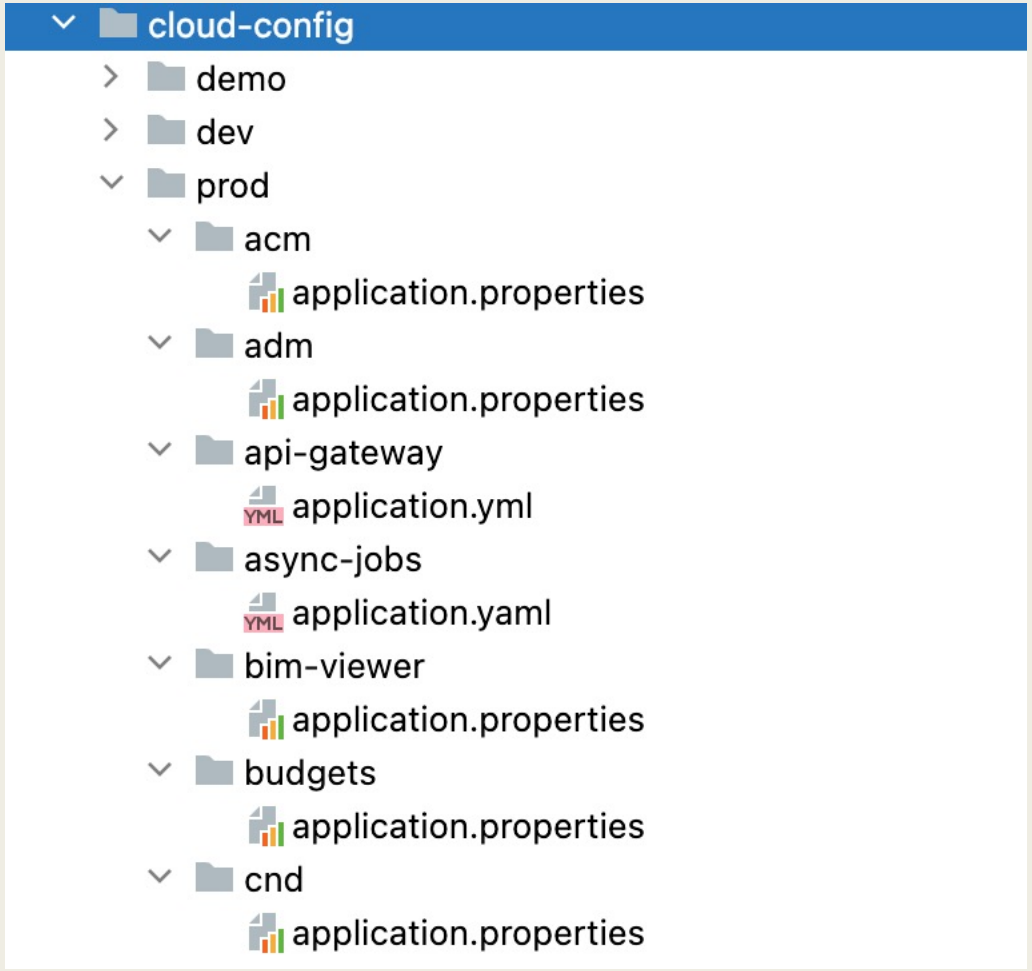
```properties
spring.application.name=config-server

spring.cloud.config.server.git.clone-on-start=true
spring.cloud.config.server.git.uri=${GITLAB_REPO_URI}
spring.cloud.config.server.git.username=${GITLAB_ACCESS_TOKEN_NAME}
spring.cloud.config.server.git.password=${GITLAB_ACCESS_TOKEN_PASSWORD}
spring.cloud.config.server.git.search-paths=dev/,dev/{application}
```

Поднять Config Server очень просто

```java
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

```properties
spring.application.name=config-server

spring.cloud.config.server.git.clone-on-start=true
spring.cloud.config.server.git.uri=${GITLAB_REPO_URI}
spring.cloud.config.server.git.username=${GITLAB_ACCESS_TOKEN_NAME}
spring.cloud.config.server.git.password=${GITLAB_ACCESS_TOKEN_PASSWORD}
spring.cloud.config.server.git.search-paths=dev/,dev/{application}
```

Разложите application.properties файлы в git-репозитории.



МИГРАЦИИ LIQUIBASE

Запуск в CI-пайплайне

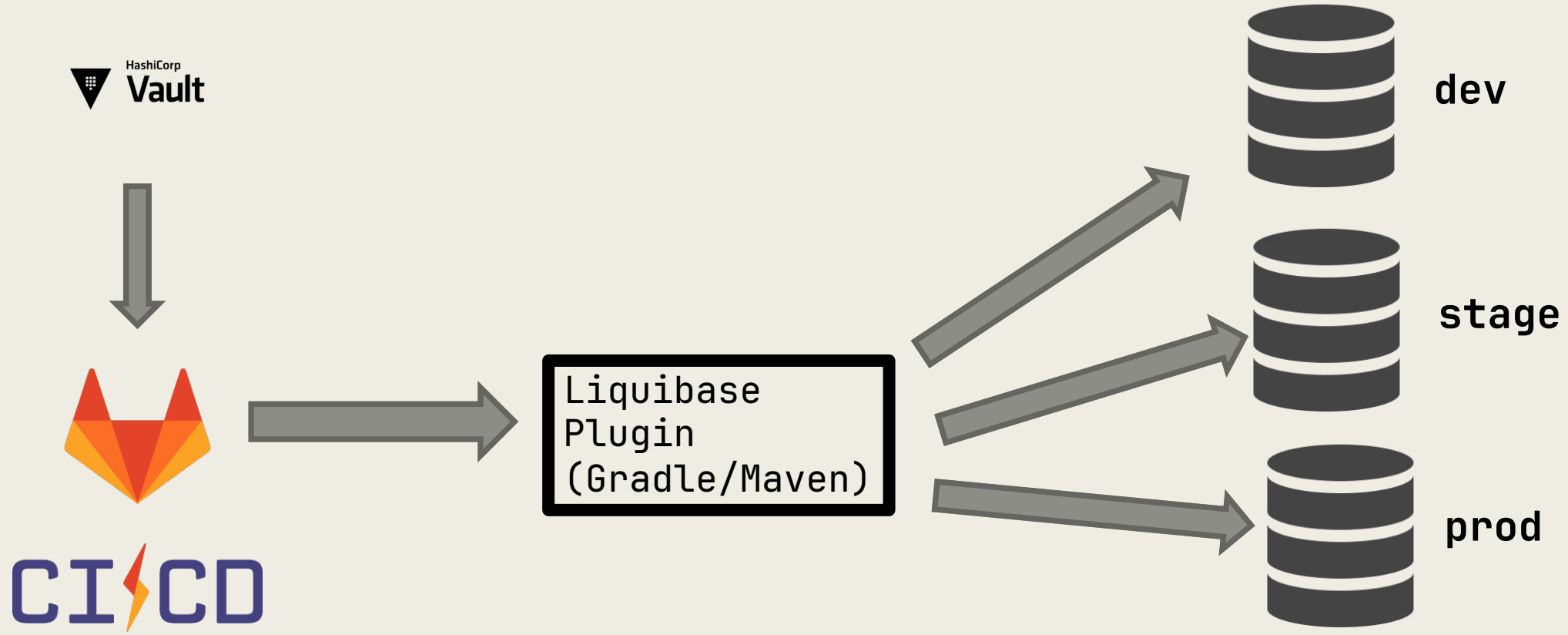
Обычно мы храним миграции БД в jar-архиве Spring Boot приложения. Что здесь не так.

- Чтобы накатить миграцию нужно делать сборку/релиз и ждать прохождение всего CI/CD пайплайна.
- Миграция может падать на разных окружения в зависимости от наполнения БД. Будем лишний раз шатать pod в kubernetes.

Обычно мы храним миграции БД в jar-архиве Spring Boot приложения. Что здесь не так.

- Запуск миграции при стартер Spring Boot приложения увеличивает время запуска.
- Накат миграции из Spring’а ограничивает возможности liquibase (если вы только не напишите лишний код).

Liquibase даёт нам gradle и maven плагины, чтобы делать хорошо.



Gradle Plugin

```
>_ gradle update
```

```
plugins {
    ...

    id("org.liquibase.gradle") version "2.2.0"

    ...
}

...

dependencies {

    ...

    liquibaseRuntime("org.liquibase:liquibase-core")
    liquibaseRuntime("ch.qos.logback:logback-core:1.2.3")
    liquibaseRuntime("ch.qos.logback:logback-classic:1.2.3")
    liquibaseRuntime("info.picocli:picocli:4.7.5")
    liquibaseRuntime("org.yaml:snakeyaml:1.33")
    liquibaseRuntime("org.postgresql:postgresql")

    ...

}

...

liquibase {
    activities.register("main") {
        val dbUrl = System.getenv("DB_URL")
        val dbUser = System.getenv("DB_USERNAME")
        val dbPassword = System.getenv("DB_PASSWORD")
        this.arguments = mapOf(
            "logLevel" to "info",
            "searchPath" to "src/main/resources/",
            "changeLogFile" to "db/changelog/db.changelog-master.xml",
            "url" to dbUrl,
            "username" to dbUser,
            "password" to dbPassword
        )
    }
    runList = "main"
}

...
```

Gradle Plugin

```
>_ gradle update
```

```
plugins {
    ...

    id("org.liquibase.gradle") version "2.2.0"

    ...
}

...

dependencies {

    ...

    liquibaseRuntime("org.liquibase:liquibase-core")
    liquibaseRuntime("ch.qos.logback:logback-core:1.2.3")
    liquibaseRuntime("ch.qos.logback:logback-classic:1.2.3")
    liquibaseRuntime("info.picocli:picocli:4.7.5")
    liquibaseRuntime("org.yaml:snakeyaml:1.33")
    liquibaseRuntime("org.postgresql:postgresql")

    ...

}

...

liquibase {
    activities.register("main") {
        val dbUrl = System.getenv("DB_URL")
        val dbUser = System.getenv("DB_USERNAME")
        val dbPassword = System.getenv("DB_PASSWORD")
        this.arguments = mapOf(
            "logLevel" to "info",
            "searchPath" to "src/main/resources/",
            "changeLogFile" to "db/changelog/db.changelog-master.xml",
            "url" to dbUrl,
            "username" to dbUser,
            "password" to dbPassword
        )
    }
    runList = "main"
}

...
```

Gradle Plugin

```
>_ gradle update
```

```
plugins {
    ...

    id("org.liquibase.gradle") version "2.2.0"

    ...
}

...

dependencies {

    ...

    liquibaseRuntime("org.liquibase:liquibase-core")
    liquibaseRuntime("ch.qos.logback:logback-core:1.2.3")
    liquibaseRuntime("ch.qos.logback:logback-classic:1.2.3")
    liquibaseRuntime("info.picocli:picocli:4.7.5")
    liquibaseRuntime("org.yaml:snakeyaml:1.33")
    liquibaseRuntime("org.postgresql:postgresql")

    ...

}

...

liquibase {
    activities.register("main") {
        val dbUrl = System.getenv("DB_URL")
        val dbUser = System.getenv("DB_USERNAME")
        val dbPassword = System.getenv("DB_PASSWORD")
        this.arguments = mapOf(
            "logLevel" to "info",
            "searchPath" to "src/main/resources/",
            "changeLogFile" to "db/changelog/db.changelog-master.xml",
            "url" to dbUrl,
            "username" to dbUser,
            "password" to dbPassword
        )
    }
    runList = "main"
}

...
```

Gradle Plugin

```
>_ gradle update
```

```
plugins {
    ...

    id("org.liquibase.gradle") version "2.2.0"

    ...
}

...

dependencies {

    ...

    liquibaseRuntime("org.liquibase:liquibase-core")
    liquibaseRuntime("ch.qos.logback:logback-core:1.2.3")
    liquibaseRuntime("ch.qos.logback:logback-classic:1.2.3")
    liquibaseRuntime("info.picocli:picocli:4.7.5")
    liquibaseRuntime("org.yaml:snakeyaml:1.33")
    liquibaseRuntime("org.postgresql:postgresql")

    ...

}

...

liquibase {
    activities.register("main") {
        val dbUrl = System.getenv("DB_URL")
        val dbUser = System.getenv("DB_USERNAME")
        val dbPassword = System.getenv("DB_PASSWORD")
        this.arguments = mapOf(
            "logLevel" to "info",
            "searchPath" to "src/main/resources/",
            "changeLogFile" to "db/changelog/db.changelog-master.xml",
            "url" to dbUrl,
            "username" to dbUser,
            "password" to dbPassword
        )
    }
    runList = "main"
}

...
```

Maven Plugin

```
>_ mvn
liquibase:update
```

```xml
<properties>
    <liquibase.version>4.21.1</liquibase.version>
</properties>

<dependencies>

    <dependency>
        <groupId>org.liquibase</groupId>
        <artifactId>liquibase-core</artifactId>
        <version>${liquibase.version}</version>
    </dependency>

</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.liquibase</groupId>
            <artifactId>liquibase-maven-plugin</artifactId>
            <version>${liquibase.version}</version>
            <configuration>
                <changeLogFile>db/changelog/db.changelog-master.xml</changeLogFile>
                <searchPath>src/main/resources/</searchPath>
                <url>${env.DB_URL}</url>
                <username>${env.DB_USERNAME}</username>
                <password>${env.DB_PASSWORD}</password>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Maven Plugin

```
>_ mvn
liquibase:update
```

```xml
<properties>
    <liquibase.version>4.21.1</liquibase.version>
</properties>

<dependencies>

    <dependency>
        <groupId>org.liquibase</groupId>
        <artifactId>liquibase-core</artifactId>
        <version>${liquibase.version}</version>
    </dependency>

</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.liquibase</groupId>
            <artifactId>liquibase-maven-plugin</artifactId>
            <version>${liquibase.version}</version>
            <configuration>
                <changeLogFile>db/changelog/db.changelog-master.xml</changeLogFile>
                <searchPath>src/main/resources/</searchPath>
                <url>${env.DB_URL}</url>
                <username>${env.DB_USERNAME}</username>
                <password>${env.DB_PASSWORD}</password>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Теперь мы можем добавить запуск миграции в GitLab CI

```yaml
.migrate-db:
  stage: migrate-db
  image: maven-3-8-4-openjdk-17-master
  variables:
    VAULT_AUTH_PATH: "id_token_jwt"
  id_tokens:
    VAULT_ID_TOKEN:
      aud: https://zen-code.ru
  script:
    - mvn -version
    - mvn clean package -DskipTests
    - mvn liquibase:status -Dliquibase.verbose=true
    - mvn liquibase:updateSQL -Dliquibase.verbose=true -Dliquibase.logging=debug
    - mvn liquibase:update -Dliquibase.verbose=true -Dliquibase.logging=debug
  when: manual
```

Теперь мы можем добавить запуск миграции в GitLab CI

```yaml
.migrate-db:
  stage: migrate-db
  image: maven-3-8-4-openjdk-17-master
  variables:
    VAULT_AUTH_PATH: "id_token_jwt"
  id_tokens:
    VAULT_ID_TOKEN:
      aud: https://zen-code.ru
  script:
    - mvn -version
    - mvn clean package -DskipTests
    - mvn liquibase:status -Dliquibase.verbose=true
    - mvn liquibase:updateSQL -Dliquibase.verbose=true -Dliquibase.logging=debug
    - mvn liquibase:update -Dliquibase.verbose=true -Dliquibase.logging=debug
  when: manual
```

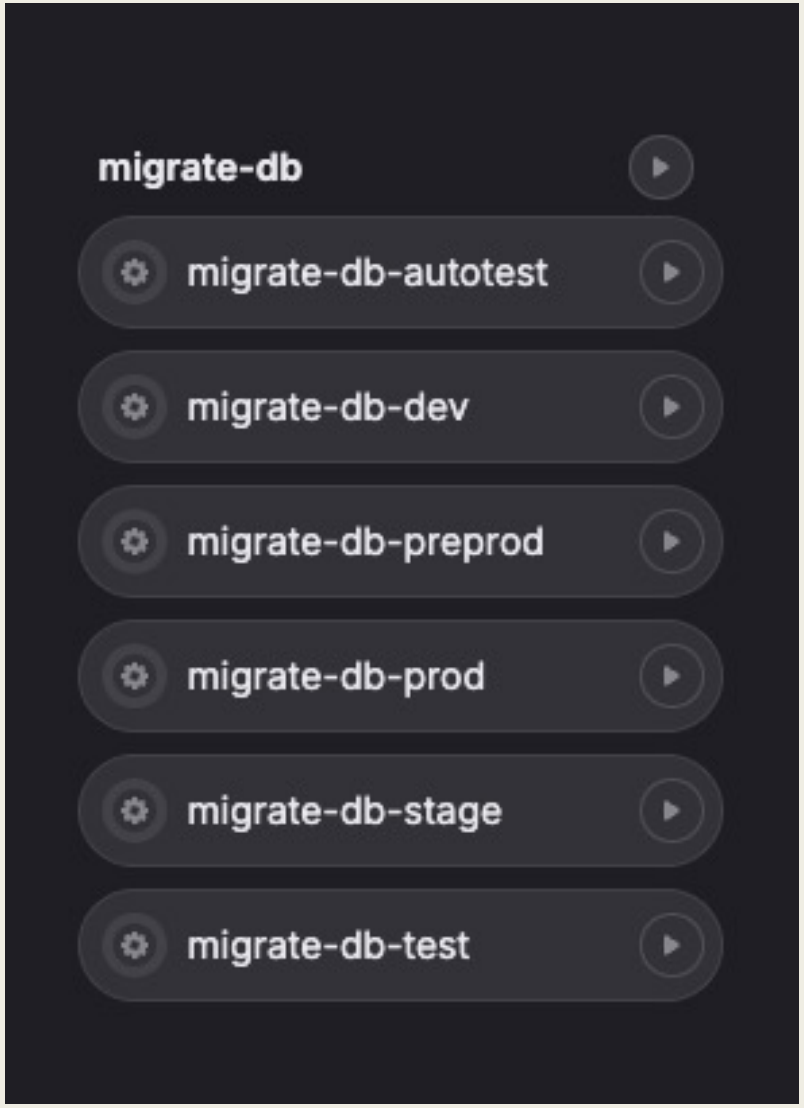
Теперь мы можем добавить запуск миграции в GitLab CI

```yaml
migrate-db-dev:
  extends: .migrate-db
  variables:
    DB_URL: "jdbc:postgresql://localhost:25432/rightdevops"
    DB_USERNAME: pg_admin
    <<: *migrate-db-vault-public-env
  secrets:
    LIQUIBASE_PASSWORD:
      vault:
        engine:
          name: kv-v2
          path: gitlab
        field: SPRING_DATASOURCE_PASSWORD
        path: zen/dev/env
      file: false
```

Теперь мы можем добавить запуск миграции в GitLab CI

```yaml
migrate-db-dev:
  extends: .migrate-db
  variables:
    DB_URL: "jdbc:postgresql://localhost:25432/rightdevops"
    DB_USERNAME: pg_admin
    <<: *migrate-db-vault-public-env
  secrets:
    LIQUIBASE_PASSWORD:
      vault:
        engine:
          name: kv-v2
          path: gitlab
        field: SPRING_DATASOURCE_PASSWORD
        path: zen/dev/env
      file: false
```

Получим такие кнопочки в GitLab CI



ВЫВОДЫ

DevOps процессы к успеху пришли (ну почти…)

Как делать хорошо

- Разработчикам изучать DevOps инструменты (Docker, Kubernetes, Linux/bash, CI-серверы).
- Инженерам по эксплуатации изучать документацию фреймворков и важных библиотек.
- Коллаборация разработки и эксплуатации сделает вас ~~богатыми~~ счастливыми.
- Проверяйте астрологический календарь-гороскоп для деплоя в продакшн.

(Он действительно существует https://deployhoroscope.ru/)

Хотелось рассказать, но не хватило места ~~на диске~~

- Детальный разбор создания docker images с помощью Spring Boot Maven/Gradle Plugin.
- Нативные feature toggles со Spring Boot Conditional Annotations.
- XX-флаги JVM и Kubernetes.
- Разработка кастомных метрик с помощью Micrometer для Prometheus и Grafana. Создание дашбордов со своими метриками в Grafana.
- Сбор логов в kubernetes и sidecar контейнеры.

”Весёлые DevOps’ы радуются успешному релизу..”

СПАСИБО ЗА ВНИМАНИЕ! ПОКА!